

Frank Puppe

# Systematic Introduction to Expert Systems

## Knowledge Representations and Problem-Solving Methods

With 129 Figures

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Frank Puppe
Universität Würzburg
Institut für Informatik
Lehrstuhl für Künstliche Intelligenz
Am Hubland
W-8700 Würzburg
Germany

ISBN-13: 978-3-642-77973-2 e-ISBN-13: 978-3-642-77971-8
DOI: 10.1007/978-3-642-77971-8

Library of Congress Cataloging-in-Publication Data available

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other ways and storage in data banks. Duplication of this publication or parts thereof is only permitted under the provisions of the German Copyright Law of September 9, 1965, in its current version, and a permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1993
Softcover reprint of the hardcover 1st edition 1993

Typesetting: Camera-ready by Peter Ryder

45/3140-5 4 3 2 1 0 - Printed on acid-free paper

# Preface

With the growing use of expert systems it is becoming increasingly apparent that the central problem is the acquisition and maintenance of knowledge. At present the most promising approach seems to be model-based knowledge acquisition, in which expert knowledge is not simply accumulated ("rule spaghetti") but structured according to some suitable "strong" problem-solving method. Typical features of strong problem-solving methods are: specialization for a particular problem class, a regular structure of the required knowledge, and restriction of the function of individual knowledge structures. Rule processing with forward or backward chaining, for example, is not a strong problem-solving method, although the knowledge structure as a set of similar rules appears regular at first sight, since the rules show great differences in syntax and expressive power and can take on many different functions, even the control of other rules as "meta-rules".

The concentration on strong problem-solving methods results from the experience and difficulties of previous research: The first phase, up to the beginning of the 1970s, was characterized by the search for general, i.e., weak problem-solving methods, symbolized by the development of the "General Problem Solver" with its unfulfilled promises. Thereafter the prevailing opinion changed to the other extreme, that good problem-solving ability requires only a great deal of expert knowledge ("In the knowledge lies the power"), while the problem-solving method itself seemed insignificant ("Any inference strategy works"). Practical experience has shown, however, that the acquisition of a large amount of expert knowledge requires suitable models of knowledge structuring, such as strong problem-solving methods. Their identification and realization in problem-specific expert system tools are expected to form the main motivation of a third phase of expert system research, which has been gaining impetus rapidly since the end of the 1980s.

The theory of strong problem-solving methods and their realization in tools is still very incomplete. Furthermore, it is unlikely that there will be strong problem-solving methods for all problem classes. Where they are found, however, they are expected to simplify the development of expert systems significantly. As long as the problem-solving methods are easy to follow for experts, well-structured expert systems are also suitable as a new medium for transmitting knowledge. Compared with other knowledge media, such as textbooks, the knowledge is more easily checked or criticized and can be presented to the knowledge receiver in a variety of flexible forms.

The aim of this introduction is to give a systematic review of the state of the art with regard to the identification and realization of strong problem-solving

methods and to indicate the prospects for their integration. It is based on experience gained by the author in consultations with users in numerous knowledge engineering courses and individual discussions, in which the main topic was usually the identification of problem-solving methods and knowledge representations in the different domains, on the results of our development and testing of tools for some of the problem-solving methods and for graphical knowledge acquisition techniques, and on an analysis of the relevant literature, which makes no claim to be exhaustive, however, on account of the great variety of more or less strong problem-solving methods and the rapid development of the field.

Part I contains a general introduction and a characterization of expert systems. The main feature of expert systems is the clear separation which is made between problem-solving strategies and knowledge. It is this separation which enables the knowledge contents to be changed easily and the solutions to be explained by indicating the knowledge used for their derivation.

Part II describes the basic techniques of knowledge representation commonly used in expert systems and the associated inference strategies: logic as the reference standard for the other techniques, rules with forward and backward chaining, frames with inheritance and attached procedures, constraints with local propagation, probabilistic reasoning with evidence values, non-monotonic reasoning with dependency networks, and temporal reasoning with temporal data bases.

In Part III we discuss various approaches to the division of problems into classes, give a review of our own scheme and describe the uniform basic structure for the representation of the problem-solving methods.

In Parts IV, V, and VI the main problem classes diagnostics (classification), construction and simulation are treated in detail. First their domains, problem types, problem features and problem-solving methods are analyzed. Then, for each of the three to six problem-solving methods in each main class, the basic idea, the object types and their attribute structures, the basic methods of knowledge manipulation, the graphical primitives for knowledge acquisition, the state of the art, and as far as possible an example of a corresponding tool from our research group are given.

Finally, in Part VII, some aspects of the integration of problem-solving methods are discussed and illustrated with examples of their application.

This work would not have been possible without the favorable climate of the University of Karlsruhe. For the development of the tools in their thesis work, I would like to thank Thomas Dyckhoff (ExAP), Ute Gappa (CLASSIKA), Klaus Goos (FEMO), Christian Hestermann (CcC), Rüdiger Hnyk (SIMUL), Susanne Mütter (FAKTA), and Karsten Poeck (COKE). For valuable suggestions for the revision of earlier manuscript versions I thank above all Wilfried Brauer, Peter Deussen, Ute Gappa, Gerhard Goos, Klaus Goos, Joachim Hertzberg, Peter Mertens, Karsten Poeck, Angi Voss, and Hans Voss. Further, special thanks are due to Peter Ryder for the excellent English translation.

*Frank Puppe*

# Table of Contents


**I Introduction** ..... 1

1 Characterization and History of Expert Systems ..... 3
- 1.1 Characterization ..... 3
- 1.2 History ..... 6

2 Programming Languages and Expert System Tools ..... 9
- 2.1 Knowledge Acquisition Problems ..... 9
- 2.2 Knowledge Acquisition with Specialized Programming Environments ..... 10
- 2.3 Architecture of Expert Systems ..... 14

3 Use and Usability of Expert Systems ..... 17
- 3.1 Potential Benefits and Modes of Use ..... 18
- 3.2 Expert Systems as Knowledge Media ..... 21
- 3.3 Criteria for Expert System Domains ..... 23
- 3.4 Summary ..... 24

**II Basic Techniques of Knowledge Representation** ..... 27

4 Logic ..... 29
- 4.1 Predicate Logic ..... 29
- 4.2 Properties of Logic Calculi ..... 30
- 4.3 PROLOG ..... 32
- 4.4 Summary ..... 33

5 Rules ..... 35
- 5.1 Forward Chaining ..... 36
- 5.2 Backward Chaining ..... 38
- 5.3 Complexity of the Precondition ..... 39
- 5.4 Rule Structuring ..... 41
- 5.5 Summary ..... 42

6 Objects/Frames ..... 43
- 6.1 Example: FRL ..... 44
- 6.2 Active Objects ..... 45
- 6.3 Cognitive Meaning of Frames ..... 46

6.4 Automatic Classification in KL-ONE Languages ..... 47
6.5 Frames and Problem-Solving Types ..... 49
6.6 Summary ..... 50

7 Constraints ..... 51
7.1 Types of Constraints and Propagation Algorithms ..... 52
7.2 Solution of a Non-Trivial Constraint Problem ..... 53
7.3 Discussion of the Solution ..... 55
7.4 Formal Characterization of Simple Constraint Systems ..... 55
7.5 Summary ..... 56

8 Probabilistic Reasoning ..... 57
8.1 Bayes' Theorem ..... 58
8.2 The Dempster-Shafer Theory ..... 61
8.3 The INTERNIST Model ..... 64
8.4 The MYCIN Model ..... 66
8.5 The MED1 Model ..... 67
8.6 Summary ..... 68

9 Non-Monotonic Reasoning ..... 71
9.1 JTMS with Direct Justifications ..... 73
9.2 ATMS with Basic Assumptions as Justifications ..... 76
9.3 Summary ..... 78

10 Temporal Reasoning ..... 79
10.1 Exact Time Relations: VM and MED2 ..... 81
10.2 Inexact Quantitative Time Relations: TMM of McDermott . 82
10.3 Qualitative Relations: the Time Calculus of Allen ..... 83
10.4 Summary ..... 85

**III Problem Classes and Problem-Solving Methods** ..... 87

11 Previous Approaches to Problem Classification ..... 89
11.1 The Approach of Stefik et al. and Hayes-Roth et al. ..... 90
11.2 The Approach of Clancey ..... 91
11.3 The Approach of Breuker et al. ..... 94
11.4 The Approach of Chandrasekaran ..... 96
11.5 The Approach of Harmon ..... 97
11.6 The Approach of McDermott ..... 98
11.7 Discussion ..... 99

12 Principles of Problem-Solving Methods ..... 101
12.1 Survey of Problem-Solving Methods ..... 103
12.2 Restriction of Use of Variables ..... 103
12.3 Structure of Knowledge Representation ..... 107
12.4 Structure of Knowledge Manipulation ..... 108
12.5 Structure of Knowledge Acquisition ..... 108

**IV Classification** ..... 113

13 Survey of the Problem-Solving Type Classification ..... 115
13.1 Domains ..... 115
13.2 Problem Types ..... 120
13.3 Analysis of Problem Characteristics ..... 121
13.4 Problem-Solving Methods ..... 123

14 Simple Classification ..... 128
14.1 Decision Trees ..... 128
14.2 Decision Tables ..... 129
14.3 State of the Art ..... 130

15 Heuristic Classification ..... 131
15.1 Knowledge Representation ..... 135
15.2 Knowledge Manipulation ..... 136
15.3 Knowledge Acquisition ..... 138
15.4 State of the Art ..... 140
15.5 Example: CLASSIKA ..... 140

16 Heuristic Classification: Additional Mechanisms ..... 149
16.1 Treatment of Uncertain Data and Solution Classes ..... 149
16.2 Treatment of Subjective Observations ..... 151
16.3 Recognition of False Observations ..... 152
16.4 Treatment of Time-Dependent Data ..... 152
16.5 Belief Revision ..... 153
16.6 Treatment of Parametrized Data and Solutions ..... 153
16.7 Treatment of Multiple Solutions ..... 154
16.8 Combined Recommendations for Multiple Solutions ..... 155

17 Set-Covering Classification ..... 156
17.1 Knowledge Representation ..... 159
17.2 Knowledge Manipulation ..... 161
17.3 Knowledge Acquisition ..... 165
17.4 State of the Art ..... 165
17.5 Example: FEMO ..... 167

18 Functional Classification ..... 170
18.1 Knowledge Representation ..... 174
18.2 Knowledge Manipulation ..... 176
18.3 Knowledge Acquisition ..... 178
18.4 State of the Art ..... 178
18.5 Example: SIMUL ..... 179

19 Statistical Classification ..... 183
19.1 Knowledge Representation ..... 183
19.2 Knowledge Manipulation ..... 184

19.3 Knowledge Acquisition ... 185
19.4 State of the Art ... 185
19.5 Variants of Bayes' Theorem ... 186
19.6 Example: FAKTA ... 188

20 Case-Comparing Classification ... 191
20.1 Knowledge Representation ... 194
20.2 Knowledge Manipulation ... 198
20.3 Knowledge Acquisition ... 199
20.4 State of the Art ... 200
20.5 Example: CcC ... 201

**V Construction** ... 205

21 Review of the Problem-Solving Type Construction ... 207
21.1 Domains ... 207
21.2 Problem Types ... 209
21.3 Analysis of Problem Characteristics ... 209
21.4 Problem-Solving Methods ... 212

22 Skeletal Construction ... 216
22.1 Knowledge Representation ... 217
22.2 Knowledge Manipulation ... 219
22.3 Knowledge Acquisition ... 220
22.4 State of the Art ... 221
22.5 Variants of Skeletal Construction: Generate and Test ... 221
22.6 Heuristic Classification and Skeletal Construction ... 222

23 Propose and Revise ... 223
23.1 Knowledge Representation ... 224
23.2 Knowledge Manipulation ... 225
23.3 Knowledge Acquisition ... 226
23.4 State of the Art ... 227
23.5 Variant for Therapy Planning by Parameter Adjustment ... 227
23.6 Heuristic Classification and Propose and Revise ... 228

24 Propose and Exchange ... 229
24.1 Knowledge Representation ... 230
24.2 Knowledge Manipulation ... 234
24.3 Knowledge Acquisition ... 234
24.4 State of the Art ... 235
24.5 Example: COKE and REST ... 236

25 Least-Commitment Strategy ... 240
25.1 Knowledge Representation ... 241
25.2 Knowledge Manipulation ... 242
25.3 Knowledge Acquisition ... 243

25.4 State of the Art ..... 243
25.5 Example: ExAP ..... 244

26 Model-Based Planning ..... 249
26.1 Non-Hierarchical Planning ..... 250
26.2 Hierarchical Planning ..... 251
26.3 Non-Linear Planning ..... 252

27 Case-Comparing Construction ..... 253

28 Partial Integration of Construction Methods ..... 255
28.1 Knowledge Representation ..... 257
28.2 Knowledge Manipulation ..... 259
28.3 Knowledge Acquisition ..... 261
28.4 State of the Art ..... 261

**VI Simulation** ..... 263

29 Review of the Problem-Solving Type Simulation ..... 265
29.1 Domains ..... 265
29.2 Classification According to Problem Types ..... 266
29.3 Analysis of Problem Characteristics ..... 266
29.4 Problem-Solving Methods ..... 267

30 Single-Phase Simulation ..... 269

31 Numerical Multiple-Phase Simulation ..... 271
31.1 Knowledge Representation ..... 273
31.2 Knowledge Manipulation ..... 275
31.3 Knowledge Acquisition ..... 276
31.4 State of the Art ..... 276

32 Qualitative Multiple-Phase Simulation ..... 278
32.1 Knowledge Representation ..... 279
32.2 Knowledge Manipulation ..... 281
32.3 Knowledge Acquisition ..... 282
32.4 State of the Art ..... 282

**VII Integration of Problem-Solving Methods** ..... 285

33 Basic Ideas for the Integration of Problem-Solving Methods ..... 287
33.1 Domains ..... 288
33.2 Levels of Integration ..... 289
33.3 References to Objects in the Knowledge Representation ..... 289
33.4 Problem Solving ..... 289
33.5 Knowledge Acquisition ..... 290
33.6 Explainability ..... 291

33.7 Data Collection ..... 291
33.8 Programming ..... 292
33.9 Survey of the Integration of Problem-Solving Methods ..... 292

34 Integration of Classification Methods ..... 295
34.1 Strengths and Weaknesses of the Problem-Solving Methods . 296
34.2 Knowledge Representation ..... 297
34.3 Knowledge Manipulation ..... 299
34.4 Knowledge Acquisition ..... 303
34.5 Explainability ..... 306
34.6 Data Collection ..... 310
34.7 Programming ..... 311
34.8 State of the Art ..... 311

35 Aspects of the Overall Integration ..... 314
35.1 Survey ..... 314
35.2 Decision-Making in Medicine ..... 316
35.3 Tendering, Configuration and Maintenance of Technical Systems ..... 317
35.4 Job and Production Planning and Quality Control ..... 320

**Appendix: Survey of Knowledge Representation Formalisms** ..... 322

**Glossary** ..... 326

**References** ..... 332

**System Index** ..... 347

**Subject Index** ..... 349

# Part I

# Introduction

# 1. Characterization and History of Expert Systems

## 1.1 Characterization

Expert systems are programs for reconstructing the expertise and reasoning capabilities of qualified specialists within limited domains. The preliminary basic assumption is that experts construct their solutions from single pieces of knowledge which they select and apply in a suitable sequence. Hence expert systems require detailed information about the domain and the strategies for applying this information to problem solving. In order to construct an expert system, the knowledge must therefore be *formalized, represented* in the computer and *manipulated* according to some problem-solving method. However, since few people have both the detailed expert knowledge and programming skills, a division of labor is desirable. This could, for example, take the following form: An expert and a computer scientist agree on a common representation of the knowledge, the expert formulates his knowledge accordingly, and the computer scientist programs the appropriate problem-solving methods. By this means both can contribute their skills in the best possible way. It is perhaps even more important to create a clear interface between the specialist knowledge for the problem in hand and the general problem-solving methods. This makes an expert system much more amenable to modification than most conventional programs. This distinction between expert knowledge and the problem-solving method, possible in many domains, is a characteristic feature of the architecture of expert systems. The only difference between expert systems and knowledge-based systems is that the former derive their knowledge from experts (Fig. 1.1).

To a first approximation, problem-solving methods are highly parametrized algorithms. As with normal algorithms, some of the parameters represent the input data of different problems within one domain, e.g., the data from different defective automobiles in an automobile diagnostics program. Other parameters enable the problem solving method to be adapted to different domains, e.g., automobile diagnostics, medical diagnostics or the identification of fungi. We call such second order parameters "knowledge" if they are meaningful to human beings.[1]

[1]If one wishes to be independent of a special problem-solving method and its concrete representation, the knowledge definition can be generalized in such a way that there must be at least one problem-solving method which can evaluate the "knowledge". This corresponds to

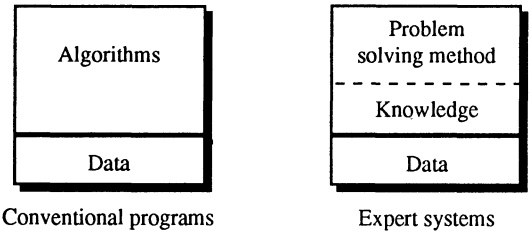



**Figure 1.1** Comparison of conventional programs, knowledge-based systems and expert systems

Since the knowledge of expert systems should come from experts, the problem-solving methods need not be identical, but a certain similarity is desirable.

One representation of knowledge which favors the distinction between knowledge and problem-solving methods consists of rules of the form "If X then Y." Figure 1.2 illustrates the difference between the traditional instruction-based program style and the rule-based style frequently used in expert systems. In the instruction-based program style primitive operations are performed sequentially in the order fixed by the programmer, whereas in the rule-based program style the expert merely lays down what is to be done in a certain situation, and the rule interpreter decides the order in which the rules are applied for problem solving. The control flow in instruction-based programs is clear but rigid, while rule-based programs display just the reverse properties.

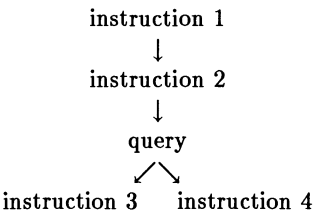
Almost every expert can immediately name a few rules from his field, and an astounding performance is often attained by combining relatively few rules. For these reasons the rule-based programming style has become very popular for expert systems. A disadvantage, however, is the complex control flow, which make large rule systems difficult to handle ("rule spaghetti"). Expert systems therefore frequently use another form of knowledge representation, in which the leading role is played by objects, not rules.

Every specialist domain has its own technical language, in which all important objects and states of the domain are named. In acquiring knowledge these concepts are collected, and their definitions and interrelations formalized. From here it is only a small step to the object-based programming style, in which the objects and states of the domain are regarded as agents which can receive messages and send them to other objects and states. For example, one could envisage a diagnostics expert system in which symptoms and diagnoses are represented as agents which send each other messages about their existence. When a diagnosis has received enough positive messages from its symptoms it establishes itself and sends messages in its turn to other diagnoses and symptoms.

---

the functional definition of knowledge by Newell [82] as a means of achieving ends. Dreyfus and Dreyfus [86] distinguish between communicable "know-what" and non-communicable "know-how", both of which are used by experts for problem solving, whereas only know-what can enter into programs. Our concept of knowledge corresponds to know-what, while know-how is equivalent to skill.

1. Instruction-based programming style:
   program = sequence of instructions and queries




   The programmer determines what is done and in what order it is done.

2. Rule-based programming style:
   program = set of rules and rule interpreters

   Rule 1: If situation X1, then action Y1.
   Rule 2: If situation X2, then action Y2.
   Rule 3: If situation X3, then action Y3.

   The expert determines what is done, and the rule interpreter determines the order.

**Figure 1.2** The difference between instruction-based and rule-based programming styles

From a higher level, the rule-based and object-based representations of knowledge can be regarded as two different ways of viewing the same basic network of objects and relationships, in which either the nodes or the links are taken to be the more "important".

So far we have characterized expert systems according to their aim, i.e., the simulation of experts, and their techniques, i.e., the separation of knowledge and problem-solving methods. The rule-based and object-based program styles are examples for the realization of such a separation. Let us now expand this characterization by comparison with other program types and knowledge media. The distinction between domain-specific details and general process strategies is not unknown in computer science, since it is also practiced in the development of system software such as compiler compilers or development languages for operating systems. However, the "knowledge representations" and "problem-solving methods" of system software development are extremely domain-specific and designed for computer scientists familiar with this area. In addition, these exact, clearly defined fields of application differ considerably from the "diffuse" domains of expert systems:

1. The knowledge and data in the domains of expert systems are usually uncertain, incomplete or time-dependent.

2. Expert systems are typically written for problem fields which are not clearly understood, at least at the beginning of the project.

Expert systems differ from data bases in that their knowledge is not simply readable as data, but can also be used for the solution of different types of problems. "Knowledge" is thus distinguished from "data" by the fact that it is coupled to instructions for its use interpretable by a computer. It follows from this, of course, that the transition from data to knowledge is a gradual one.

Another way of characterizing expert systems is to list the necessary properties. These include:

| | |
|---|---|
| Transparency: | Expert systems can explain their solution by quoting the knowledge used. |
| Flexibility: | Single pieces of knowledge can easily be added, changed or deleted. |
| User friendliness: | The use of expert systems requires no previous knowledge of programming languages (neither for the end user nor for the expert). |
| Competence: | On account of the expert knowledge, expert systems have a high problem-solving capability in their domain. |

Figure 1.3 summarizes the different ways of characterizing expert systems. Although each characterization is relatively vague on its own, their combination nevertheless gives a sufficiently clear picture and enables a distinction to be made from other programs. A simple test for determining whether a program is an expert system is to check its attributes, i.e., test the system with a "normal" problem (regularly solvable by experts), have the solution explained, and make small changes in the knowledge base.

## 1.2 History

The history of problem-solving programs may be regarded as a development from universal to specific problem solvers. The search and game programs of the 1960s [Barr 81, Chap. 2] were generally aimed at solving clearly formulated search problems, e.g., the proof of mathematical theorems, or chess problems. For this purpose, general, efficient search strategies were developed, e.g.

- the means-ends analysis (difference analysis) in the "General Problem Solver" [Newell 72], which is also used in planning (see Chap. 26),
- the alpha-beta strategy in games, in which only those moves or countermoves are considered which can be more favorable or less favorable, respectively, than the moves already analyzed [Nilsson 82, Sect. 3.4]
- the A* algorithm for heuristic searching, in which an estimate of the effort still needed to reach the aim is included in the evaluation function for selecting the next step [Nilsson 82, Sect. 2.4],
- special resolution methods for the proof of theorems [Nilsson 82, Chap. 5].

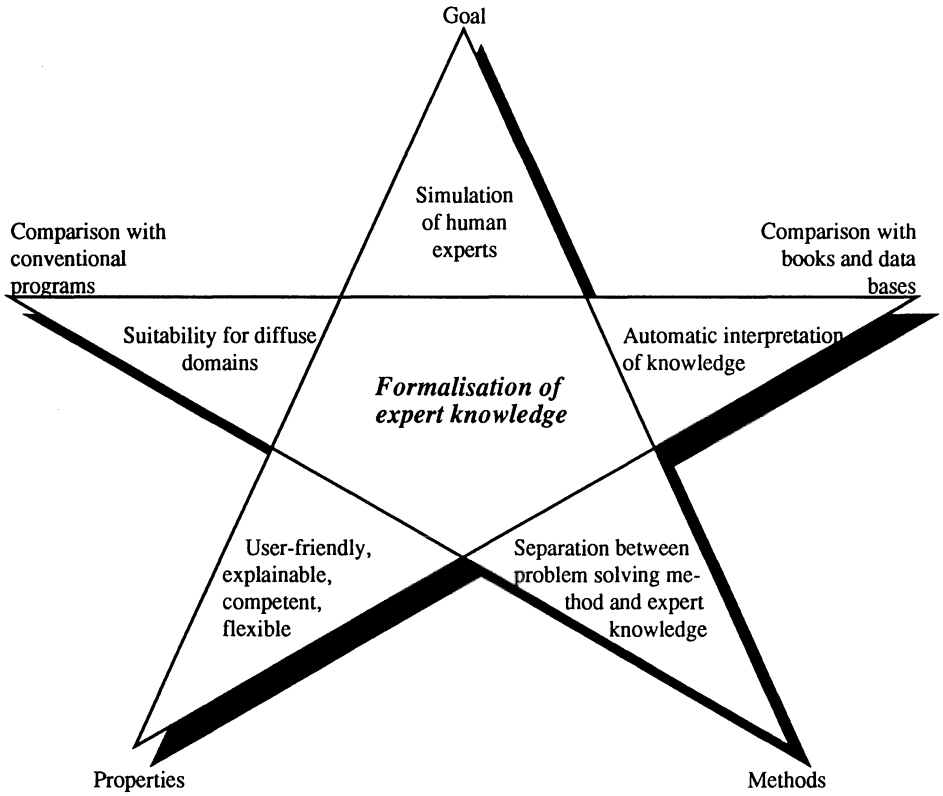



**Figure 1.3** Summary of the different ways of characterizing expert systems

For expert systems, however, these strategies play rather a subordinate role, just because of their generality. This finding was not generally accepted until during the 1970s, a fact which is expressed in the paradigm change described by Goldstein and Papert: "The fundamental problem of understanding intelligence is not the identification of a few powerful techniques, but rather the question of how to represent large amounts of knowledge in a fashion that permits their effective use and interaction. ... Thus we see AI as having shifted from a *power-based* strategy for achieving intelligence to a *knowledge-based* approach." (quoted after [Duda 83, p. 220]).

The first programs which showed the superiority of the coding of domain-specific knowledge over general problem-solving methods were MACSYMA, which simplifies algebraic expressions, and DENDRAL, which identifies molecules by means of mass spectrograms, both of which are in routine use today. However, the title "Grandfather of Expert Systems" is generally given to the expert system MYCIN developed at Stanford University for the diagnosis and therapy of bacterial blood infections and meningitis. Distinguishing features of this program are its methodical clarity, especially in the separation of the knowledge representation and the problem-solving method, and its explanatory component. On account of

these advantages it has become a model for succeeding systems. Further "famous" expert systems are:

- HEARSAY II for understanding spoken language, with a vocabulary of 1000 words and a simple grammar, which popularized the "blackboard architecture" (cf. Chap. 34) for the orderly coordination of different sources of knowledge,
- PROSPECTOR (see Sect. 13.1.5) for identifying geological formations in which test borings for mineral resources are worthwhile, and which correctly predicted a molybdenum deposit valued at 150 million dollars (however, test borings would have been made without PROSPECTOR),
- INTERNIST for diagnosing approximately 75% of all internal diseases, which is distinguished by the size of its knowledge base and is shortly to become available commercially as an electronic reference book QMR (**Q**uick **M**edical **R**eference, see Sect. 13.1.2).

The experience with expert systems of the first generation was soon reflected, in the late 1970s, in tools which considerably reduced the development time of new systems. The first tool was EMYCIN, which has retained the techniques and the knowledge representation of MYCIN, but may also be combined with other knowledge bases from medical and non-medical domains. An example is PUFF for the evaluation of lung function tests, which was implemented in EMYCIN and required only about five person-years development time compared to the twenty person-years for MYCIN.

The first expert system tools generalized the rule formalism above all. Recent research is concentrating on the development of more powerful tools, such as hybrid general tools, which combine several knowledge representations or program styles, including rules and object-based representations. Problem-specific tools are often seen as the most promising attempt to reduce the still tremendous effort required for the development of expert systems by restricting themselves to one problem-solving type, e.g., diagnostics, and using specialized problem-solving techniques (see Sect. 2.2).

Today more than one thousand expert system tools of all prices and qualities are commercially available, with which many experiments are being performed in industry and research. We expect that in the coming years the evolution of the tool systems by mutation (development of new tools) and selection (through experience in the use of tools) will make rapid progress.

# 2. Programming Languages and Expert System Tools

## 2.1 Knowledge Acquisition Problems

The central problem ("bottleneck") in the development of expert systems is finding a way of transferring a great deal of detailed knowledge to the program. The most important primary sources of knowledge are experts and textbooks. However, the knowledge in the minds of the experts is not easily accessible, because experts usually have little time or motivation to pass it on and mostly have great difficulties in doing so. In many fields of application there are no good textbooks, and even if they are available they usually require background knowledge and/or practical experience to be understood, so that textbook knowledge alone is seldom sufficient for the development of successful expert systems. The situation is made even worse by the fact that the knowledge often changes rapidly.

The prevailing approach to this problem at present is the invention of a new profession, that of the all-round "knowledge engineer", who acquires knowledge by reading textbooks and above all by communication with specialists, and formalizes it for the expert system. However, this *indirect knowledge acquisition* is not only expensive, but also prone to error on account of the unavoidable problems of communication between the expert and the knowledge engineer, who usually lacks the background knowledge of the expert. If one considers further the fact that many expert systems require continual maintenance on account of changing tasks and optimization of the knowledge, indirect knowledge acquisition with two groups of highly-paid specialists can become too expensive in the long run.

From the very beginning it was therefore both the wish of industrial users and the aim of many research workers to use learning methods for automating knowledge acquisition. Unfortunately, *automatic knowledge acquisition* turned out to be extremely difficult, which is not surprising in the face of the following considerations: Learning requires knowledge; the more one knows, the easier it is to learn, and human experts need about ten years of intensive occupation with their field before they become experts, whereby, unlike programs, they can build on general knowledge. Even learning programs therefore require complex knowledge acquisition. Perhaps even more serious is the fact that the already difficult validation problem is becoming even more acute. When a human being constructs or alters a knowledge base himself, he can estimate the performance

of the expert system much better than when a learning program makes changes of its own accord. To put it another way, who would take the responsibility for automatically generated knowledge in expert systems used in practice? At present the potential of learning programs is to be seen rather in the support of knowledge acquisition, e.g., in the fine adjustment of evidence values, where experts tend to have problems.

At present the best hope of making knowledge acquisition more economical would appear to lie in *direct knowledge acquisition*, i.e., the experts should be responsible for the construction and maintenance of expert systems in the same way as the author of a textbook. On the part of the experts this requires the ability to formalize and structure their knowledge.[1] In addition, they must be supported by comfortable environments for knowledge acquisition, with which they can easily input and test their knowledge. The environment then translates the knowledge into an internal knowledge representation which can be interpreted by a problem-solving method.[2]

## 2.2 Knowledge Acquisition with Specialized Programming Environments

In order that such a programming environment may be used easily by experts, it must be adjusted as far as possible to their ways of thinking and expressing themselves, which are reflected to a great degree in their specialized language. Compared to natural languages, specialist languages are more strictly formalized, contain special terms for concepts of the domain and often use graphics and pictures for representing information. In addition, particular domains often have typical problem-solving methods. Since the specialized language and the problem-solving methods are specific to the application, one cannot expect to be able to construct a universally applicable tool. On the contrary, the more specialized the tool is, the greater its use for the domain. Figure 2.1 shows various possible compromises between the requirements of wide applicability and time saving.

The main types of tools are:

1. Universal programming languages.

2. General expert system tools, which support basic knowledge representations frequently used in expert systems together with the appropriate inference strategies. Examples of the first generation are OPS5, KEE, ART and Knowledge Craft.

3. Problem-specific expert system tools (shells), which are specially designed for certain problem-solving types and provide suitable problem-solving

[1] That this is not a matter of course is shown by the fact that experts are often not good at putting over their knowledge.

[2] An up-to-date review of the various research approaches in the field of knowledge acquisition can be found for example in [Karbach 90].

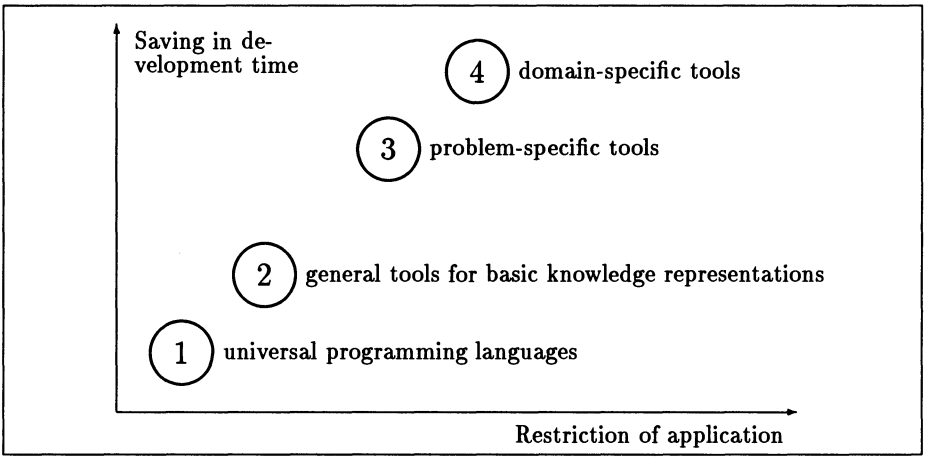



**Figure 2.1** Types of expert system tools. Of course there are great differences within each group, e.g., in universal programming languages between assembler languages and various high-level languages such as FORTRAN, PASCAL or LISP, or between simple and hybrid general tools, of which the latter support more than one basic technique

methods, knowledge representations and knowledge input forms. Examples are SALT (see Sect. 23.4) and MED2/CLASSIKA (see Sect. 15.5).

4. Domain-specific expert system tools, which, in addition to their function as problem-specific tools, also provide basic knowledge of the domain (e.g., technical terms). Example: OPAL (see Chap. 21).

The most important basic knowledge representations in expert systems (see Part II) are: rules with forward and backward chaining; frames (objects) with inheritance, defaults and attached procedures; constraints; evidences; justification networks for non-monotonic reasoning; and time specifications for temporal reasoning. The corresponding general expert system tools are mostly embedded in universal programming languages and may be regarded as an extension of the function of these languages in the form of one or more basic knowledge representations. The close interlocking is also shown by practical experience: 50% or more of the program codes of large expert systems constructed with a general tool are written in a universal programming language. In their further development, general expert system tools and universal programming languages seem to be drawing closer together, as shown also by the increasing emphasis placed on object-oriented programming in both fields.

Domain-specific tools are founded on the basic knowledge representations, but they are considerably more specialized. The three most important classes of domain-specific tools are classification (diagnostics), construction and simulation (see Part III). Figure 2.2 indicates that the problem-solving methods for the problem classes are founded on the basic knowledge representations but represent a new dimension (see Parts IV, V and VI).

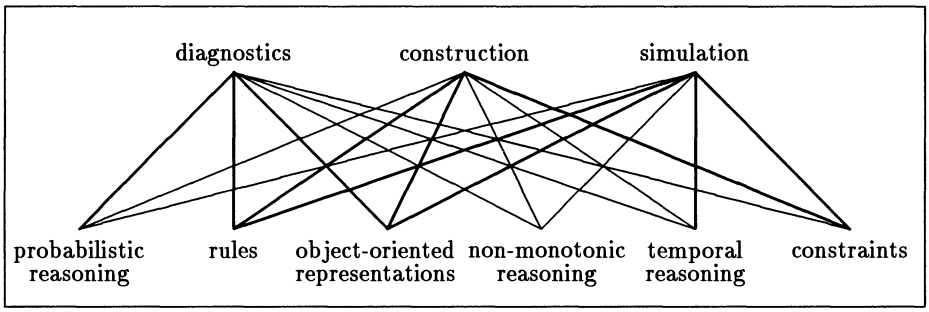



**Figure 2.2** Rough correspondence between the main problem classes and the basic knowledge representations

Problem-specific tools make it possible to construct an expert system with considerably less effort and knowledge of programming languages than with general tools. Their usefulness increases enormously when specialist experts are able to handle them directly and develop application programs by themselves, as is also possible with spreadsheet systems, for example. Due to lack of experience with problem-specific tools, it is not yet possible to answer the question as to whether fully developed problem-specific tools generally allow the development of expert systems directly by the specialist. If not, a subdivision of the class of problem-specific tools in Fig. 2.1 according to the criterion "usable by specialists" would be appropriate.

The classification of different expert system tools in the general context of program development is illustrated in Fig. 2.3.

The two most important dimensions of program development are the linguistic freedom and the degree of abstraction. The linguistic freedom is a measure of how vaguely or precisely the task and problem-solving procedure are specified. The degree of abstraction indicates whether the description is given more in terms of the concepts of the domain or in a programming language. The development of a program begins with the identification of a problem, which is first formulated in a vague and abstract way, and should finish with a precise problem solving procedure, consisting of an interplay between various components, such as human beings, sensors, control units, old programs and the new program. Here we consider only the newly developed computer program.

The recommended programming procedure is first to specify the problem in a problem-oriented language and then translate the precisely defined task into a machine language. Experience has shown, however, that people usually follow a mixed procedure, and "hackers" tend to go directly to the level of a machine-oriented programming languages. An important area of progress in computer science is the development of programming languages which are becoming closer to the application with each "generation", from machine language via assembler to high-level languages, whereby the necessary transformations are performed by a compiler. At present considerable effort is being put into the development of even more abstract programming languages ("specification languages"), a category in which the expert systems may be fully integrated. While most specifi-

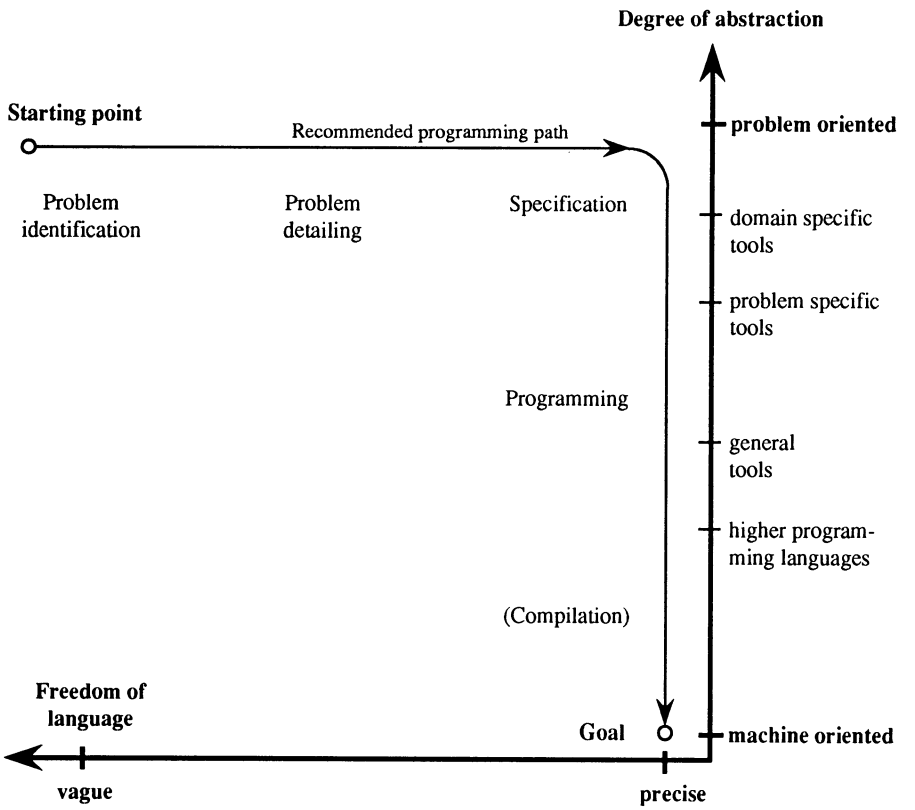



**Figure 2.3** Idealized scheme of program development

cation languages are based on mathematical or logical notations, expert system tools preferably use the specialized languages of the different domains. It is important that the specifications can be translated, automatically if possible, into the program code of a machine language. The closeness of the problem-oriented or domain-specific expert system tools to the application is achieved by specialization, resulting in a variety of different tools, each with a limited spectrum of applications, which is not shown in Fig. 2.2.

The recommended procedure for program development can be divided into different steps, according to the professions involved (Figs. 2.3 and 2.4)[3]. First of all a problem must be identified and its importance recognized, which is the job of the manager. The manager then delegates the problem to a specialist, who is responsible for the precise formulation of the problem and the abstract design for a problem-solving procedure. In the third step the task and the details of the problem-solving procedure are transformed by a system analyst or a knowledge engineer into a program specification, which is converted in the fourth step into a executable program by a programmer.

[3]These schematic diagrams illustrate the logical steps of the program development rather than the actual action, in which many iterations between all steps are unavoidable.

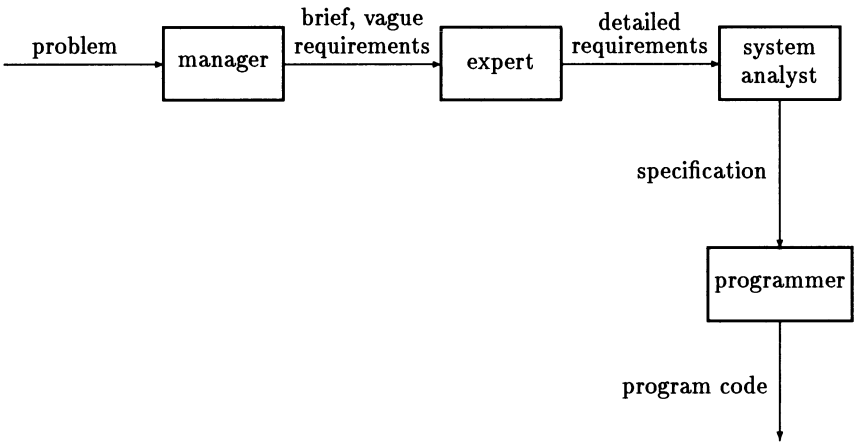



**Figure 2.4** Groups involved in the development of a program (after [Rich 87, Fig. 2]). For the sake of clarity the multiple iteration loops are not shown.

The aim of the development of expert system tools is to automate the last step in the transformation of the specification in a program written with a conventional programming language, and to enable the expert to carry out the detailing of the program up to the specification. It must be borne in mind that nothing other than an upward movement along the abstraction axis in Fig. 2.3 can be automated, since the precise representation of an initially vague problem can hardly be supported by formal aids. On the other hand, the potential gain is considerable, since having all programming activities in the hands of the specialist helps to avoid excessive administrative work and misunderstandings.

## 2.3 Architecture of Expert Systems

By the architecture of an expert system we mean its external form, i.e., the various program modules and their relationships to each other. The functional division between the expert knowledge and the problem-solving strategies is reflected, as far as the architecture is concerned, in the two main modules: the knowledge base and the control system.

The control system contains program code for the problem-solving strategies and for the interface to the user. The interface consists of three relatively independent parts for the user dialog, for generating explanations and for knowledge acquisition. We thus have the following sub-modules:

- The *problem-solving component* interprets the expert knowledge for the solution of the problems specified by the user.

- The *interviewer component* controls the dialog with the user and/or automatically reads in measured data. If no user dialog takes place, the expert system is called an *embedded system*, otherwise an *interactive system.*

- The *explanation component* makes the mode of operation of the expert system transparent. It helps both the user seeking the reasons or justification for the solution suggested and also the expert wishing to localize errors in the knowledge base.

- The *knowledge acquisition component* enables the expert to enter his knowledge to the expert system and alter it later. Depending upon the quality of the knowledge acquisition component, the expert may need the help of a knowledge engineer or he can be assisted by learning techniques, which evaluate case data bases, for example.

The knowledge base also comprises several parts. According to the origin of the knowledge, distinctions are made between *domain-specific knowledge* of experts, *case-specific knowledge* of users and *intermediate or final results* derived by the problem-solving components. Another way of subdividing the knowledge is according to its use. This gives the subdivisions *factual knowledge, derivation knowledge* and *control knowledge.* Derivation knowledge (e.g., rules) controls the use of the factual knowledge, and control knowledge (e.g., so-called meta-rules) controls the use of the derivation knowledge. The user's case-specific knowledge and the intermediate or final results consist typically of factual knowledge. The expert supplies not only factual knowledge (e.g., catalog knowledge) but also derivation and control knowledge. In the summary of the architecture of an expert system shown in Fig. 2.5 only the division of the knowledge base according to the origins is given.

Figure 2.5 also indicates the various groups which deal with an expert system:

- the developer of the control system,

- the experts, who need suitable control systems for constructing and testing knowledge bases, and

- the actual users, who wish to solve problems with the aid of the expert system.

The control system is often called the *shell.* Expert system shells continue the general trend in computer science to develop programming languages of even higher levels, i.e., closer to the application, so that the programmer needs to concern himself less and less with the concrete realization and can concentrate instead on the logical structure and the abstract control flow of the problem. The correspondence between the terms in expert system shells and programming environments is shown in Fig. 2.6.

With expert system shells it now seems to be becoming realistic for the people who have the knowledge to handle the computer on their own and become program developers, instead of having to depend on programmers as "interpreters".

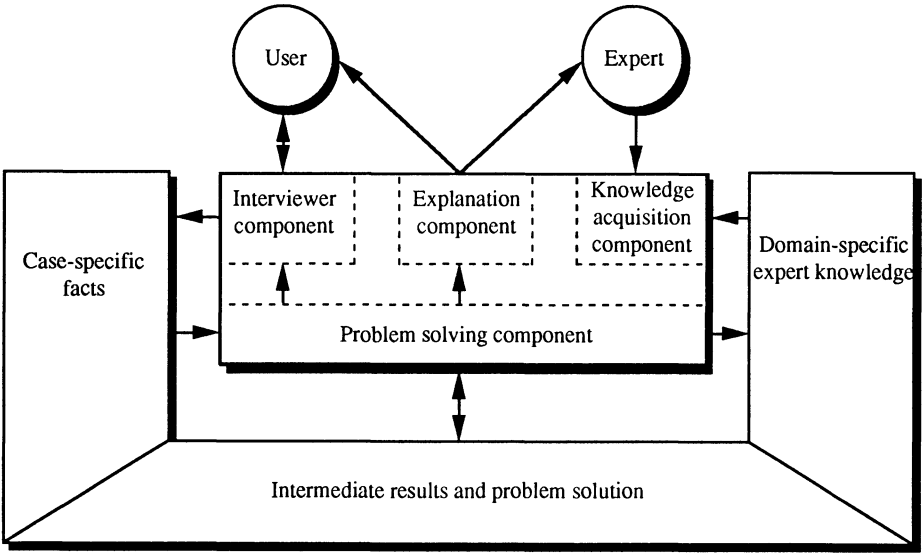



**Figure 2.5** General architecture of expert systems

| Expert system shell | Programming environment |
|---|---|
| knowledge representation | syntax |
| knowledge base | program |
| knowledge acquisition component | editor and compiler |
| problem-solving component | program interpreter or compiler |
| interviewer component | input/output functions |
| explanation component | tracer |

**Figure 2.6** Comparison of terms for expert systems and programming environments

This principle is already well established in the case of spreadsheet programs and word processors, for example. The ultimate aim of this development is programming at the "knowledge level" [Newell 82], where the problem-solving component is intelligent enough to use all available knowledge, independently of its representation at the "symbol level", in the best possible way for solving the problem.

# 3. Use and Usability of Expert Systems

It is still difficult to estimate the extent to which expert systems have commercially permeated the data processing world. Feigenbaum et al. report an industrial acceptance with over 1500 expert systems in use world-wide by the end of 1987, with a rising trend and profit-to-cost ratios of 10 to 100 [Feigenbaum 88], whereas Coy and Bonsiepen, in a critical analysis of expert systems [Coy 89], write that at least in the FRG industry is cautious with investments, and civilian state support is the motor of the market:

> "According to a report of the BMFT[1] [...] on the situation of AI research, a sum of at least 300 million DM was provided for the civilian financial support of research on artificial intelligence and expert systems technology in the period from 1984 to 1989/90. This sum went into cooperative projects of industry, universities and other state research institutes. The amount of military support through the BMVg[2], the IABG[3] and other industrial and state research and development departments is not given. Optimistic estimates give a maximum figure of 309 million DM for the market volume of this technology in the FRG from 1985 to 1988 [...] not including the firms' own contributions, which may be assumed to be of the same order of magnitude. Civilian support and the market are therefore in balance for the present, and the financial assistance is the motor of this market." [Coy 89, p. 164].

The extensive appraisal by Mertens et al. shows a so far continuous exponential increase, albeit starting at a low level, of expert systems in use in the German-speaking countries (Fig. 3.1) [Mertens 90]. The authors admit, however, that the distinction from conventional decision-support systems presents great problems, and they have opted for admission to their set of expert systems even in cases where there were doubts about the proportion of knowledge-based elements in the whole system.

A limited industrial use of expert systems may thus be taken as a fact, and the analysis of Mertens et al. shows high growth rates, but the great benefits expected seem still to lie in the future.

In this chapter we will first discuss the potential benefits. From this we will show that expert systems are not just problem-solving programs, but also a new

[1] *Bundesministerium für Forschung und Technologie* (Federal Ministry of Research and Technology).

[2] *Bundesministerium für Verteidigung* (Federal Ministry of Defense)

[3] *Industrieanlagen-Betriebsgesellschaft* (Industrial Plant Operating Society).

medium for the communication of knowledge. Finally, a review of an exhaustive list of criteria for the evaluation of expert system projects will be given. Throughout this discussion we will bear in mind our contention that insufficient understanding of suitable problem-solving methods greatly hinders the exploitation of the actual potential of expert systems.

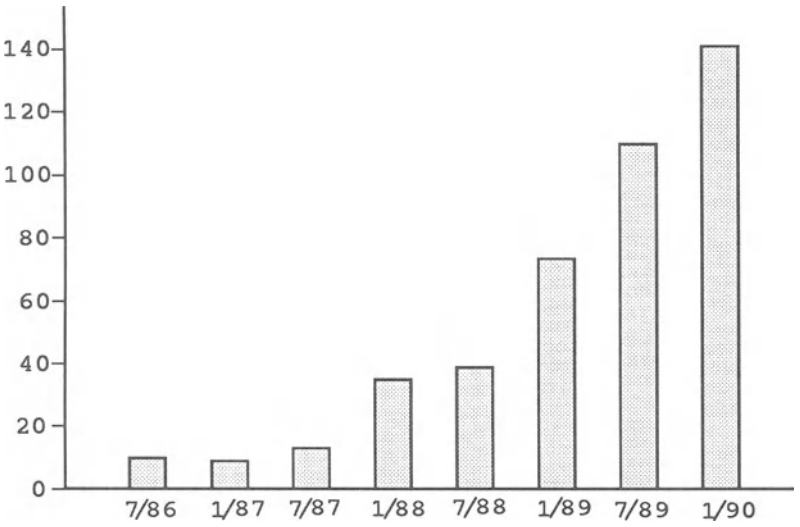



**Figure 3.1** Development of working expert systems in the German-speaking area after [Mertens 90]

## 3.1 Potential Benefits and Modes of Use

Almost all books on expert systems discuss in detail the various possible benefits, which may be summarized as follows: "The new technology provides the possibility for developing rapid and pragmatic solutions for a broad spectrum of problems which at present elude an effective solution." On the other hand, quantitative, detailed cost-profit analyses of expert systems are so far hardly to be found in the literature – with the exception of R1-XCON [Barker 89]. Generally, qualitative benefits are cited and/or quantitative values roughly estimated. Possible benefits include:

1. Effects of rationalization:
    - saving personnel costs through quicker and/or better solutions,
    - savings due to less waste in production,

- better use of capacity and higher production efficiency,
- savings by automatic documentation.

2. Effects of improved quality:
   - exploitation of rationalizing effects for quality improvement,
   - assurance of a minimum standard of the problem solution,
   - checking of solutions found by people or other computer programs,
   - higher qualification or quicker training of employees,
   - quicker and/or better drawing up of offers to obtain orders,
   - equipping expensive products with their "own intelligence", e.g., for some level of self-diagnosis,
   - further automatization of activities in an already partly automated environment, thus eliminating weak links (e.g., in fabrication).
3. Positive organizational effects in companies:
   - multiplication of knowledge,
   - preservation of knowledge,
   - decentralization of decision making,
   - favoring exchange and evolution of knowledge,
   - quicker distribution of company directives.

The risks include – apart from the possibility that the desired benefits fail to appear, leaving just the development and investment costs – in particular a dependence upon a poorly functioning expert system. This is true both for single decisions, when they can no longer be checked, and for the method of decision making, when the techniques used before introduction of the expert system are no longer available. One must also consider the possibility that, even if the expert system works well, the general organizational operating conditions may worsen. A specific risk of expert systems is insufficient secrecy of sensitive knowledge and problem data, since both must be formulated quite specifically in expert systems.[4]

For the cost-benefit analysis it is helpful to consider the exact function of the expert system. We distinguish three types:

1. *Embedded expert systems for decision making.* The expert system draws its input data via a computer link from other machines, e.g., measuring systems, data bases or other programs. In this way considerable rationalizing and quality improvement effects are possible, in particular where a human being is the weak link in an otherwise fully automated chain of operations.

[4]In this connection we maintain the quite controversial point of view that perfect data protection is not possible, and the danger of abuse is greater, the more attractive the data are for others.

A great potential for embedded expert systems is seen in the highly automated factory of the future, and also in equipping expensive, complex products and plants with problem-solving capabilities, so that they can monitor and interpret their state. Another example is a credit card information system from American Express, which when asked whether a certain credit is to be accepted extracts data from various data bases and makes a decision by applying a few rules.

2. *Interactive expert systems for decision making.* Interactive expert systems draw all or part of their data from a human user. Here one must distinguish between two types of application, according to whether the user or the system takes the responsibility for the problem solution. In the latter case the interactive expert system is similar to an embedded system in that it makes its own decisions. Such interactive decision making systems can have problematical side effects if they compete with experts or make the user play the role of a measuring apparatus. On the other hand they can also be a useful tool for the users, with which they can deal with problems they cannot solve without external help, thus increasing the scope of their actions.

3. *Interactive expert systems for advising.* In other application scenarios of interactive experts systems the user remains responsible for the solution of the problem, and the role of the expert system is to advise, criticize or make suggestions. In this case the principle function of the expert system is not to generate results, but to impart relevant knowledge. Such advisory or assisting systems are designed for expert users, who in principle can solve problems alone. As an additional benefit they contribute to improving the qualification of the user and therefore may be used also for basic and advanced training.

Since advisory systems interfere least with the internal organization of businesses and generate no dependence on poorly functioning expert systems, they cause comparatively low costs and few risks. They are therefore suitable candidates for introducing a new technology into a firm. By contrast it is often difficult to avoid the risk of serious errors with decision-making systems, since expert systems can only have a more or less local view of the problem. The situation is more critical, the bigger and complexer the expert system, which is why mainly relatively small decision-making expert systems are in use at present.

Expert systems whose main benefit is the efficient transfer of knowledge should not be regarded primarily as problem solvers, but rather as new media for knowledge. This aspect has so far been practically ignored in the cost-benefit discussion of expert systems. The consequences are discussed in greater detail below.

## 3.2 Expert Systems as Knowledge Media

The view of expert systems as knowledge media emphasizes the fact that knowledge is to be imparted. Figure 3.2 gives a summary of the most important knowledge media by means of which knowledge bearers can pass on knowledge to any number of knowledge receivers, since no direct contact is necessary.

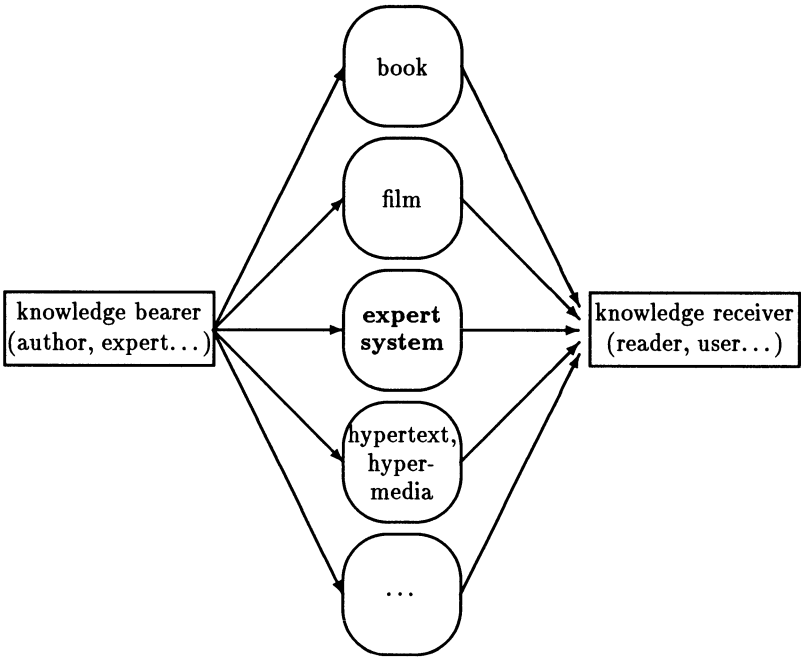



**Figure 3.2** Survey of some knowledge media which allow a knowledge bearer to pass on knowledge to any number of knowledge receivers

Expert systems have the following advantages over other knowledge media:

- *Controllability by the user.* Books, and to a greater extent films, are sequential knowledge media, i.e., knowledge can be received practically only in the order, in the case of films only at the speed, in which it is sent. Further, it is hardly possible to take into account the differences in previous knowledge and interests of the different users. In expert systems, on the other hand, the knowledge can be prepared according to different criteria, read in an associative way and above all applied to concrete problem cases. In principle, hypertext and hypermedia systems also have this advantage.

- *Verifiability.* The decisive advantage of the expert systems as knowledge media is that the knowledge can be verified not only by other people but also by the knowledge medium itself. With expert systems it is possible to

carry out experiments to find out which and how much knowledge is needed for the solution of concrete problems.

The more easily and effectively knowledge can be verified in given domain, the faster is the evolution of knowledge. A good verifiability of knowledge has been guaranteed so far above all in the so-called exact sciences, such as physics, by the application of mathematical methods. By contrast the main advantage of expert systems is that they are also able to express and interpret vague, incomplete and non-numerical knowledge, and also make clear what effects the vagueness has on the quality of the resulting problem solutions. It is to be expected that in the long run the greatest benefits of expert systems will result from the promotion of knowledge evolution in domains which have so far been poorly formulated.

The advantages may be illustrated by the example of a hypothetical, fully developed medical expert system for an established medical field. Such an expert system would make it possible to compare different orthodox opinions, which may be expressed by different expert systems or by different knowledge sections within one expert system, and thus make their advantages and disadvantages verifiable. Further, the theoretical training could be considerably improved, since the students could first read the expert system knowledge conventionally or in an associative, hypertext way, have solved case examples explained by the explanation components, study the effects of small variations in the symptoms and finally have their acquired problem-solving ability tested and criticized by the expert system by offering their own case solutions.

In order to make full use of the expert system as a knowledge medium, the following technical conditions should be fulfilled:

- The expert systems must be based on problem-solving methods which are easily followed by human beings.
- The effort required for the knowledge bearer to formulate the knowledge must be as small as possible.
- The form in which the knowledge is presented to the receiver must be as good as possible.

Since knowledge according to our definition can only be formulated with regard to the interpretation by a problem-solving method, the usefulness of the knowledge medium depends primarily on how well the communication partners in Fig. 3.2 understand the problem-solving method concerned. Problem-solving methods which people find difficult to follow must therefore be excluded.[5] These include generally all processes requiring complex calculations, those having no symbolic knowledge representation (in particular connectionist models, if they represent a "black box" for the user) or those involving long chains of reasoning which are difficult to follow. Those problem-solving methods which are usual in the particular domain are very suitable.

[5]Unless they are so important that a special training is worthwhile, e.g., mathematical thinking. The group of possible users of the expert system is limited accordingly.

The second requirement concerns the ease with which the expert can formulate his knowledge for the expert system. In principle the knowledge formulation should require no more effort than writing a corresponding book.[6] The knowledge representation required by the problem-solving method should be presented to the expert on a level which is so close to the application that he requires no special programming knowledge for the input. For this purpose graphical means of representation are especially suitable. If there are different problem-solving methods in one domain, the expert should not be compelled to enter similar pieces of information repeatedly.

The third requirement is the quality of the knowledge presentation to the user. For this purpose basically the same graphical means of representation as those used by the expert for his input are suitable. In addition, routines can be provided which prepare the knowledge in a structured form according to further criteria. The knowledge can be offered in two forms: as a printout on paper, similar in form to a book, or on the screen, where the user can read the text associatively as in hypertext systems. The screen mode also supports other user modes, above all the case-specific selection of knowledge parts, where the user enters the features of his problem and the expert system either generates and explains a solution or criticizes the solution suggested by the user. Further, the user should be able to compare the effects of different alternative solutions.

## 3.3 Criteria for Expert System Domains

Lists of criteria for judging suitable domains are very important for every technology. Figure 3.3 shows an example of an easily evaluated check list for potential expert system applications, which has been tested by its authors.

A striking feature of this and other lists of criteria is, however, that criteria concerning the methodical suitability are very vague, e.g., in Fig. 3.3 criteria such as "predominantly heuristic knowledge", "not general knowledge", or "similarity to existing, successful expert systems". It would appear that there are no exact data concerning the question as to which methodical classes of domains are especially suitable for expert system techniques, as the following quotation from an expert system journal for managers complains:

> "Ever since AI and expert systems broke upon the commercial scene, people have been asking what AI is good for. It's a reasonable question, especially when you consider the vague and contradictory way early advocates described and positioned the various expert system-building tools and techniques.
>
> Early advocates of expert systems usually just described specific techniques and assumed that commercial practitioners would immediately recognize the appropriate use for these techniques. From the beginning, we have tried to turn the process around. Our goal has been to help people

[6] This comparison can only serve as a rough guide, since books and expert systems have both overlapping and disjunctive parts.

> identify problems and then suggest which AI techniques are most effective in solving those problems. It has been a struggle since, in many cases, academic advocates have not really known which techniques are most appropriate for specific problems. Thus, to draw our conclusions about which techniques are most useful, we have had to observe the market to see how commercial enterprises have, in fact, used expert systems. It has taken a while, but the results are finally beginning to emerge."[7] [ESS 88, p. 1]

The fact that the technical feasibility of expert system projects is difficult to estimate has surely contributed much to the mystification of the expert system technique. A de-mystification requires above all a division of domains into problem classes with a statement about which problem classes are well solvable with which problem-solving methods and where difficulties are to be expected. If this is generalized, the following framework for the classification of expert systems follows:

1. Division of concrete problem domains into problem classes.
2. Relation of the problem classes to the problem-solving methods.
3. Mapping the problem-solving methods in the basic knowledge representation.
4. Implementing the basic knowledge representations.

## 3.4 Summary

The applicability of expert systems is hindered at present by the lack of a theory of which problem-solving methods are available for which problem classes, whereby the practicability of such a theory must be judged by the usefulness of expert system tools based on it. Significant progress would be expected in the following central aims of the expert system technique:

- Evaluating the technical feasibility of expert system projects.
- Simplifying knowledge acquisition and ensuring maintenance.
- Making use of expert systems as knowledge media.

An approach towards such a theory is presented below. The basic techniques of knowledge representation are described in Part II, and Part III presents the division of problem domains into problem classes and a review of the problem-solving methods. The most important problem-solving methods for classification, construction and simulation are described in Parts IV, V and VI. In Part VII some remarks on the integration of problem-solving methods follow.

[7] The classification suggested here will be considered in more detail in Sect. 3.5.

| | |
|---|---|
| **I.** | **Essential criteria** |
| 10 | The users of the system expect great benefits in routine operation. |
| 10 | The users have realistic expectations concerning the size and limits of the system. |
| 10 | The project has much management commitment. |
| 10 | The task does not require natural language processing. |
| 7 | The task is knowledge-intensive, but not too large. |
| 8 | The task is heuristic in nature. |
| 10 | Test cases of all degrees of difficulty are available. |
| 7 | The system can grow in increments (the task is divisible). |
| 10 | The task requires no or little common sense. |
| 8 | No optimum problem solutions are required. |
| 10 | The task will still be relevant in the foreseeable future. |
| 7 | It is not essential that the system be ready at a short deadline. |
| 8 | The task is easy, but not too easy for expert systems. |
| 10 | An expert exists. |
| 10 | The expert is a genuine expert. |
| 10 | The expert is committed to the project for the whole duration. |
| 8 | The expert is cooperative. |
| 8 | The expert is articulate. |
| 8 | The expert is reliable and has general project experience. |
| 8 | The expert uses symbolic reasoning. |
| 7 | It is difficult, but not too difficult, to transfer the expert knowledge (e.g., teach). |
| 10 | The expert solves the problems with cognitive, not motory or sensory skills. |
| 10 | Different experts agree on good solutions. |
| 10 | The expert need not be creative in solving the problem. |
| **II.** | **Desirable criteria** |
| 8 | The management will support the project after completion. |
| 4 | The introduction of the system will not require much reorganization. |
| 4 | The user can interact with the system. |
| 4 | The system can explain its reasoning to the user. |
| 4 | The system asks not too many and no unnecessary questions. |
| 4 | The task was known earlier to be problematical. |
| 4 | The problem solutions in the task are explainable. |
| 5 | The task does not require response times which are too short. |
| 8 | There are successful expert systems which resemble the planned system. |
| 5 | The planned system can be used in several locations. |
| 3 | The task is dangerous or at least unattractive for humans. |
| 4 | The task also contains subjective knowledge. |
| 3 | The expert will not be available in the future (e.g., because of retirement). |
| 4 | The expert was able to identify himself intellectually with the project. |
| 4 | The expert does not feel threatened. |
| 2 | The problem-solving knowledge which the expert uses is structured, at least loosely. |

**Figure 3.3** List of criteria for evaluating expert system projects. They are based on [Slage 88], where they are explained. The 40 criteria are weighted with numbers from 0 to 10, divided into essential and desirable criteria, and classified in each group according to (1) user and management, (2) task and (3) experts. For a potential application each criterion is evaluated with a number from 0 (does not apply) to 10 (applies fully). The final evaluation is given by the weighted sum of all individual scores divided by the sum of all weight factors and is also a number between 0 and 10. If the individual value of an important criterion (weight factor $> 5$) is poor ($< 4$), this leads to a global devaluation. In [Slage 88] two candidates for expert system projects are evaluated as examples: a project estimated to be positive with a weighted average of 9.3, and a projected estimated to be negative with a value of 7.0.

# Part II

# Basic Techniques of Knowledge Representation

# 4. Logic

The knowledge representation which has been most thoroughly investigated experimentally and is often used as a reference point for other representations is first-order predicate logic. However, since its practical use for expert systems is relatively slight, it is used here mainly as an introduction to the problems of knowledge representation systems and as a motivation for the development of other formalisms. Comprehensive introductions to predicate logic can be found, e.g., in [Bibel 87], [Charniak 85], [Genesereth 87], [Lloyd 87], [Melvin 91], [Nilsson 82].

In order to achieve the goal of expert systems, the reproduction of the special knowledge and reasoning capabilities of qualified experts, the intuitive concept of reasoning must be formalized. In this process, subjective associations connected with individual concepts should play no role, but all relevant knowledge must be explicitly formulated. Derivations may be performed only according to explicit rules.

## 4.1 Predicate Logic

From the same aims there arose the idea of a calculus in which objects or states of the real world could be described by propositions of a calculus, from which, with the aid of general rules of inference, other propositions could be derived, which in turn also applied to objects or states of the world. The starting propositions of the calculus are called axioms, facts or premises, and the derived propositions theorems or inferences. A particularly simple calculus is the *propositional calculus.* The basic concepts consist only of simple propositions (statements) such as "it is raining" or "the abdominal pains are severe" (usually represented by capital letters $A, B$). These propositions are related with "and", "or", "negation" and "implication" $(\wedge, \vee, \neg, \rightarrow)$. The most important rule of inference is the *modus ponens*, which states that $A$ and $A \rightarrow B$ together imply $B$.[1] With the aid of the propositional calculus only concrete statements may be made about the world; it

[1]The propositional calculus also includes certain generally valid statements (logical axioms), e.g., $A \rightarrow (B \rightarrow A)$, $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$ and $\neg\neg A \rightarrow A$.

is not sufficient for describing general laws, such as in the following, for example:
Premise 1: All human beings are mortal.
Premise 2: Socrates is a human being.
Inference: Socrates is mortal.

For this purpose the propositional calculus must be expanded to first-order *predicate logic* by using predicates (e.g., human-being) instead of simple statements and allowing individuals (e.g., Socrates) as arguments of the predicates.[2] Individuals can be represented as constants or variables and also as function expressions, which can be regarded as a complex way of referring to objects: (+ 3 4) and (teacher-of Plato), for example, refer to the objects 7 and Socrates, respectively. Propositions about the existence of, or referring to all individuals are introduced by the quantifiers "there exists" ($\exists$) and "for all" ($\forall$), e.g., $\forall x((\text{human-being } x) \rightarrow (\text{mortal } x))$. Further, in the transition from propositional calculus to predicate logic, two axioms and a rule are added to the modus ponens and the axioms of the propositional calculus. The first axiom is the specification axiom $(\forall x p(x) \rightarrow p(t))$, which enables all-quantified variables to be replaced by definite values.[3] The second axiom concerns the relation between $\forall$ and $\rightarrow$: $(\forall x(A \rightarrow B)) \leftrightarrow (A \rightarrow \forall x B)$, where $x$ may not appear relevantly in $A$ ("quantifier shift axiom"). Finally one needs the generalization rule: $A$ implies $(\forall x A)$. Fig. 4.1 gives a complete summary of the syntax of a calculus for first-order predicate logic.[4]

In first-order predicate logic the quantifiers "for all" of "there exists" can be applied only to individuals, e.g., $(\forall x(px))$, whereas second-order predicate logic allows them to be applied also to sets or properties of individuals, e.g., $(\forall p(px))$. However, second-order predicate logic is difficult to implement and is therefore rarely used in applied computer science.

## 4.2 Properties of Logic Calculi

All formalisms for knowledge representation together with their inference strategies can be regarded as calculi. The usefulness of a calculus depends on its internal properties and on how well the real world, i.e., the domain, can be mapped in the calculus. An indispensable requirement is the *soundness* of the calculus, which is given when all conclusions which may be derived syntactically (i.e., within the calculus) also follow semantically (i.e., in the real world). The *expressive power* determines which statements about the world are representable. A calculus is *complete* if all conclusions which hold semantically are also derivable syntactically, and *decidable* if it is always possible to decide whether a given proposition follows from the axioms or not. Two very important properties in connection with

[2]Here we use the functional notation, in which the expression is enclosed in brackets, and the first argument represents the predicate or the function which is applied to the following argument. For Premise 2 we thus write (human-being Socrates).

[3]Of course no conflict of variables may arise in this substitution.

[4]Apart from the one chosen here, there are other ways of stating the axioms and rules of predicate logic.

| | | |
|---|---|---|
| Symbols: | • constants | $A, B, C$ |
| | • variables | $x, y, z$ |
| | • predicate symbols | $p, q$ |
| | • function symbols | $f, g$ |
| | • relation symbols | $\wedge, \vee, \neg, \rightarrow$ |
| | • quantifiers | $\forall, \exists$ |
| Terms: | • constants | $A, B$ |
| | • variables | $x, y$ |
| | • function symbols, applied to correct number of terms e.g., $(fx(fBB))$ | |
| Formulas: | • predicate symbols, applied to correct number of terms (atomic formulas) e.g., $(pxy(fxB))$ | |
| | • If $r$ and $s$ are formulas, then $(r \wedge s), (r \vee s), (\neg r)$ and $(r \rightarrow s)$ are formulas. | |
| | • If $x$ is a variable and $p$ a formula, then $(\exists x p)$ and $(\forall x p)$ are formulas. | |
| Axioms: | • proposition calculus axioms, specification axiom, "quantifier shift axiom" | |
| Rules of inference: | • modus ponens, generalization rule | |

**Figure 4.1** Syntax of a calculus for first-order predicate logic

expert systems are *adequacy* and *efficiency*, which express how "naturally" (i.e., simply and elegantly) problem domains may be described and how efficiently solutions to the problems may be derived. Efficiency is often attainable only for the derivation of those inferences which are particularly relevant in the context of the application. A useful property is *local confluence*, which is given when the derivation of a theorem cannot be prevented by the application of any derivation step. Finally, *consistency* is also interesting, which is not a property of the calculus, however, but means that a set of propositions may not contradict themselves. With an inconsistent set of premises (e.g., $A$ and $\neg A$) any theorem whatsoever can be proved in predicate logic.

Ideally, the knowledge representation should have all the above properties to the maximum degree. This is of course not possible; the relatively powerful second-order predicate logic, for example, is neither complete nor decidable; the less powerful first-order predicate logic is also undecidable, but at least complete; and the simple propositional calculus is both complete and decidable, even though no efficient decision procedure exists in practice. A good knowledge representation is distinguished by the best compromise for the domain concerned.

For the automatic production of the proofs of mathematical theorems, first-order predicate logic has proved successful, especially since the introduction of the *resolution calculus* by Robinson [65]. Resolution calculus works with clauses and the resolution principle as the only rule of inference. A *clause* is a disjunction of literals, which represent propositions in propositional calculus and atomic formulas in predicate logic and can be negated, e.g., $(A \vee B \vee \neg C)$. The clauses

restrict the expressive power of first-order predicate logic only slightly, since its formulas may be converted into a conjunction of clauses, as described, e.g., in [Nilsson 82, pp. 145–149]. Two clauses can be *resolved* when one contains a literal which is negated in the other: a new clause is formed by combining the two old clauses and eliminating the complementary pair of literals. The two clauses $(A \vee C)$ and $(\neg C \vee D)$, for example, resolve to $(A \vee D)$. With formulas containing variables, very much effort may be required to determine whether the variables may take values leading to the occurrence of such a complimentary pair. This is the subject of unification theory [Siekmann 87]. The validity of a theorem in first-order predicate calculus can be tested by first negating the theorem to be proved, then transforming all formulas into the clausal form and resolving until the empty clause is derived (e.g., $(P)$ and $(\neg P)$ give the empty clause). This shows that the negation of the theorem leads to a contradiction, so that the theorem must be valid. A general introduction to the theory of the proof of theorems is given in [Melvin 91].

For knowledge representation in expert systems, however, predicate logic has certain serious disadvantages with regard to expressive power, adequacy and efficiency. In many domains the knowledge and data are unsure, incomplete or time-dependent, which is difficult or impossible to express in predicate logic. For large knowledge bases one needs adequate structuring means such as hierarchies or contexts, which do not exist in first-order predicate calculus. The efficiency can often be considerably improved by adding application-dependent control knowledge, but the clausal form of resolution calculus is poorly suited for this purpose. Even the inability of first-order predicate calculus to deal with inconsistent statements can be a disadvantage, as pointed out by Minsky [75, Appendix]. Instead of forbidding inconsistent propositions, one only needs to make sure that they are not used in the same context.

## 4.3 PROLOG

A calculus which is less powerful but more efficiently interpretable than resolution calculus is the *Horn clausal calculus*, on which the programming language PROLOG is based [Clocksin 81]. A Horn clause is one with exactly one non-negated literal, e.g., $(\neg A \vee \neg B \vee \neg C \vee X)$. This can be transformed to a logically equivalent form and written as a "rule" with an implication arrow: $(A \wedge B \wedge C \rightarrow X)$. In the PROLOG notation the implication arrow is reversed, and $\wedge$ is replaced by a comma: $(X \leftarrow A, B, C)$. The most important limitation of Horn clauses compared to resolution calculus is that disjunctions of the form $(X \vee Y \leftarrow A)$, which is equivalent to $(\neg A \vee X \vee Y)$, cannot be represented. The application of the resolution principle in PROLOG is simple, since to derive a literal it is only necessary to find a clause with the literal on the left side and then derive all the literals on the right side (i.e., in the above example derive $A$, $B$, and $C$ to infer $X$). The processing procedure is equivalent to backward chaining of rules and is described in detail in Sect. 5.2. The main work of the PROLOG interpreter

is unification, i.e., finding substitutions which make two literals with variables equal. If there are several Horn clauses whose left sides can be unified with the present goal, PROLOG tries all substitutions one after the other and so is able to find all solutions of a problem and overcome all dead ends by backtracking (unless it runs into an endless loop, see below). A dead end results, for example, when a substitution is incompatible with subsequent, necessary ones.

The PROLOG interpreter attempts to unify the goal literal with the left sides of Horn clauses in the order in which they occur in the PROLOG program. Thus PROLOG's control knowledge is coded in the program sequence. In a case distinction it is therefore necessary to list the simple cases before the complicated ones. This is illustrated by the following example in PROLOG notation, which represents the rule "an ancestor is either a parent or the ancestor of a parent":

$$\begin{aligned} \text{ancestor}(A, N) &\leftarrow \text{parent}(A, N) \\ \text{ancestor}(A, N) &\leftarrow \text{parent}(A, X),\ \text{ancestor}(X, N). \end{aligned}$$

If one reversed the order of the two clauses for "ancestor" or the two literals on the right side of the second clause PROLOG could finish up in an endless loop, depending upon the fact base.

The significance of the order of clauses and literals, control instructions such as "cut" and other non-logical elements of PROLOG cast doubt upon its claim to be "**PRO**gramming in **LOG**ic". The criticism of first-order predicate logic at the end of Sect. 4.2 concerning expressive power, adequacy and efficiency applies almost as well also to PROLOG. Although PROLOG, like other universal programming languages, has too low a level of abstraction for large expert systems, it has become very popular as a control language for fairly small expert systems, since it provides a rule interpreter with backward chaining and unification (see Chap. 5 and, e.g., [Walker 87]).

## 4.4 Summary

The formalization of the intuitive concept of reasoning is the object of logic. For this purpose many calculi have been developed for deriving or proving theorems from axioms with the aid of rules of inference. Some important properties of calculi are soundness, expressive power, completeness, decidability, adequacy for the domain to be modeled, efficiency, local confluence and consistency. One of the most thoroughly studied calculi is first-order predicate logic, which serves as a reference for other calculi; it is complete and locally confluent, but not decidable. Detailed treatments can be found, e.g., in [Lloyd 87, Melvin 91] and in many introductory books on artificial intelligence. The weaknesses of first-order predicate logic for many domains are its deficiencies in expressive power, adequacy and efficiency, which can be improved by explicit representation of control knowledge, of knowledge structuring and of uncertain, incomplete and time-dependent knowledge. The optimum mixture of these properties, which may

involve deliberate restriction of expressive power to improve efficiency, depends upon the requirements of the particular domain.

# 5. Rules

A rule consists of a *precondition* (premise) and an *action.* The precondition describes a situation in which the action is to be performed. There are two types of action:

- *Implications*, with which the truth of a statement is derived (Rule (1) in Fig. 5.1) and
- *Instructions* for changing a state (Rule (2) in Fig. 5.1).

```
(1)  If     1.  stiff neck and
            2.  high temperature and
            3.  impairment of consciousness occur together,
     then       meningitis is suspected.

(2)  stack(box1, box2)
     if     1.  clear(box1)
            2.  holding(box2)
     then   1.  on(box2, box1)
            2.  clear(box2)
            3.  holding()
```

**Figure 5.1** Examples of rules with implications (1) and instructions (2)

Since experts often formulate their knowledge in the form of rules, rules are the most widely-used knowledge representation in expert systems. The division of the knowledge into as many "fragments" as possible makes a knowledge base modular and therefore easy to modify. The adaptability of the basic formalism to the domain-specific compromises between expressive power and efficiency is particularly attractive. For example it is relatively easy to extend rules for the representation of uncertain or incomplete knowledge by including statements about uncertainty or exceptions, or to adjust their expressive power to special requirements of the problem domain. There is also an important difference between rule

systems with implications and those with instructions, since the latter are generally not locally confluent. Despite all the variations of the basic formalism, a few general statements may be made about rule-based systems.

A rule-based system consists of a data base containing the valid facts, the rules for inferring new facts and the rule interpreter for controlling the inference process. There are two basic alternatives for processing the rules:

- *Forward chaining.* Starting from a given data base, the actions of rules whose preconditions are fulfilled by the data base are performed (i.e., these rules are "triggered"), which produces changes in the data base. This procedure is repeated until no more rules are applicable.
- *Backward chaining.* Starting with a given goal, only those rules whose action part includes the goal are checked. If some parameters of the preconditions are unknown, they are requested from the user or derived with other rules.

In the following sections we describe the two processing strategies of a rule interpreter, proposition types of different expressive powers in the rule preconditions, and mechanisms for structuring sets of rules and single rules. Possibilities for structuring the data base are treated in Chap. 6. For the present it is sufficient to regard the data base as an unordered set of facts.

## 5.1 Forward Chaining

In forward chaining the rule interpreter derives all implications which may be inferred from the data base (see Fig. 5.2). The selection process in step (4) of Fig. 5.2 is usually carried out in two phases:

- Pre-selection: determining the set of all executable rules, also called the conflict set.
- Selection: selection of a rule from the conflict set by means of a conflict-resolving strategy.

Components of the rule interpreter: 1. data base
2. rules

(1) DATA ← initial data base,
(2) *until* DATA satisfies termination condition *do*
(3) *begin*
(4) choose executable rule $R$ whose precondition is satisfied by DATA,
(5) DATA ← result of applying action part of $R$ to DATA,
(6) *end.*

**Figure 5.2** Forward-chained rule interpreter (after [Nilsson 82, p. 21])

In locally confluent systems the conflict-resolving strategy is used only to avoid superfluous rules, but it is more important in non-confluent systems, where the application of a wrong rule may prevent a solution being found. The most important conflict-resolving strategies, which of course can be combined with one another, are:

- Selection by order, e.g.:
  - Apply the first matching rule (trivial strategy).
  - Apply the most current rule, i.e., the one whose precondition applies to the most recent entries in the data base.
- Selection according to the syntactic structure of the rule, e.g.:
  - Apply the most specific rule: A rule $R1$ is more specific than $R2$ if the precondition of $R1$, e.g., $(A\&B\&C)$, is subsumed by the precondition of $R2$, e.g., $(A\&C)$.
  - Apply the syntactically largest rule, i.e., the rule which contains the most propositions.
- Selection by means of supplementary knowledge, e.g.:
  - Apply the rule with the highest priority. For this purpose each rule must be given a priority, which may be represented, e.g., by a number.
  - Additional rules, the so-called meta-rules, control the selection.

A simple way of implementing a forward-chained interpreter is to check the preconditions of all rules, sort the applicable rules in an agenda (i.e., an ordered list) according to the criteria given by the conflict-resolving strategy, and then perform the action of the leading rule in the agenda. Then the agenda is deleted and the process begins anew.

A first improvement of this "brute force" algorithm is to make use of the similarity between the preconditions of different rules. In this way rules with a common proposition may be eliminated together if the proposition does not hold. For this purpose the rules are structured in a tangled hierarchy by constructing a network of interlocking trees in which the internal decision nodes are formed by the propositions and the "leaves" by the rules. The path from the root of the tree to a rule includes all the propositions of the precondition of this rule.

A second improvement follows from the consideration that a complete new calculation of the agenda requires far more effort than a modification on the basis of changes. From one cycle to the next, the data base is altered only by the action of the selected rule, which may add or delete portions of data. Thus one only has to test whether the deleted data destroy the applicability of any rule in the old agenda and whether the added data make any new rules applicable. The latter test is performed by running through the rule network from the new data as starting point.

Both ideas for improvement have been realized in the RETE algorithm, which is implemented in the forward-chained rule interpreter OPS5 [Forgy 81, Brownston 85]. OPS5 uses a conflict resolving strategy which selects the most current and most specific rules.

Another way of improving the "brute force" algorithm is to index all rules, so that each parameter produces a reference to its rule. When a parameter takes on a new value or changes its value, the references enable the rules concerned to be found and checked efficiently. An example of a complete rule indexing is the rule interpreter of MED2 [Puppe 87, Sect. 4.4].

## 5.2 Backward Chaining

While forward chaining allows conclusions to be drawn only from a given data base, a backward-chained rule interpreter is suitable for requesting still unknown facts. Figure 5.3 shows a schematic representation of the recursive structure of such an interpreter.

A backward-chained rule interpreter starts with a pre-determined goal (in Fig. 5.3: *FINDOUT*(GOAL)). If the aim is not known in the data base, the rule interpreter first decides whether it can be derived or must be requested. For this purpose there are rules analogous to the conflict-resolving strategies of the forward-chained rule interpreter, e.g., first ask and then derive, first derive and then ask or mixed forms with further decision criteria. In the case of a derivation, all rules which include the goal in their action part are processed. These can be found efficiently if they are indexed according to their action parts. If a parameter is unknown when a rule is checked (in Fig. 5.3: *MONITOR*(RULE)), an intermediate goal is generated to determine this parameter, and the procedure *FINDOUT*(GOAL) is called recursively to derive this intermediate goal. The end result is the determination of values for the final goal and for all intermediate goals, the evaluation of the relevant rules and the formulation of the necessary questions. Backward chaining therefore contains implicitly a dialog control, whereby the order of the questions put depends upon the order of the rules for deriving a parameter and upon the order of the propositions in the precondition of a rule. This dependence on the order reduces the modularity of the rules system considerably, however. The efficiency of a backward-chained rule interpreter is determined by the formulation of the goal: the more precise the goal, the smaller is the search tree of rules to be checked and questions to be asked (e.g., in medicine: "What is the name of the disease?" as against "Is X the name of the disease?").

Well-known examples of backward-chained rule interpreters are EMYCIN [van Melle 81] and PROLOG (see Sect. 4.3).

Procedure *FINDOUT*(GOAL)

(1) *If* (GOAL can be inferred),
(2) *then* (Set RULE-LIST = list of all rules whose action part fulfills GOAL),
(3) *until* (RULE-LIST = empty) *or* (GOAL inferred) *do*
*MONITOR*(first or next rule from RULE-LIST) and
delete this rule from RULE-LIST.
(4)*else* (request GOAL).

Procedure *MONITOR*(RULE)

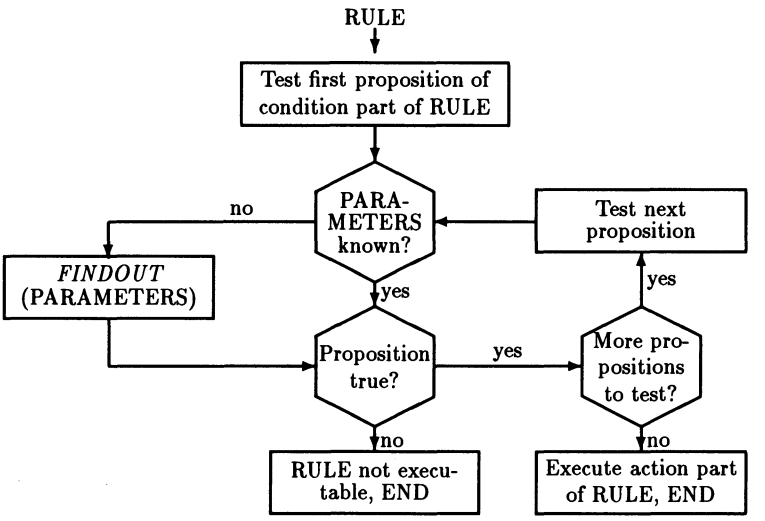



**Figure 5.3** Backward-chained rule interpreter (after [Buchanan 84, p. 106])

## 5.3 Complexity of the Precondition

Different rule systems have formalisms of differing expressive powers for describing the preconditions and the rules. The differences are similar to those between propositional and predicate logic. For this reason the number of rules is not a sufficient indication of the size of an expert system. Ideally, each relevant situation of the domain should be covered by just one rule, which is possible only if the rule formalism is powerful enough. The description of a situation by different rules would impair the modularity of the knowledge base, since these rules would be meaningful only as a group. On the other hand, greater expressive power of the rule formalism requires a more complex and often more inefficient rule interpreter, so that one must check how much complexity is necessary for the domain.

The simplest method of evaluating preconditions is *look-up in the data base.* For example, in the case of the query "Is the pain intensification synchronous with respiration?" it is only necessary to check whether the value "synchronous with respiration" is stored under the object "pain intensification". With the logical relations "and", "or" and "negation" many situations can be described in this way. In the case of numerical or temporal relationships looking-up alone is not sufficient, because calculations are required in addition, e.g., for the query "Did shortness of breath occur before the thorax pains?" In addition to the basic arithmetical predicates such as "greater", "smaller", "from...to", etc., it may be very useful for special applications to have more complex predicates available, e.g., for recognizing typical temporal sequences.

For an elegant formulation of a robot rule of the type "Grip the largest block on the table" one needs rules with variables which are assigned values in the precondition and used in the action. While the general form of comparison of two expressions with variables (*unification*) is seldom used in expert systems, the special case in which constants from the data base must be assigned to variables in the rules (*pattern matching*) is relatively frequent (see Fig. 5.4).

```
      (block     ^size      X
                 ^position  table)
  -   (block     ^size      {> X}
                 ^position  table)
  →   (modify 1  ^position  grip arm)
```

**Figure 5.4** Example of pattern matching in OPS5: "Grip the largest block on the table." The size of the block is denoted by the variable $X$ (^size $X$), with the additional condition (second statement) that no block on the table (^position table) is larger. The action (third statement) says that the position of the block specified in the first statement of the precondition is to be modified (^position grip arm)

| Proposition type | Evaluation method | Examples |
|---|---|---|
| including | look-up in data base | Is block-1 red? |
| increasingly | look-up in data base and calculation | Is block-1 bigger than block-2? |
| complicated | pattern matching (and calculation) | Which is the biggest red block on the table? |
| relationships | unification | — |

**Figure 5.5** Differently expressive proposition types in rules.

Figure 5.5 summarizes the complexity types of rule systems. For most expert systems looking-up in the data base and calculation or pattern matching are sufficient.

## 5.4 Rule Structuring

Like large programs, large sets of rules should also be structured with the aid of modules. For this purpose rules which belong together should be bound together in groups which can be activated as a whole. In this way, rules in diagnostic systems may be classified according to the hypotheses which they test. In construction systems such as R1 different states are assigned to different sets of rules, and only the rules of the current state are investigated at any given time. In R1 the states are called "contexts" and are always given as the first rule premise. In the RETE algorithm they have the effect that the rules are implicitly grouped during compilation. Explicit representations of such problem-specific structures is discussed in Part IV ("Classification") and Part V ("Construction").

Another form of structuring is the syntactic distinction between different types of rules and between different types of propositions in the preconditions of a rule. As an illustration we will analyze a MYCIN rule, of which we will first give the syntactic form, then the causal relationships on which it is based and finally a structured representation of the rule (after [Clancey 83, p. 236]). The syntactic form of the rule is:

If

1. the type of infection is meningitis,
2. no laboratory data are available,
3. the meningitis type is bacterial,
4. the patient is older than 17 years and
5. the patient is an alcoholic,

then there is evidence for E. Coli (0.2) and Dipplococcus (0.3).

The reason for the relationship described by the "alcoholic rule" is the following: alcoholics are more easily infected by E. Coli and Dipplococcus because their immunosystem is weakened, so that bacteria which are normally present in the intestinal flora of humans can cause meningitis.

A comparison of the causal justification with the syntactic form shows that the individual preconditions of the rule differ greatly in significance. While the first three propositions define the context in which the rule is to be applied, the fourth statement describes a precondition for asking about the fifth, really significant fact. The fourth precondition is necessary, because MYCIN has a backward chained control strategy: before the system asks whether the patient is an alcoholic, it should make sure that he or she is older than 17. Thus three different types of precondition are mixed in this rule:

- context knowledge about the applicability of the rule,
- inference knowledge, and
- dialog knowledge about the relationships between the different questions.

Taking into consideration the two types of precondition the rule may be represented in structured form as follows:

1. *Inference rule.* If the patient is an alcoholic, then there is evidence for E. Coli (0.2) and Dipplococcus (0.3).
   Context: preconditions 1–3 of the old rule.

2. *Dialog rule.* If the patient is younger than 18 he or she is not an alcoholic.

The division of the preconditions into normal and context conditions can also be exploited for improving the efficiency of the rule interpreter, since rules whose context is not fulfilled can be ignored. A further distinction between negated propositions in the precondition and exceptions will be discussed in Chap. 9 "Non-Monotonic Reasoning". Figure 5.6 shows a structured rule.

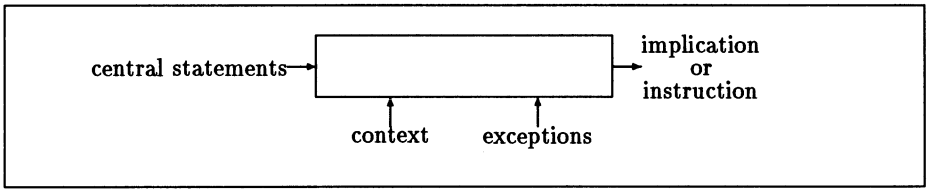



**Figure 5.6** Representation of a structured rule

## 5.5 Summary

Rule systems were introduced by Post [43] in connection with formal languages as a general computing mechanism. Their main advantages are their modularity and the adaptability of the rule formalism to concrete requirements, and their main disadvantage is that the control flow is difficult to understand. At first they were used in artificial intelligence for cognitive modeling [Newell 72].

Their great popularity in expert systems is mainly due to the development of MYCIN [Shortliffe 76] at Stanford University and its successors [Buchanan 84] and also R1 [McDermott 82,84]. Two powerful rule systems are OPS5 [Brownston 85] and PROLOG [Clocksin 81], the former with a forward-chained and the latter with a backward-chained rule interpreter. A good review of rule systems is given by [Davis 78]. On account of the lack of clarity of large rule sets ("rule spaghetti"), rules should be structured by typification, by means of contexts and above all by relating them to objects (see Chap. 6).

# 6. Objects/Frames

In simple rule systems the actual "knowledge" resides exclusively in the rules, whereas the data base is an unstructured, passive set of facts. This chapter deals with formalisms by means of which a set of facts can be better structured, economically stored and furnished with "basic knowledge" about their application. A first step in structuring is to collect together all propositions concerning a particular object in a data structure like the records in PASCAL, property lists in LISP or relations in a data base (see Fig. 6.1). The basic ideas of object-oriented

| Object | Property | Values |
|---|---|---|
| elephant | is-a: | mammal |
| | color: | gray |
| | has: | trunk |
| | size: | large |
| | habitat: | ground |

**Figure 6.1** The structured representation of facts by collecting together all the properties of an object in a data structure

(frame-based) knowledge representation are inheritance hierarchies, attached procedures and default values, which are explained briefly below.

*Inheritance hierarchies* serve for economic data conservation. Instead of storing all the properties of each object, the objects are structured in a hierarchy, and only the individual properties are stored with the object itself, while the general properties are attached to the predecessor of the object in the hierarchy and "inherited" by all successors. When requesting the properties of an object, the value is first sought in the object itself, then in its predecessor in the hierarchy, and so forth. In order to increase flexibility, object-oriented systems often use *tangled inheritance hierarchies*, in which an object inherits the properties of more than one predecessor and the selective suppression of the inheritance of individual properties is possible.

*Attached procedures* are small programs which are attached to a property of an object and are carried out when the value of that property is written or read. They can also be used for economy of data storage by having an attached procedure calculate new parameters from available ones (e.g., age from date of birth). Further, value changes can be monitored by attached procedures, e.g., by

displaying the new value on the screen each time it is changed. Such monitored values are called "active values". Attached procedures can of course be inherited within the object hierarchy in the same way as values. A generalized form of attached procedures has the property that the activation does not depend upon a change in a parameter ("methods", see Sect. 6.2).

*Default values* are pre-assigned values which are usually, but not always, correct and can therefore be overwritten in particular instances, as in the following example: Birds can fly (bird ability-to-fly yes), penguins are birds (penguin is-a bird) but cannot fly (penguin ability-to-fly no). The treatment of default values requires, however, the ability to withdraw conclusions with all consequences, which requires relatively complex procedures (see Chap. 9).

In the following sections we illustrate the basic concepts by means of the system FRL as an example, discuss the significance of the transition from passive data to active objects, consider the frame concept from the point of view of cognitive psychology and outline the automatic classification of unknown objects according to their properties in the KL-ONE family of languages. In Sect. 6.5 we indicate that generalizations of the frame idea contain experience-based problem-solving strategies for diagnostics, construction and simulation.

## 6.1 Example: FRL

FRL (**F**rame **R**epresentation **L**anguage, [Roberts 77]) is a simple language which is suitable for illustrating the basic concepts. It extends the language range of LISP with a set of functions with which one can define, fill out and change structured objects and also read out the stored information. In FRL the objects are called "frames" and their properties "slots". A slot has different "facets" – e.g. for distinguishing between its current and default values – each of which can take on a "value". The statement "Clyde's color is gray" would be represented as follows in FRL: *frame* Clyde, *slot* color, *facet* $value, *value* gray. A slot can have a total of six facets:

| | |
|---|---|
| $value: | current value of the slot, |
| $require: | value range (allowed values for $value), |
| $default: | default value (valid only if no value for $value can be found), |
| $if-added: | attached procedure in LISP code which is executed whenever a value is assigned to $value, |
| $if-removed: | attached procedure in LISP code which is executed whenever $value is deleted, |
| $if-needed: | attached procedure in LISP code which is executed whenever the value of $value is required. |

For the construction of inheritance hierarchies there is a special, pre-defined slot in FRL called "AKO" (a kind of). For example, if one wishes to express the fact that Clyde is to inherit all the properties of an elephant, this is done by writing: *frame* Clyde, *slot* AKO, *facet* $value, *value* elephant. Another example for a frame in FRL is shown in Fig. 6.2.

| Frame: | expert system | |
|---|---|---|
| Slots: | AKO: $value program | |
| | programming environment: | |
| | $require: | LISP, PROLOG, ... |
| | $default: | LISP |
| | $if-needed | "look up references" |

| Frame: | MYCIN |
|---|---|
| Slots: | AKO: $value expert system |

**Figure 6.2** An example of a frame in FRL, illustrating the function of inheritance hierarchies and default values. Question: "In which program environment was MYCIN implemented?" Answer: "Presumably LISP! Since there is no entry under the facets '$value' and '$default' of the slot 'programming environment' in the frame 'MYCIN', the default value 'LISP' is inherited from the predecessor (AKO) 'expert system' ".

Other frame languages have different types of facets, e.g., for a more exact specification of the mechanism of inheritance. A distinction is often made between generic objects (object classes) and individuals, which are instances of generic objects. This leads to a division of the AKO slots of FRL into genuine instances (individual → generic concept, e.g., Tweety is a penguin) and subsets (generic concept → generic concept, e.g., a penguin is a bird).

## 6.2 Active Objects

In FRL the frames are no longer passive data, but active objects which can carry out calculations on their own account. This makes it possible to store data much more economically than in the case of conventional data bases, since data can often be derived from inheritance hierarchies and attached procedures and therefore need not be stored explicitly. Preserving the consistency of the data base is also simplified, since the redundant storage of data can be practically avoided. If the idea of extending the data base function is followed further, even a simple forward- or backward-chaining rule interpreter can be simulated with the attached procedures, e.g., in FRL by storing the rules which derive something from data under $if-added and the rules which serve to derive data under $if-needed. With given data the execution of the $if-added facets would then provide further data, and in the case of a given goal this would be determined by the execution of the $if-needed facets. This is the basis for the extension of the data base INGRES [Stonebraker 85] by two additional operators which simulate the $if-added and $if-needed facets. The idea of treating data as active objects has been generalized in the object-oriented programming languages developed independently of FRL, such as SMALLTALK [Goldberg 83], in which a program consists of a set of objects which can send each other messages and respond by executing *methods*. Methods differ from attached procedures in that they are activated not by value changes but by messages from other objects.

Object-oriented knowledge representations can also be treated as graphs, the nodes corresponding to the objects and the links to their relations (slots). Since the links must be named on account of their different meanings, the graphs are also called *semantic nets.* In agreement with [Nilsson 82, Sect. 9.2] we do not regard semantic nets as an independent knowledge representation, but only as a graphical form for displaying object-oriented knowledge representations, since a node with its links to other nodes is represented internally as an object with properties and values.

To the user of a program it makes no difference whether the data are stored explicitly as in a data base or derived implicitly from other data with procedures. A suitable level of abstraction for describing data structures is provided by *abstract data types* [Aho 83], in which a data type is characterized purely functionally by its construction, modification and reading functions, the actual implementation of which remains open.

## 6.3 Cognitive Meaning of Frames

The idea of active objects also has a cognitive root, which was emphasized in Minsky's [75] classical paper on frames. Minsky postulates that when people enter a living room, for example, or take part in a children's birthday party or experience some other stereotype situation, a cognitive data structure developed from previous experience is activated and forms the frame for understanding the current situation. The slots of the frame are represented by relevant properties of the situation such as the number and positions of the windows and the furniture in a living room. Typical features are given as default values: for example one expects a sofa, two armchairs, a couch table, etc. in a living room. Similarly for a children's birthday party there are default values for the decoration (balloons) and the food (cake and ice-cream). The frame selected for the situation is confirmed if values can be assigned to the individual slots and in particular if the default values are valid. If certain default values are found to be incorrect, "excuses" are sought or the actual values are accepted as deviations from the norm. However, if too many discrepancies occur the frame must be replaced by another one which fits better. An example from [Minsky 75] shows the usefulness of the frame concept for understanding everyday stories: *Jane was invited to Jack's birthday party. She wondered if he would like a kite. A friend tells Jane: "He already has a kite. He will make you take it back."* Although the personal pronoun "it" refers syntactically to the old kite, it may be assumed from the frame structure for children's birthday parties that only presents are taken back, so that "it" must refer to the new kite.

According to Minsky, frames serve (1) for recognizing stereotype objects such as a living room, (2) for acting in stereotype situations such as a children's birthday party and (3) for answering questions about stereotype and concrete objects. Only the third aspect is considered in most frame representations, e.g., FRL. The main problem with the first aspect, the recognition of objects, is the question of default values. Since default values can be overwritten it is difficult to use them

for the recognition of objects, as Brachmann [85a] noted in a polemical article: "What's big, gray, has a trunk and lives in the trees? An Elephant – I lied about the trees (i.e., the trees are an overwritten default value)."

A radical consequence of this is to forbid default values and deviations in the definitions of objects, which is the basic idea of the KL-ONE languages described in the following section. This makes automatic classification possible, i.e., fitting an object described only by its properties into an existing, precisely defined object hierarchy.

## 6.4 Automatic Classification in KL-ONE Languages

The central idea of the KL-ONE languages (KL-ONE, KRYPTON, KL-TWO, NIKL, BACK, MESON,...; reviews are given in [Brachman 85b] and [von Luck 91]) is the strict division of statements about an object (in KL-ONE "concept") into defining and optional properties and the distinction between generic concepts in the T box (terminological box) and instances in the A box (assertion box). For the present we are interested only in the defining properties in the T box, which make automatic classification possible. There the user can no longer declare a frame as a sub-frame of another frame by a simple assignment to an AKO slot as in FRL, but must specify those properties (in KL-ONE "roles") with respect to which a concept is more special than its predecessor.

In KL-ONE the generic concepts are either primitive or defined through other concepts and characterized by a number of roles. The roles, which may also be primitive or defined, represent a mapping between two concepts and have specifications for value range and cardinality, which may be restricted in the definition of new concepts. The cardinality of roles in KL-ONE languages is a new characteristic which is not contained explicitly in FRL or predicate logic.

The most important language structures, common to all members of the KL-ONE languages, are given below (graphical representation in Fig. 6.3):

1. Definition of concepts:

   - Conjunction (conj-generic): A concept is the unification of the properties of several concepts.
     Example: bachelor = conjunction(single-person man)
   - Value restriction (VR-generic): The set of values of a role is restricted with respect to a higher-ranking concept.
     Example: old-town = value-restriction(quarter has-buildings old-houses) (a quarter in which the buildings are old houses).
   - Number restriction (NR-generic): The cardinality of a role is restricted with respect to a higher-ranking concept.
     Example: big-family = number-restriction(family has-children$[3, \infty]$) (a family with at least three children)

- Specialization (prim-generic): Specialization without sufficient condition.
  Example: man = specialization(person) (A man is a person, but it is not specified which additional properties men have compared to persons.)

2. Definition of roles:

   - value differentiation (VR diffrole): A role whose set of values is restricted.
     Example: has-daughter = value-differentiation(has-child woman) (a child who is a woman)
     has-father = value-differentiation(has-parent man)
   - Chaining (role chain): A role which is the chaining of several roles.
     Example: has-grandchild = chaining(has-child has-child)
   - Specialization (prim-role): Specialization without sufficient condition.
     example: has-technical-profession = specialization(has-profession)

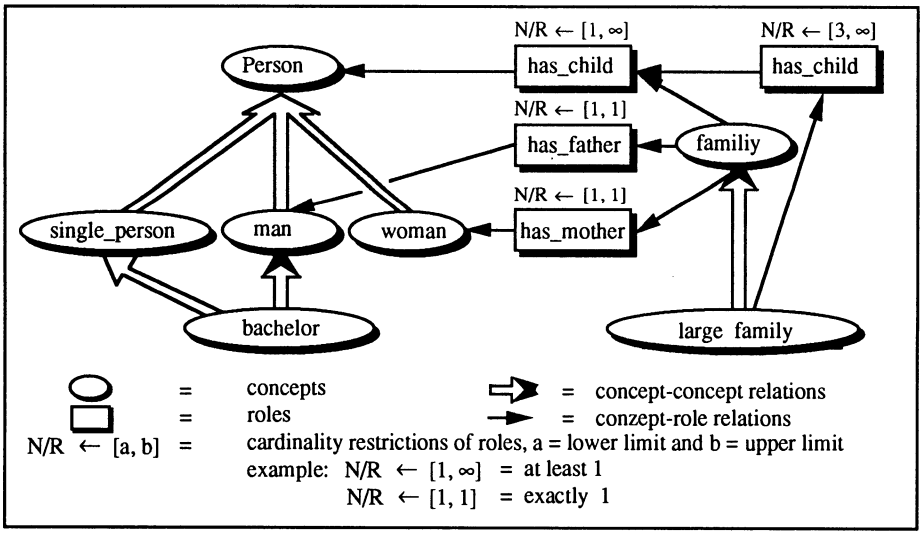



**Figure 6.3** Graphical representation of the concepts bachelor and big-family in KL-ONE

Fitting new concepts into an existing classification hierarchy is the task of the *classifier*[1], the central element of KL-ONE languages. It consists of two components:

1. A predicate "subsumption" which is true if one concept subsumes another, i.e., all the roles of a concept subsume those of another. A role of a concept

[1]In addition to the classifier, which orders generic concepts, recent KL-ONE dialects often have a "realizer", which recognizes individuals of the A box as instances of generic concepts.

subsumes a role of another concept if both the value range and the cardinality of the role of the higher-ranking concept include those of the role of the lower-ranking concept.

2. A search algorithm for finding the most specific concept (SC) which still subsumes the new concept (NC), i.e., the predicate (subsumption NC SC) is true, but (subsumption NC SC′) is false for all subconcepts SC′ of SC.

The complexity of the classifier depends very sensitively on the expressive power of the language elements. The language structures described above (slightly extended, see Fig. 3.1.2 in [Brachman 83]) very probably lead to a super-exponential complexity of the classifier, which can be reduced at least to $O(n^2)$ by leaving out the value restrictions for roles. However, a classifier may work quite well in "normal" cases even if the complexity is super-exponential. One should therefore carefully choose the best compromise between the required expressive power of the language and the efficiency of the classifier [Levesque 85].

On account of the limited expressive power it is possible, in recent KL-ONE dialects, to define optional properties for a concept which are irrelevant for the classification. An example of this is KRYPTON, in which the T box consists of the above-mentioned language concepts and is coupled to a general theorem prover in first-order predicate logic for evaluating the A box.

The KL-ONE languages have the great advantage of functional, implementation-independent semantics, which makes the classifier possible. However, the price for this, the elimination of all vague relationships and exceptions in the definition of concepts, is very high. In many applications of expert systems the important relationships cannot be described in the way required for the T box of KL-ONE. Haimowitz [88] gives a report of the numerous difficulties encountered in mapping the ABEL knowledge base in NIKL (see Sect. 17.4).

## 6.5 Frames and Problem-Solving Types

In this section we wish to point out that the problem-solving strategies for heuristic classification (Chap. 15) and skeletal construction (Chap. 22) can be regarded as a further development of Minsky's frame ideas for the recognition of stereotyped objects and for reacting to stereotyped events. Heuristic classification is based on the assumption that the identity of a new object can be decided only by comparison. The attachments of properties to a concept are therefore qualified by probabilities and exceptions. The fitting of a new object into the existing concept hierarchy can no longer be proved as in the case of the classifier in KL-ONE, but is derived probabilistically (Chap. 8) or non-monotonically (Chap. 9). Figure 6.4 shows the frame representation of a diagnosis, in which the relationships to symptoms or other diagnoses are represented by attached procedures (rules).

Planning problems in stereotype situations can also be solved frequently with a frame-like skeletal plan. The latter consists of series of general instructions,

which only need to be refined in a suitable way. Further, problem-solving strategies for simulation (see Part VI) are often based on the idea of active objects which exchange messages and change as a result.

Properties of a diagnosis frame:

- *a priori* probability
- Rules for hypothesis generation
- Rules for categorical evaluation
- Rules for probabilistic evaluation
- Examinations for confirming diagnosis
- Predecessor diagnoses
- Successor diagnoses
- Competing diagnoses
- Therapies

**Figure 6.4** Frame representation of a diagnosis

## 6.6 Summary

Starting from a discussion of the structure of the data base of a rule interpreter it was shown that it may be of advantage not to store all data explicitly but to derive many data by means of simple rules of deduction. A generalization of this idea of active objects leads to the point of view that experience-based problem-solving strategies for diagnostics, construction and simulation can be regarded as instantiating, refinement or alteration of objects (frames). The broad applicability of frames and their cognitive adequacy were shown by Minsky[75] in his classical frame paper.

It is therefore not surprising that object-oriented ideas have been used in a great number of innovations in programming techniques. In a broader sense these include (1) data bases; (2) "data bases" with built-in inferences such as FRL [Roberts 77] or Stonebraker's [85] extension of INGRES; (3) object-oriented extensions of existing programming languages (e.g., CLOS from LISP or C++ from C); (4) special object-oriented programming languages such as SMALLTALK [Goldberg 83]; (5) KL-ONE languages for the automatic classification of objects according to their precisely defined properties [von Luck 91]; (6) abstract data types for the implementation-independent description of the functionality of objects; (7) semantic nets for graphical representation.

# 7. Constraints

Constraints serve for the representation of relationships between variables and are particularly suitable for representing local boundary conditions which the problem solution must fulfill in any case regardless of the particular solution. For example, one can regard the wish of a teacher to have one day in the week free as a constraint in setting up the timetable of lessons. Each new constraint brings about a further restriction of the solution space. The aim of constraint systems is to find a solution which takes all constraints into consideration, e.g., setting up a timetable which fulfills all boundary conditions.

EL [Stallman 77], one of the best-known constraint systems, is used for the simulation of electrical circuits. An electrical resistance, for example, is treated in EL as a constraint between the input and output voltages and the current (see Fig. 7.1). As often happens in the field of expert systems, EL was quickly developed further to an application-independent constraint language [Sussman 80].

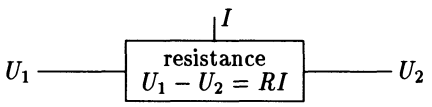



**Figure 7.1** Example of a constraint in EL: A resistance $R$ causes a drop in voltage according to Ohm's Law

In EL all circuit components are represented as constraints and connected by common variables (voltages and currents) to form a constraint network. If the voltage and current of the power source are known, the values are propagated through the constraint network in the same way as a real current, so that a value can be calculated for each variable.

While constraints express non-directional relationships between variables which can usually be resolved explicitly for each variable (e.g., $U = RI$), rules represent directed relationships (e.g., $A \Rightarrow B$[1]).

A constraint can often be regarded as a mathematical equation, and the constraint network as a system of simultaneous equations. The solution of the simultaneous equations corresponds to the propagation of the constraints. However, constraints are not restricted to equations; they can also express inequalities and

[1]From the validity of $B$ it is not possible to deduce anything about the validity of $A$.

non-numeric relationships. Constraints are therefore excellently suited for quantitative or qualitative modeling of systems and physical relationships.

## 7.1 Types of Constraints and Propagation Algorithms

Like rules, constraints are based on a very general idea for which there are very many different realizations. For the representation of a constraint such as the relationship between the height and weight of a person, for example, one has the following possibilities, amongst others:

- Tables
- Functions, e.g., normal-weight[kg] = height[cm] − 100
- Predicates, e.g., for "under-weight": $\frac{\text{actual-weight}}{\text{normal-weight}} < 0.8$

Constraints can be implemented as data bases (for tables), as rules (e.g., if $A$ and $B$ are known, then $C := A + B$; if $A$ and $C$ are known, then $B := C - A$; etc.) or as any general program. The input of the constraint propagation algorithm is a constraint network in which some of the variables are assigned values, and the output is the assignment of values consistent with the constraints to further variables. If all variables take exactly one value the solution is unique. Sets of values for the variables mean multiple solutions, and the assignment of an empty set to a variable means that there is an inconsistency. The essential feature of the propagation is that the restriction of the set of values of a variable due to its associated constraints is passed on to other variables until no further restriction of the value range of any variable is possible.

Propagation algorithms may be distinguished according to what can be propagated through a variable:

- fixed values only (e.g., $X = 5$),
- sets of values (e.g., $X \in \{3, 4, 5, 6\}$) or
- symbolic expressions (e.g., $X = 2Y$).

They also differ according to whether assumptions may be made for case distinctions, which must be revised in the case of a contradiction. For this a procedure for the revision of conclusions is required (see Chap. 9 "Non-Monotonic Reasoning"). For the propagation of fixed values or value sets simple forward chaining is usually sufficient as a control strategy as for rules. The introduction of symbolic expressions and case distinctions, on the other hand, must be controlled heuristically.

## 7.2 Solution of a Non-Trivial Constraint Problem

A well-known heuristic problem (see Fig. 7.2), discussed informally below, may serve to illustrate the various propagation algorithms and acquaint the reader with the handling of constraint systems. The first and perhaps most difficult problem

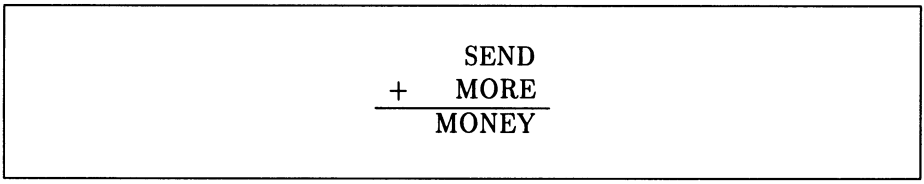



**Figure 7.2** The student's cry for help to his father

is to find an elegant formulation of the task. The given representation as a single equation with eight unknowns is not helpful. Instead we should set up from this equation five equations (one for each column). One possible formulation of the constraints is a mixture of functions and predicates, e.g., for the last column "if $(D + E < 10)$ then $(D + E = Y)$ else $(D + E = Y + 10)$." However, this formulation becomes very cumbersome for the other columns since more and more case distinctions are needed. A much more elegant description, which is already half the solution, is shown in Fig. 7.3, where the amounts carried over to successive columns are represented by new variables $U_1 \ldots U_4$.

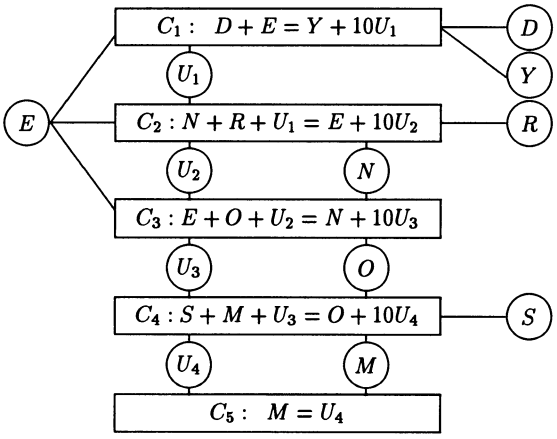



**Figure 7.3** Constraint representation with five constraints $C_1 \ldots C_5$ and four new variables $U_1 \ldots U_4$ for the problem of Fig. 7.2. There is an additional condition that no two of the variables $\{D, E, M, N, O, R, S, Y\}$ may have the same value.

The value ranges of the variables are:

$$\begin{array}{lll} D, E, N, O, R, Y & \in & \{0,1,2,3,4,5,6,7,8,9\}, \\ M, S & \in & \{1,2,3,4,5,6,7,8,9\}; \text{ and} \\ U_1, U_2, U_3, U_4 & \in & \{0,1\}. \end{array}$$

The intersection set of the value ranges of $M$ and $U_4$, which according to $C_5$ represent the same digit, gives immediately $U_4 = M = 1$. Thus the digit 1 can be removed from the value sets of all the other variables (this step is not always mentioned explicitly below), and equation $C_4$ simplifies to $S + U_3 = O + 9$. Thus no further fixed values can be propagated, since in the propagation of fixed values a constraint is normally activated only if the values of $n - 1$ out of $n$ variables are known. We therefore have recourse to the method of case distinction.

Assumption: $U_3 = 1$

$\Rightarrow$ $C_4$ simplifies to $S = O + 8$. From this follow $S \in \{8,9\}$ and $O = 0$, since $M = 1$ requires $O \neq 1$. Hence $C_3$ is also simplified to $E + U_2 = N + 10$, which cannot be satisfied by any consistent value for $E$, since $E < 10$. Thus the assumption $U_3 = 1$ was false, and this and all derived values must be reset.

Hence follows the new assumption $U_3 = 0$

$\Rightarrow$ $C_4$ simplifies to $S = O + 9$, for which the only consistent solution is $S = 9$ and $O = 0$. For $C_3$ we then have $E + U_2 = N$. The assumption $U_2 = 0$ leads immediately to the contradiction $E = N$, thus $U_2 = 1$ and hence $E + 1 = N$. Since $C_2$ also contains the variables $E$ and $N$, and no fixed values can be calculated, further progress can best be made here with a symbolic propagation, replacing the variable $N$ in the constraint $C_2$ by $E + 1$. $C_2$ then reduces to $R + U_1 = 9$. A case investigation for $U_1$ gives uniquely $U_1 = 1$ and $R = 8$. For $C_1$ we then have $D + E = Y + 10$. Since each of the remaining variables $D, E, N, Y$ can still take six values, case distinction does not seem reasonable, and a symbolic propagation is also of no help. On the other hand, the propagation of sets of values is helpful. The present sets are $D, E, N, Y \in \{2,3,4,5,6,7\}$. The equation $E + 1 = N$ has the effect that when the set is propagated from $E$ to $N$ the value 2 drops out for $N$, and the reverse propagation results in 7 having to be deleted for $E$. From the equation $D + E = Y + 10$ one may derive the inequality $D + E < 11$, which leaves the values $\{6,7\}$ for $D$ and $\{5,6\}$ for $E$ when the set is propagated. Since now only 2 values are left for $E$, a simple case distinction is again possible. $E = 6$ implies $D = 7$ and $N = 7$, which is a contradiction. Therefore $E = 5$, which gives directly $N = 6$, $D = 7$ and $Y = 2$. The unique solution of the problem is shown in Fig. 7.4.

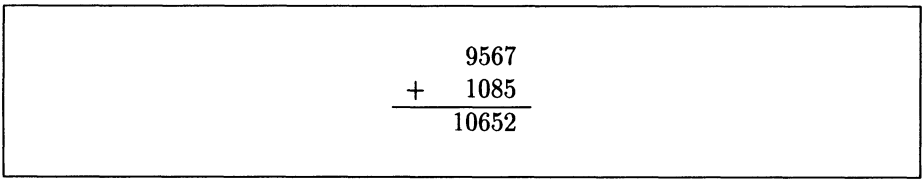



**Figure 7.4** The solution of Fig. 7.2

## 7.3 Discussion of the Solution

The very economical solution of the constraint problem discussed above can be found only by very efficient constraint systems:

- A simple constraint system propagating only *fixed values* would not find any solution, because after the first step all constraints have more than one unknown variable.
- A fairly simple constraint system which differs from a simple constraint system in that it can also make *case distinctions* would find a solution only after much random guessing. If the value sets of the variables are infinite (e.g., real numbers), a case distinction is basically impossible.
- A constraint system which can also propagate *value sets* would considerably reduce the search space of the simple system but would still need to do much guessing.
- The technique of *symbolic propagation*, which is very effective for complex problems, is sufficient, but very difficult to implement, since the system must also be able to carry out algebraic transformations. In problems like the example discussed above this would be relatively easy, however, since only the arithmetic operations addition and subtraction are needed.
- Even if a system has all techniques at its disposal, it still needs good heuristics to determine when to apply which techniques. A case distinction, for example, is suitable only if a variable has a very small value set and the cases do not branch further.

## 7.4 Formal Characterization of Simple Constraint Systems

The following short formal characterization of constraints was taken from [Voss 89]. A constraint is a triad (name, variables, definition), where the definition describes a relationship between the variables, e.g., for the constant resistance $R$ in Fig. 7.1: $\{\text{“resistance”}, \{U_1, U_2, I\} \in \mathbb{R}, (U_1 - U_2 = R \cdot I)\}$. A constraint

problem consists of a set of constraints, connected by common variables, and a set of initial values for the variables. A solution of the constraint problem is a maximal restriction of the value ranges of all variables. A simple constraint propagating algorithm which propagates only fixed values and value sets, is shown in Fig. 7.5.

Input: constraint problem (constraint network and initial values).
Output: maximal consistent assignment of values or value sets to all variables.

1. Assign the initial values to all variables.
2. Activate all constraints in which the variables appear and put them on the list ACTIVE.
3. Select a constraint from ACTIVE and deactivate it. If ACTIVE is empty: STOP (result = current values of variables).
4. Calculate the new value sets for the variables concerned with the selected constraint. If the value set of a variable is empty: STOP (contradiction).
5. Put all constraints referring to the variables whose values sets were restricted on ACTIVE.
6. GO TO 3.

**Figure 7.5** Constraint propagating algorithm for fixed values and value sets

## 7.5 Summary

Constraints represent relationships between objects. They constitute an important representation method for planning and simulation systems (see Parts V and VI). Constraints can be implemented with tables, rules or free program code. Techniques for solving a constraint system are simple propagation with fixed values or value sets, case distinction and symbolic propagation (usually difficult to implement). One of the first and most ambitious constraint systems was EL [Stallman 77]. Other well-known constraint systems are the Waltz algorithm for edge marking in drawings [Waltz 75] and the Allen algorithm for the propagation of temporal relationships (see Sect. 10.3). General constraint languages are described in [Sussman 80] and [Güsgen 88]. A formal representation of constraint propagating methods is given in [Voss 89] and [Hentenryck 89].

# 8. Probabilistic Reasoning

In classical logic it is usually assumed that a proposition is either true or false, whereas in real life the truth value of a proposition may be known only with a certain probability or not at all. Since such uncertainties are often encountered in the field of expert systems, the knowledge representation and especially the problem-solving strategy must be extended accordingly. The two basic approaches are probabilistic reasoning and non-monotonic reasoning, which is discussed in the next chapter. The basis of probabilistic reasoning [Genesereth 87] is the attachment of a probability to each proposition to express the uncertainty. The uncertainties may be derived from representative statistics or estimated by experts.[1]

The most important domain for the treatment of uncertainties is diagnostics, where it is required to recognize patterns (diagnoses) on the basis of their properties. The uncertainties in diagnostics arise from the following sources:

- ascertainment of symptoms (determining the symptomatic evidence),
- evaluation of symptoms (relating symptoms to diagnoses), and
- inadequacies in the reckoning scheme.

In expert systems the uncertainties in the ascertainment of symptoms are usually estimated by the user and the uncertainties in the evaluation of symptoms by experts. Since different people will judge these uncertainties differently, and a normalization is not possible, the reckoning enlarges the total uncertainty considerably. In order not to make the accuracy appear better than it actually is, simplified assumptions are often made in expert systems by distinguishing only a few degrees of uncertainty (e.g., certain, probable, possible, etc.). In general, the reckoning schemes are also correspondingly simple. These methods are often called "ad hoc representations" by statisticians on account of the lack of theoretical foundation. On the other hand, the conditions for the application of well-founded statistical procedures for probabilistic reasoning are far from being fulfilled in most application domains. One cannot therefore expect an ideal formalism for treating uncertainties, and this fact is reflect in the variety of existing

[1] If we wish to distinguish between statistically derived uncertainties and those estimated by people, we use the term probabilistic (in the narrow sense) only for the former case and denote the latter as uncertain reasoning with evidences or certainty factors. In contexts where this distinction is of no importance both terminologies will be used synonymously.

approaches. In the following sections a selection of procedures will be described from the statistical and the non-statistical field: Bayes' Theorem, the Dempster-Shafer Theory, the INTERNIST model, the MYCIN model and the MED1 model.

Although the various methods differ greatly in detail they are founded on the same basic algorithm for the assessment of diagnoses (see Fig. 8.1).

1. Start with the *a priori* probabilities of all diagnoses.
2. For each symptom modify the probabilities of all diagnoses according to the symptom-diagnosis probabilities.
3. Select the most probable diagnosis.

**Figure 8.1** Basic algorithm for assessing diagnoses.

For the application one requires calculations or estimates of the symptom-diagnosis probabilities for all relevant symptom-diagnosis pairs and the symptom-independent *a priori* probabilities of the diagnoses.

## 8.1 Bayes' Theorem

Bayes' Theorem [Charniak 85, Sect. 8.2] enables the most probable diagnosis assuming the symptoms $S_1 \ldots S_m$ to be determined from the *a priori* probabilities $P(D_i)$ of a set of diagnoses and the conditional probabilities $P(S_j/D_i)$, i.e., the relative frequency of the occurrence of a symptom $S_j$ in the case of the diagnosis $D_i$. The relative probability $P_r$ of a diagnosis compared to the other diagnoses is calculated by the formula given in Fig. 8.2. The probabilities on the right hand side of this formula can be determined from case data bases.

$$P_r(D_i/S_1 \& \ldots \& S_m) = \frac{P(D_i) \cdot P(S_1/D_i) \cdot \ldots \cdot P(S_m/D_i)}{\sum_{j=1}^{n} P(D_j) \cdot P(S(1)/D_j) \cdot \ldots \cdot P(S_m/D_j)}$$

**Figure 8.2** The form of Bayes' Theorem mainly used for diagnostics

We will now examine the conditions under which the application of Bayes' Theorem is correct. We first consider the situation where there is only one symptom and one diagnosis, and extend the formula in steps to the form of Fig. 8.2.

The conditional probability $P(S/D)$ expresses the relative frequency of the occurrence of the symptom $S$ when the diagnosis $D$ is valid:

$$P(S/D) = \frac{|S \cap D|}{|D|}, \quad \text{where:}$$

$$\begin{aligned} |S| &= \text{frequency of the symptom } S \\ |D| &= \text{frequency of the diagnosis } D \\ |S \cap D| &= \text{frequency of simultaneous occurrence of } S \text{ and } D. \end{aligned}$$

The probability $P(D/S)$ of a diagnosis $D$ assuming a symptom $S$ is given by

$$P(D/S) = P(D) \cdot \frac{P(S/D)}{P(S)}.$$

Proof:

$$\begin{aligned} P(D/S) &= \frac{|S \cap D|}{|S|} = \frac{|D|}{|D|} \cdot \frac{|S \cap D|}{|S|} = |D| \cdot \frac{|S \cap D|}{|D|} \cdot \frac{1}{|S|} \\ &= P(D) \cdot \frac{P(S/D)}{P(S)}. \end{aligned}$$

Example: Suppose statistics show that the following probabilities hold:
$P(\text{bronchitis}) = 0.05$
$P(\text{cough}) = 0.2$
$P(\text{cough/bronchitis}) = 0.8$
i.e., if a patient has bronchitis, coughing is found in 80% of all cases.
Then we have:

$$\begin{aligned} P(\text{bronchitis/cough}) &= P(\text{bronchitis}) \cdot \frac{P(\text{cough/bronchitis})}{P(\text{cough})} \\ &= 0.05 \cdot \frac{0.8}{0.2} = 0.2. \end{aligned}$$

Thus the probability that a patient with a cough has bronchitis is 20%, i.e., four times the *a priori* probability of bronchitis.

If many symptoms instead of a single one are evaluated, the correlations of every relevant combination of symptoms to diagnoses should strictly be determined. However, this would lead to an explosion in the number of combinations: with 50 symptoms we have already $2^{50}$ different constellations. It is therefore assumed that the symptoms – except when they are caused directly by the same diagnosis – are independent of one another, i.e., for two symptoms $S_i$ and $S_j$

$$P(S_i \& S_j/D) = P(S_i/D) \cdot P(S_j/D).$$

From this follows

$$\begin{aligned} P(D/S_1 \& \ldots \& S_m) &= P(D) \cdot \frac{P(S_1 \& \ldots \& S_m/D)}{P(S_1 \& \ldots \& S_m)} \\ &= P(D) \cdot \frac{P(S_1/D) \cdot \ldots \cdot P(S_m/D)}{P(S_1 \& \ldots \& S_m)}. \end{aligned}$$

Note that the denominator $P(S_1\& \ldots \& S_m)$ cannot be resolved, since these symptoms are not fully independent of one another, as they are in fact caused by the same diagnosis.

In general one is not interested in the absolute probability of a diagnosis, but in the difference between the most probable diagnosis and its competitors. If we assume that the set of diagnoses is complete and that the diagnoses mutually exclude one another, the relative probability $P_r$ of a diagnosis with respect to its competitors is

$$
\begin{aligned}
P_r(D_i/S_1\& \ldots \& S_m) &= \frac{P(D_i/S_1\& \ldots \& S_m)}{\sum_{j=1}^{n} P(D_j/S_1\& \ldots \& S_m)} \\
&= \frac{\left(\frac{P(D_i) \cdot P(S_1/D_i) \cdot \ldots P(S_m/D_i)}{P(S_1\& \ldots \& S_m)}\right)}{\left(\frac{\sum_{j=1}^{n} P(D_j) \cdot P(S_1/D_j) \cdot \ldots \cdot P(S_m/D_j)}{P(S_1\& \ldots \& S_m)}\right)} \\
&= \frac{P(D_i) \cdot P(S_1/D_i) \cdot \ldots \cdot P(S_m/D_i)}{\sum_{j=1}^{n} P(D_j) \cdot P(S_1/D_j) \cdot \ldots \cdot P(S_m/D_j)}.
\end{aligned}
$$

As we have seen, the application of Bayes' Theorem in the form of Fig. 8.2 is correct only when numerous conditions are fulfilled, which may be summarized as follows:

1. *The symptoms may only depend upon the diagnoses and must be independent of one another.* This is the critical point and becomes more and more problematical, the more symptoms are ascertained.

2. Completeness of the set of diagnoses.

3. *Mutual exclusion of the diagnoses* (single fault assumption). This assumption is justified only in relatively small diagnosis sets.

4. As far as possible *correct and complete statistics* for deriving the *a priori* probabilities of the diagnoses and the conditional symptom-diagnosis probabilities.

5. *Constancy of the probabilities.* This is problematical, not only because the appearances of the diagnoses may change, but also because of changes in the diagnostic methods. New diagnostic techniques, changes in the training of the diagnostician or a different division of labor can lead to new preselections and hence to different *a priori* probabilities of the diagnoses.

Since the conditions are generally not fulfilled, the application of Bayes' Theorem usually suggests a much greater exactitude than is generally possible in practice.

Once one has accepted the fact that an exact treatment of probabilities is not possible in most domains, on can consider how the sources of error in Bayes'

Theorem can be reduced by use of additional knowledge. Admittedly the theoretical foundation becomes more difficult with each new mechanism. Below we give a summary of the mechanisms which are explained for example systems in the rest of the chapter.

- Symptom combinations can be taken into account by rules such as: $A\&B\&C \xrightarrow{X} B$ (see MYCIN and MED1 models).
- The condition that diagnoses should be mutually exclusive (single fault assumption) can be weakened by partitioning the set of diagnoses. Multiple diagnoses can be recognized by the fact that one diagnosis cannot explain all symptoms (see INTERNIST Model).
- With Bayes' Theorem the confidence of the result cannot be estimated. This can be achieved by stipulating probability intervals instead of probabilities for the diagnoses (see Dempster-Shafer Theory). Another possibility when there is great uncertainty is first to establish only rough diagnoses and refine them later as more knowledge becomes available.
- In Bayes' Theorem a symptom which supports a diagnosis to $X\%$ is valued at $(100-X)\%$ against it. Intuitively it seems more reasonable to evaluate the positive and negative evidences of a diagnosis separately (see all following formalisms).
- Even a rigid reckoning scheme is a source of uncertainty if its conditions are not fulfilled. Therefore one can offer different reckoning schemes for evaluating different diagnoses (see MED1 model):
- Finally one can try to abolish the uncertainties altogether and replace them by detailed knowledge such as exceptions from rules and causal models (see summary in Sect. 8.6).

## 8.2 The Dempster-Shafer Theory

The Dempster-Shafer Theory [Gordon 85] differs from the application of Bayes' Theorem in the following points:

- Coarse diagnoses (classes of diagnoses) and tangled diagnosis hierarchies can also be represented. A class of diagnoses is represented as a set of fine diagnoses (see Fig. 8.3).
- The valuation is based on conditional symptom-diagnosis probabilities $P(D/S)$ instead of $P(S/D)$ as in Bayes Theorem.
- The difference of $(100-X)\%$ with respect to a probability of $X\%$ for a diagnosis is not valued against the diagnosis, but interpreted as an uncertainty interval, i.e., the probability of a diagnosis is not $X\%$, but lies between $X\%$ and 100%.

- A probability against a diagnosis is represented as a probability for the complement of the diagnosis, i.e., the set of all other diagnoses, and has the effect that the uncertainty interval is reduced from the top (i.e., the maximum probability is correspondingly smaller than 100%).

In the Dempster-Shafer Theory the evaluation of probabilities is more complicated than in the application of Bayes Theorem, since probabilities for sets of diagnoses must be related to one another, and the probability of a particular set of diagnoses must be calculated from the distribution of probabilities over all sets of diagnoses. The reckoning scheme may be explained by means of an example with four hierarchically structured diagnoses (see Fig. 8.3).

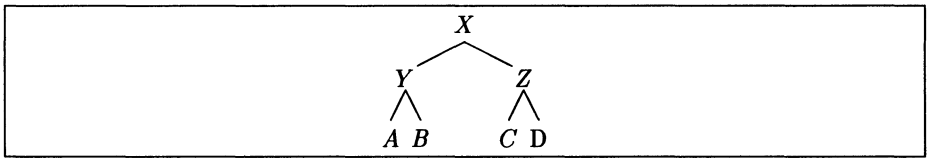



**Figure 8.3** Example of a hierarchy: $X = \{A, B, C, D\}, Y = \{A, B\}$ and $Z = \{C, D\}$ are classes of diagnoses

Suppose we have the following starting probabilities, which may be *a priori* probabilities or conditional symptom-diagnosis probabilities: $E_1$: against $A$ 30%; $E_2$: for $Y$ 60%. Evidence against $A$ means evidence for the complement of $A$, which in this case consists of the set of diagnoses $\{B, C, D\}$. Evidence for $Y$ is evidence for the set $\{A, B\}$.

When assigning evidences for a set of diagnoses the remaining probability (the difference to 100%) is always applied to the set of all diagnoses, in this case therefore to $\{A, B, C, D\}$. When evidences are coupled, the probabilities are multiplied and the intersection sets are formed. This gives the following matrix:

| $E_1 \downarrow$ | | $E_2 \rightarrow$ | $\{A, B\}$ | = | 0.6 | $\{A, B, C, D\}$ | = | 0.4 |
|---|---|---|---|---|---|---|---|---|
| $\{B, C, D\}$ | = | 0.3 | $\{B\}$ | = | 0.18 | $\{B, C, D\}$ | = | 0.12 |
| $\{A, B, C, D\}$ | = | 0.7 | $\{A, B\}$ | = | 0.42 | $\{A, B, C, D\}$ | = | 0.28 |

The total probability of a set of diagnoses is expressed by means of an uncertainty interval $[a, b]$, which is an estimate of the upper and lower probability limits.

$a$ = the sum of the probabilities of the set and all its subsets.

$b$ = the difference between 100% and the sum of the probabilities of the complement of the set and the complements of all its subsets.

For the diagnoses $\{A\}$ and $\{B\}$ we thus obtain

- $\{A\} = [\{A\}, 1 - (\{B, C, D\} + \ldots + \{B\})] = [0, 0.7]$, i.e., the probability for $\{A\}$ lies between 0 and 70%.
- $\{B\} = [\{B\}, 1 - (\{A, C, D\} + \{A, C\} + \{A, D\} + \ldots)] = [0.18, 1]$, i.e., the probability for $\{B\}$ lies between 18 and 100%.

If a further evidence $E_3$ of 50% for $\{D\}$ is added, the distribution of probabilities is changed as follows:

| $(E_1 \& E_2) \downarrow$ | | $E_3 \rightarrow$ | $\{D\}$ | = | 0.5 | $\{A,B,C,D\}$ | = | 0.5 |
|---|---|---|---|---|---|---|---|---|
| $\{B\}$ | = | 0.8 | $\{\}$ | = | 0.09 | $\{B\}$ | = | 0.09 |
| $\{A,B\}$ | = | 0.42 | $\{\}$ | = | 0.21 | $\{A,B\}$ | = | 0.21 |
| $\{B,C,D\}$ | = | 0.12 | $\{D\}$ | = | 0.06 | $\{B,C,D\}$ | = | 0.06 |
| $\{A,B,C,D\}$ | = | 0.28 | $\{D\}$ | = | 0.14 | $\{A,B,C,D\}$ | = | 0.14 |

The empty subset $\{\}$ signifies that no diagnosis applies. Under the assumption that the set of diagnoses is complete, the empty set can be eliminated by dividing all other probabilities by the factor (1 - probability of the empty subset), in our example 1 - 0.3 = 0.7, so that the sum of all probabilities without the empty subset is 1. We thus obtain:

$$\begin{aligned}
\{B\} &= \frac{0.09}{0.7} \approx 13\% \\
\{D\} &= \frac{0.2}{0.7} \approx 28.5\% \\
\{A,B\} &= \frac{0.21}{0.7} = 30\% \\
\{B,C,D\} &= \frac{0.06}{0.7} \approx 8.5\% \\
\{A,B,C,D\} &= \frac{0.14}{0.7} = 20\%
\end{aligned}$$

The probabilities for $\{A\}$ and $\{B\}$ have changed as follows:

- $\{A\} = [\{A\}, 1 - (\{B,C,D\} + \ldots)] = [0, 0.5]$, i.e., between 0 and 50%.
- $\{B\} = [\{B\}, 1 - (\{A,C,D\} + \ldots)] = [0.13, 0.715]$ i.e., between 13 and 71.5%.

The uncertainty intervals, which are very broad with little information and become increasingly narrower with additional information, and the representation of diagnosis hierarchies are among the advantages of the Dempster-Shafer Theory. The basic assessment by means of symptom-diagnosis instead of diagnosis-symptom probabilities makes the determination of the probabilities slightly more difficult, because the case data bases are usually sorted according to diagnoses. The main disadvantage, however, is that the amount of computation required is enormous, since with $n$ diagnoses $2^n$ sets have to be considered. In [Gordon 85] it is shown that the computational effort can be drastically reduced by sticking to strict, non-tangled hierarchies instead of using tangled diagnosis hierarchies. Another disadvantage is that the Dempster-Shafer Theory, like Bayes' Theorem, has no efficient mechanism for taking into account symptom correlations and multiple diagnoses.

## 8.3 The INTERNIST Model

Since the conditions for statistical reckoning schemes are rarely fulfilled to such a degree that their application brings good results, most diagnostic expert systems use simplified schemes, which are generally based on probabilities estimated by experts and make much less claim to exactitude. A simple and well-proved model is that of INTERNIST, with which Dr. Myers has developed one of the largest knowledge bases for expert systems in internal medicine [Miller 82].

The INTERNIST scheme differs from the application of Bayes' Theorem in the following points:

- Separate assessment of positive and negative evidence.
- No *a priori* probabilities.
- Classification of the diagnoses into groups according to similarities.
- Consideration of symptoms which are present but not explained.

| | Import-Value | Diagnosis-1 | Diagnosis-2 | ... |
|---|---|---|---|---|
| Symptom-1 | 2 | 0 / 5 | — | |
| Symptom-2 | 3 | 1 / 3 | 4 / 2 | |

*Frequency*
*Evoking Strength*

**Figure 8.4** Components of the diagnosis assessment in INTERNIST. Example: symptom 2 (e.g., "anemia") is relatively important (import = 3), weakly evokes (evoking strength = 1) diagnosis 1 (e.g., "occlusion of portal vein"), because it may have many other causes, but usually occurs with diagnosis 1 (frequency = 3)

The significance of a symptom in itself and in connection with a diagnosis is characterized by three parameters (see Fig. 8.4):

1. The "Evoking strength" (a number between 0 and 5, corresponding to the symptom-diagnosis probability): How strongly does the symptom evoke the diagnosis?
2. The "Frequency" (a number between 1 and 5, corresponding to the diagnosis-symptom probability): How often does a symptom occur with a diagnosis?
3. The "Import value" (a number from 0 to 5): How serious is a symptom?

The ranking of these parameters is estimated by an internist (see Fig. 8.5). and is chosen in such a way that two symptoms of a given rank have approximately

| | Evoking strength (diagnosis is cause of symptom) | Frequency (symptom occurs with diagnosis) |
|---|---|---|
| 0 | extremely rarely | — |
| 1 | rarely | rarely |
| 2 | often | often |
| 3 | mostly | mostly |
| 4 | in the majority of cases | in the majority of cases |
| 5 | practically always (pathognomonic symptom) | practically always |

**Figure 8.5** Verbal description of the categories

the same significance as one symptom of the next higher rank. The observed symptoms support a diagnosis with their *evoking strength*; symptoms which are not present, but are actually expected in connection with the diagnosis on account of their *frequency*, weaken the suspicion. Internally this is done by attributing point values to the categories of Fig. 8.5 (see Fig. 8.6). Finally it is considered how well a diagnosis can account for the symptoms which are present. The import value (scale from 0 to 5) indicates how important it is that a symptom is explained, i.e., the higher the import value, the greater is the need for an explanation. If a diagnosis cannot explain a symptom which is present but still unexplained, i.e., if the symptom is not on its symptom list, the diagnosis is given a reduction of points according to the symptom's import value (see Fig. 8.6).

| Category | Evoking strength | Frequency | Import value |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 1 | 4 | -1 | -2 |
| 2 | 10 | -4 | -6 |
| 3 | 20 | -7 | -10 |
| 4 | 40 | -15 | -20 |
| 5 | 80 | -30 | -40 |

**Figure 8.6** Point values of the categories

The final point score has no absolute significance in INTERNIST, but is simply used for comparison with the scores of other diagnoses. In contrast to the application of Bayes' Theorem, however, not all diagnoses compete with one another, but only those which can explain the same symptoms, as determined by INTERNIST on the basis of the actual symptom set. Within such a group of competing diagnoses, the best one is established when it has a sufficiently large lead (80 points) over its competitors. After a diagnosis has been established all its symptoms are marked as explained, and for the remaining symptoms other possible diagnoses are sought. In addition, established diagnoses add a bonus to diagnoses associated with them, the strength of which, like the evoking strength, is stipulated in the knowledge base.

Apart from the fact that combinations of symptoms and *a priori* probabilities for diagnoses are not taken into account, the INTERNIST model seems to be a useful compromise between simplicity, exactness and expressive strength in handling uncertain knowledge, which is confirmed above all by the existence of the large, relatively good knowledge base.

## 8.4 The MYCIN Model

Like the Dempster-Shafer Theory and INTERNIST, MYCIN [Shortliffe 75] has a separate evaluation of positive and negative evidence and also permits the representation of symptom combinations with rules. While the rules in their entirety must be mutually independent, the symptoms may correlate with one another within a rule. The reckoning scheme of MYCIN does not compare the diagnoses with one another like INTERNIST, but calculates for each diagnosis a positive and a negative "probability[2]" according to the following formula:

$$P = \begin{cases} P_{\text{old}} & \text{if} P_{\text{new}} < 0.2 \\ P_{\text{old}} + (1 - P_{\text{old}}) \cdot P_{\text{new}} & \text{otherwise} \end{cases}$$

$$\begin{aligned} P &= \text{new total probability} \\ P_{\text{old}} &= \text{old probability of the diagnosis} \\ P_{\text{new}} &= \text{rule probability} \end{aligned}$$

The rule probability is calculated by multiplying the implication probability (which comes from the expert) by the probability of the precondition (which is input by the user).

Example: Rule 535: If 1. the stain of the organism is gram-positive, and
2. the organism grows in chains, and
3. the morphology of the organism is coccus,
then the probability is 70% that the
organism is a streptococcus.

Suppose the user has input the following probabilities:

| | |
|---|---|
| Organism gram-positive: | 100% |
| Organism grows in chains: | 60% |
| Organism is coccus: | 70% |

Then the probability of the precondition is given by the minimum of the three individual probabilities (with premises related by "or" the maximum is usually taken): min(1.0, 0.6, 0.7) = 0.6.

[2]MYCIN does not calculate probabilities in the statistical sense, but a "measure of belief" and a "measure of disbelief", which are added to give a "certainty factor".

From this it follows that the rule probability is
$0.7 \cdot 0.6 = 0.42 = 42\%$.

Suppose the old probability for streptococcus, derived from other rules, is 50%, the new probability (measure of belief) is
$0.5 + (1 - 0.5) \cdot 0.42 = 0.71 = 71\%$.

If negative rules give a probability (measure of disbelief) of 30%, then the total probability (certainty factor) of streptococcus is
$0.71 - 0,3 = 0.41 = 41\%$.

A special feature of MYCIN compared to all other schemes is that the user can specify the uncertainties in assessing the symptoms in great detail. However, this gives rise to new problems which have not been satisfactorily solved:

- Different situations are treated as if they were the same: if all propositions in the precondition of a rule are only 50% certain, this is treated in exactly the same way as if one were 50% and all others 100% certain.
- For the expert the estimation of the rule probabilities is made very difficult, because he must take into account the symptom probabilities estimated by the user.

## 8.5 The MED1 Model

In MYCIN the symptoms need not be independent of one another, but the rules must. This assumption is not always appropriate, as shown by the system CASNET [Kulikowski 82] for the diagnosis of glaucoma. Since the glaucoma rules correlate strongly with each other, the probability of a diagnosis is calculated as the maximum of the rule probabilities. Hence, in order to be able to express different correlations of the rules, a two-stage reckoning scheme was implemented in MED1 [Puppe 83].

In the first stage the evidences of all positive and negative rules which have been triggered are added up. In the second stage the sum is put into one of the following seven probability classes using an interval scheme of six numbers: impossible, almost impossible, improbable, neutral, probable, highly probable, certain. The expert can define the scheme of intervals differently for each diagnosis and, through different interval sizes, simulate various forms of evidence strengthening, such as the rapid transition from low to high evidence in a small range, which is typical for subjective evaluation (see Fig. 8.7).

The schematic approximation of various types of evidence strengthening with intervals is shown in Fig. 8.8.

The two-stage reckoning scheme of MED1 is very flexible, but difficult to explain, since the evaluation of a rule is meaningful only in the context of the reckoning scheme.

| Sum of the rule probabilities | | | Probability classes |
|---|---|---|---|
| 0–50 | points | = | neutral |
| 50–60 | points | = | probable |
| 60–100 | points | = | highly probable |
| >100 | points | = | certain |

**Figure 8.7** Example of a critical region of evidence increase

## 8.6 Summary

The great variety of models for probabilistic reasoning used in expert systems indicates that no model is really satisfactory. The main problem is the uncertain starting position. Even the data collection itself can hardly be made objective in the many fields of application, since it frequently depends essentially on human senses.

The quantification of these uncertainties, which was tentatively attempted in MYCIN and is the main subject of fuzzy logic [Zadeh 75], cannot remove the arbitrariness of the input. In the MYCIN example, different users would presumably give different estimates, based on the same laboratory findings, for the probabilities that the organism grows in chains or is coccus. For this reason most expert systems allow no uncertainties in the data collection, but if there are uncertainties based on experience they are represented by qualitatively different values. This provides a simplified but clear basis for the evaluation of the data by the expert or by statistical calculations. Nevertheless, different experts or different data material lead to different estimates in the evaluation of the rules, and no reckoning scheme can compensate for this.

As a radical way out of this dilemma it has been suggested to do without the use of probabilities altogether. As long ago as 1978, Szolovits and Pauker, in an analysis of the most important expert systems for medical diagnostics, recommended "as much categorical reasoning as possible and as little probabilistic reasoning as necessary" [Szolovits 78]. Since then, the following main approaches have been made:

- Representation of the uncertainties as exceptions from the rules (see Fig. 8.9 and Chap. 9 "Non-Monotonic Reasoning").
- Replacement of heuristic rules by causal models in which the uncertainties can be replaced by more exact descriptions on various levels of detail (see Chaps. 17 and 18).

Both approaches show that much of what is subsumed under probabilistic numbers can be represented explicitly. Nevertheless it is fairly clear that one cannot completely dispense with probabilistic approaches, since it is often the case that precise knowledge is either lacking or cannot be represented in a practicable

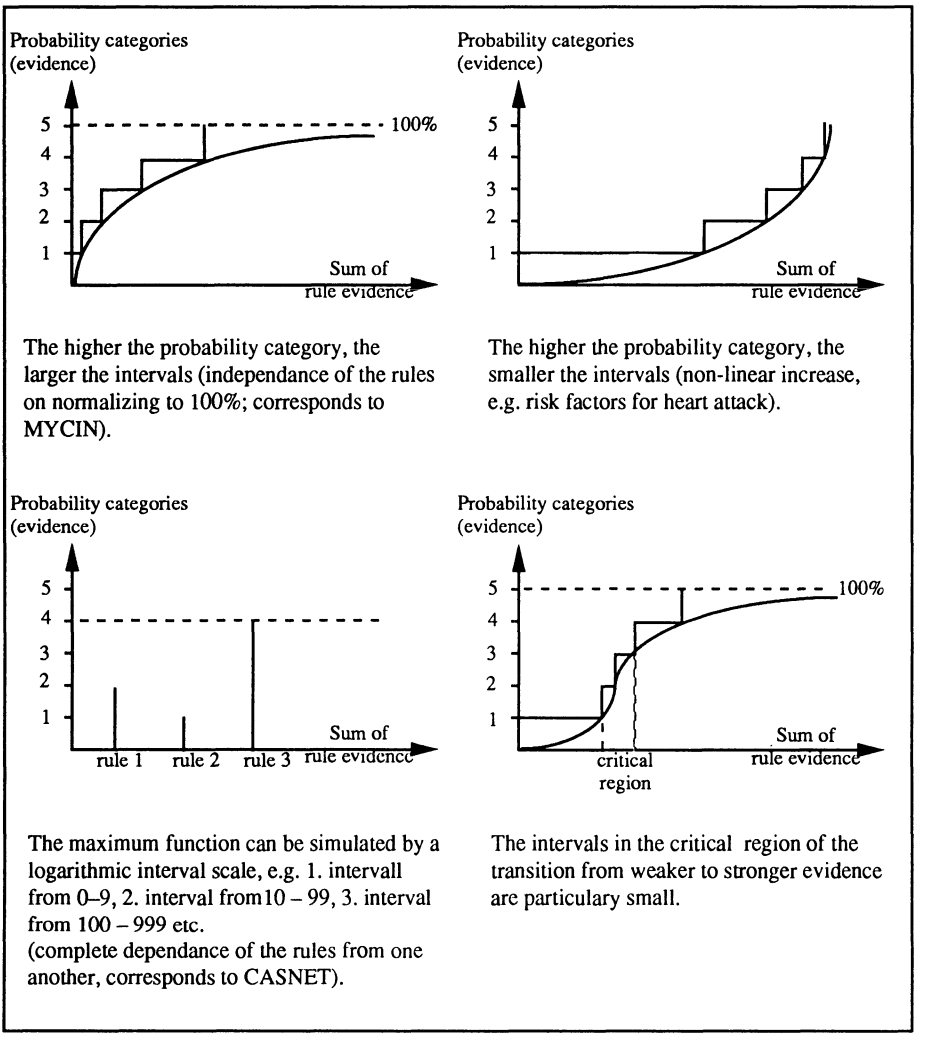



**Figure 8.8** Approximation of various evidence strengthening curves with intervals

form. A very interesting result was obtained in an experiment with the MYCIN knowledge base, in which small arbitrary manipulations (< 30%) of the rule probabilities did not change the final result [Doyle 83, p. 40]. Apparently, therefore, the exact numbers are not so important. This applies all the more, the better the structuring of the knowledge.

The consequence is that a good model of probabilistic reasoning is distinguished not so much by an apparently precise reckoning scheme, but more by the representation of the many aspects of the treatment of uncertainties. In addition to the causal models and the use of certain rules with exceptions, these also include the other aspects mentioned in the criticism of the application of Bayes' Theorem (end of Sec. 8.1). A attempt to combine many aspects of uncertain rea-

| Probabilistic reasoning | Non-monotonic reasoning |
|---|---|
| $A \Rightarrow B$ (90%)<br>The rule is true in 90% of all cases | $A \Rightarrow \mid\mid - C$<br>The rule is true, except when one of the exceptions from $C$ applies. |

**Figure 8.9** Probabilistic and non-monotonic reasoning

soning in one system is contained in the evaluating scheme of MED2 [Puppe 86]. A new, promising, well-founded approach to probabilistic reasoning is the method of causal probabilistic nets (Bayesian nets) [Pearl 88].

# 9. Non-Monotonic Reasoning

A deduction in classical logic is valid for all time, but in non-monotonic reasoning new information may mean that conclusions may have to be withdrawn (see Fig. 9.1).

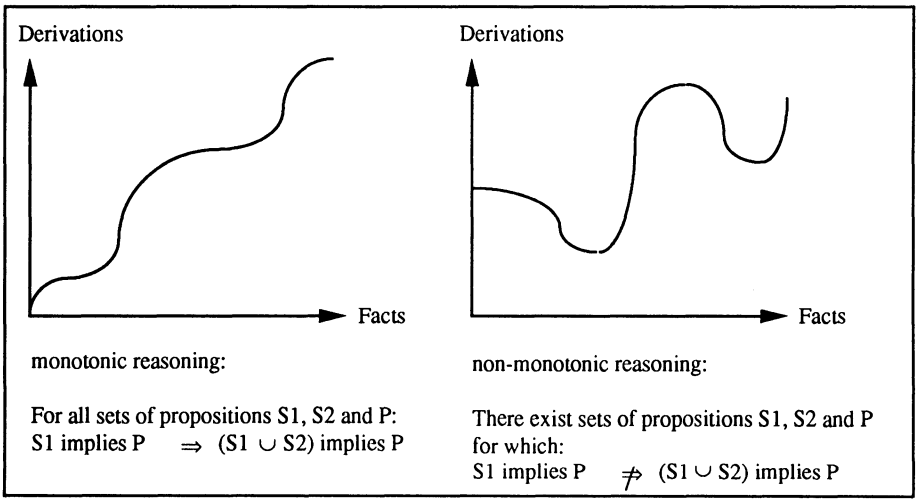



**Figure 9.1** Monotonic and non-monotonic reasoning

This at first sight unsystematic way of solving problems is the normal thing in the real world, since complete, error-free data collection is either impossible or much too complicated. Usually one has default values, or it is required to draw plausible conclusions from incomplete data and correct them as necessary, for example in the following situations:

- Default values are changed.
- Exceptions to rules are discovered.
- New evidence contradicting established conclusions becomes known.
- Input data are corrected.
- Input data change with time.

Exceptions to rules are something basically different from negated propositions in the preconditions. In the case of negated statements it must be proved that none of these statements applies before the rule fires. With exceptions, on the other hand, it must be proved that at least one exception applies in order to prevent the action of the rule. This means that lack of knowledge about a particular proposition must be counted *for* the rule in the case of exceptions and *against* the rule in the case of negated statements. Exceptions are based on a logic with at least three values, i.e., a statement can be true, false or unknown. However, non-monotonic reasoning must also be used for two-valued logic, if the fact that one cannot prove a theorem leads to the conclusion that the theorem is not valid ("negation as failure", "closed world assumption", example: PROLOG). Thus under the closed world assumption the value "unknown" is represented as "false" and may lead to a revision, whereas "unknown" does not exist in classical logic, so that its derivations always remain valid unless the axioms are changed.

The formalisation of non-monotonic reasoning is not only a difficult theoretical problem – various calculi have been developed but they are generally semi-decidable and of no use for implementation – but also a practical problem in the development of expert systems. Only the second aspect is discussed below. Good reviews of theories of non-monotonic reasoning are given, e.g., in [Brewka 89] and [Genesereth 87, Chap. 6].

When revising a conclusion ("belief revision"), it must be remembered that other conclusions may depend upon it, and the revision of one element in this network may set off a chain reaction. It is the task of belief revision systems or truth maintenance systems[1] in such cases to establish the state which would have occurred if the changed or new fact had been taken into account from the start.

The simplest algorithm is the *recalculation* of all conclusions from the changed input data. A simple improvement is to keep a record of all conclusions and to recalculate from the point where the changed fact or the revised conclusion was mentioned for the first time (*backtracking*). The effort required for both of these "brute force " methods increases rapidly of course with the amount of data to be managed and is acceptable only in small expert systems. It is considerably more economical to avoid the recalculation of all or a great part of a session and correct only those conclusions which are actually affected by the change. For this purpose one must store the justifications for each conclusion and, when a fact is revised, check recursively which justifications are no longer valid.

The two most important approaches differ in what is stored as the justifications for a conclusion:

- Direct justifications (JTMS, e.g., TMS [Doyle 79]).
- Basic assumptions on which the direct justifications are based (ATMS, e.g., ATMS [de Kleer 84]).

[1]To the terminology: TMS (Truth maintenance system) – or alternatively RMS (reason maintenance system) – is the generic term for belief revision systems. The two main types are JTMS (justification-based TMS) and ATMS(assumption-based TMS). The TMS due to Doyle and de Kleer's ATMS are examples of concrete systems, which unfortunately bear the same names as the generic terms.

The basic ideas of JTMS and ATMS may be illustrated by the following example, based on a two-valued logic without exceptions but with the closed world assumption. The rules $R1$: $A \rightarrow B$, $R2$: $B \rightarrow C$ and $R3$: $D\&\neg E \rightarrow C$ are given.

In JTMS a justification is valid if all the positive (monotonic) preconditions are valid and all the negative (non-monotonic) preconditions are not valid (see Fig. 9.2).

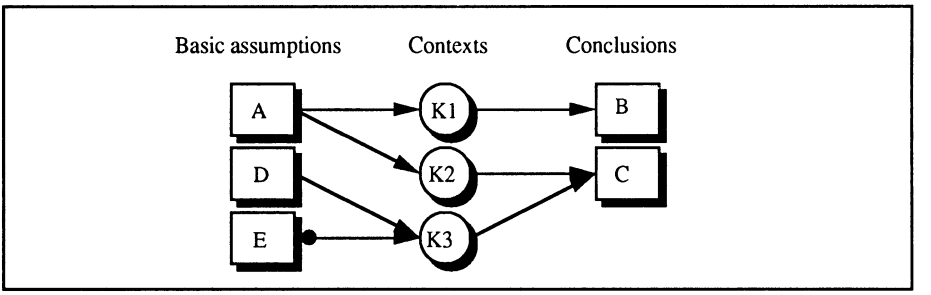



**Figure 9.2** Interdependence network in JTMS (a negation is indicated by a point on the arrow)

In ATMS $B$ is valid in the context $K1 = \{A\}$ and $C$ in the contexts $K2 = \{A\}$ and $K3 = \{D, \neg E\}$ (see Fig. 9.3). The difference between JTMS and ATMS has in this case the effect that in ATMS the basic assumption $\{A\}$ supplies a justification context for $\{C\}$, whereas in JTMS this role is played by the rule $R2$.

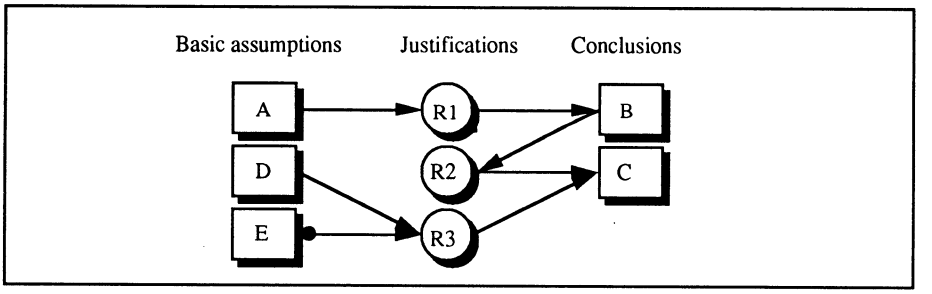



**Figure 9.3** Interdependence network in ATMS. The context is a minimal set of basic assumptions under which a conclusion is valid.

## 9.1 JTMS with Direct Justifications

Figure 9.4 shows the basic algorithm for the revision of conclusions in an interdependence network with direct justification.

The main problem with the JMTS approach is the treatment of circular justifications (e.g., $S1 \rightarrow S2$ and $S2 \rightarrow S1$ in Fig. 9.5), which prevent the withdrawal of a conclusion in the loop.

Input: Change of a fact.
Output: Propagation of the change with establishment of a consistent state.

1. If an input datum or a conclusion has changed, check all justifications connected with it.
2. If a justification becomes invalid, check whether the conclusion has other justifications.
3. If a conclusion no longer has any valid justifications, withdraw it and call the algorithm recursively with the withdrawn conclusion. Otherwise no change is necessary.

**Figure 9.4** Basic algorithm of JMTS

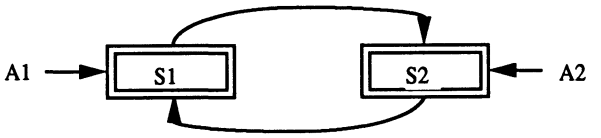



**Figure 9.5** Monotonic loop: Once $S1$ and $S2$ have been established, they can no longer be withdrawn by the basic JTMS algorithm (Fig. 9.4). For the sake of clarity the justifications are not shown as in Fig. 9.2

If the loop contains no negations, as in Fig. 9.5, it is called *monotonic*, otherwise *non-monotonic*. A non-monotonic loop is called *even* if it contains an even number of negations, otherwise *odd* (see Fig. 9.6).

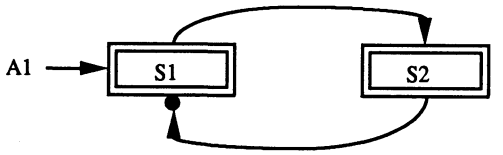



**Figure 9.6** An odd, non-monotonic loop with the rules $A1 \rightarrow S1$, $S1 \rightarrow S2$ and $S2 \rightarrow \neg S1$ (the negation is indicated by a point on the arrow). If $S1$ is established, the JTMS algorithm leads to an endless loop, because $S1$ and $S2$ are alternately established and withdrawn

In order to deal with such circular problems, the basic algorithm of Fig. 9.4 must be modified, which makes it considerably less efficient or restricts its generality. It should not therefore be modified more than required by the type of loops expected in the intended domain. Since odd non-monotonic loops are semantically meaningless, they are not considered in most belief revision systems. If they are present, the system runs into an endless loop. This can be avoided if odd non-monotonic loops are recognized either during the construction of the knowledge base as in ITMS (see point 3 below) or at run time as in Goodwin's [82] algorithm.

Monotonic loops, on the other hand, are useful in many fields of application, since they can be used to avoid a rigid direction of deduction. Take for example the reciprocal correlation between liver cirrhosis and high blood pressure in the portal vein: if there is evidence for one of these conditions, then the other is also likely. Extensions of the algorithm shown in Fig. 9.4 for the treatment of monotonic loops are based on the distinction between non-circular ("well-founded") and circular justifications. The most important types are:

**1. The "Current Support" Strategy:** In Doyle's [79] TMS and other JTMS systems [McAllester 80, Goodwin 82], the first justification derived for a conclusion is singled out as its current support, because it cannot be circular. As long as a conclusion retains its current support when a revision takes place, its validity is beyond question. If not, however, a complex procedure must be carried out to determine its validity, because one can no longer rely on the remaining justifications, which may be circular. The conclusion is therefore temporarily withdrawn, and this change is propagated through the interdependence network. In this process conclusions loose their circular justifications, as can be seen from the example shown in Fig. 9.7. If, after completion of this procedure, a temporarily withdrawn conclusion still has well-founded justifications, it is re-established, a new current support is set and the renewed establishment is propagated. Conclusions finally left over without justifications are withdrawn permanently.

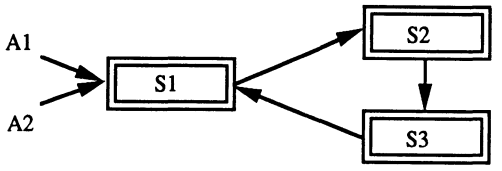



**Figure 9.7** Example illustrating the different JTMS techniques for the treatment of loops

The disadvantage of the current support strategy is that many conclusions may be withdrawn and re-established more often than is necessary. Referring to Fig. 9.7, if $S1$ has $A1$ as current support and $A1$ becomes invalid, it is no longer possible to distinguish between the circular justification from $S3$ and the well-founded justification from $A2$. $S1$ – and as a consequence $S2$ and $S3$ – are therefore withdrawn temporarily. Then it is found that S1 still has a justification from $A2$, which is designated the new current support. Finally, $S2$ and $S3$ regain their support and are re-established.

**2. Indication of Circular Structures by the Programmer:** For dynamic storage management systems such the LISP garbage collector it is a very efficient strategy to count the number of references to an object and to de-allocate (erase) any object for which the number sinks to zero. The problem is again how to deal with circular data structures, which can never be de-allocated with this strategy.

A solution suggested by Bobrow [80] is for the programmer to define circular structures, so that the references to the structure as a whole can be counted. As soon as there are no more external references to a circular structure it can be de-allocated.

Example: In Fig. 9.7, $S1$, $S2$ and $S3$ must be defined as a circular structure. The justifications from $A1$ and $A2$ are external references to this structure. If both justifications are withdrawn, the circular structure is de-allocated.

This approach has the advantage of being efficient, but requires, in the form suggested by Bobrow, additional work on the part of the programmer, since he or she must define the circular structures. An improvement would be to provide for automatic pre-detection of loops as in ITMS (see next point). Another difficult problem is the treatment of nested circular structures.

**3. Automatic Pre-Detection of Loops:** If the knowledge base permits pre-detection of loops, i.e., if the rules used for loops do not contain any variables, the system can deal with the loops automatically without the help of the programmer. An efficient method for a rule-based system is ITMS (immediate check TMS, [Puppe 87]). In this system all loops are registered during the construction of the knowledge base by marking all rules which can supply circular evidence. As soon as a conclusion is established, its circular rules are blocked, provided they have not yet fired. Those justifications which are not blocked are therefore always well-founded. Since the number of justifications is counted in ITMS, it uses more information than the current support system and therefore works more efficiently.

Example: In Fig. 9.7 the rule $S3 \rightarrow S1$ for $S1$, the rule $S1 \rightarrow S2$ for $S2$ and the rule $S2 \rightarrow S3$ for $S3$ were given the attribute "circular rules" during construction of the knowledge base. When $S1$ is established via $A1$ and $A2$, the rule $S3 \rightarrow S1$ is blocked, so that only the well-founded justifications from $A1$ and $A2$ are stored under $S1$. If A1 is withdrawn, it is therefore recognized immediately that $S1$ still has a second justification.

A particular advantage of ITMS is that, unlike the other techniques discussed in this chapter, it can easily be adapted to probabilistic reasoning, since probabilities can be added instead of categorical justifications (see [Puppe 87]). The disadvantage of ITMS is its limitation to loops whose rules do not contain reference variables, since the pre-detection is otherwise impossible in most cases.

## 9.2 ATMS with Basic Assumptions as Justifications

The approach in which the original basic assumptions are used instead of direct justifications for a conclusion has quite different properties. The main effort required is for handling the interdependence network, while the actual withdrawal of a fact can be carried out very efficiently.

The primary data structure of ATMS is a node consisting of three components:

- the datum which it represents (a proposition concerning the domain),

- the set of basic assumptions under which the node is valid,
- the set of its direct justifications (which serve above all for documentation).

In Fig. 9.3 five nodes $X1$–$X5$ would be generated for the propositions $A$, $B$, $C$, $D$, $\neg E$ together with the rules $A \rightarrow B$, $B \rightarrow C$, $D\&E \rightarrow C$, with the following components:

| | | Datum | Context | Direct Justification | |
|---|---|---|---|---|---|
| $X1$ | = [ | $A$, | $\{\{A\}\}$ | $\{(A)\}$ | ] |
| $X2$ | = [ | $B$, | $\{\{A\}\}$ | $\{(A \rightarrow B)\}$ | ] |
| $X3$ | = [ | $C$, | $\{\{A\},\{D,\neg E\}\}$ | $\{(B \rightarrow C),(D\&\neg E \rightarrow C)\}$ | ] |
| $X4$ | = [ | $D$, | $\{\{D\}\}$ | $\{(D)\}$ | ] |
| $X5$ | = [ | $\neg E$ | $\{\{\neg E\}\}$ | $\{(\neg E)\}$ | ] |

In ATMS a node contains no information about its validity, which can only be determined by comparing its basic assumptions with the global context. Thus the nodes $X1$–$X4$ are valid in the global context $\{A, D\}$. If $A$ is withdrawn, only $X4$ remains valid, which can be derived without manipulation of the interdependence network.

In order to minimize the amount of contexts for a node, redundancies which might occur are removed when a new justification is added. If the rule $D \rightarrow C$ is added, for example, then the new context $\{D\}$ replaces the old context $\{D, \neg E\}$, and the node $X3$ now has the following contents: $[C, \{\{A\}\{D\}\}, \{(B \rightarrow C)(D\&\neg E \rightarrow C)(D \rightarrow C)\}]$.

A special feature of ATMS is the treatment of inconsistent rules, e.g., the additional rule $F \rightarrow \neg C$. Instead of rejecting the rule, ATMS simply takes note of all sets of inconsistent basic assumptions, which are handled by means of so-called no-good nodes, e.g., no-good = $[\{A, F\}, \{D, \neg E, F\}]$. All global contexts which contain a no-good set as a subset are inconsistent.

The loop problem of the JTMS approaches is dealt with automatically in ATMS, since the loops are broken up on reduction to the basic assumptions. Thus in Fig. 9.6 the context for $S1$ would simply be the basic assumptions $\{\{A1\}\{A2\}\}$ (likewise for $S2$ and $S3$).

The main operations of ATMS are comparisons of sets, and this is the critical issue with regard to its efficiency. A possible implementation of the set operations is the representation of the basic assumptions as bit vectors, which can then be related by "and" or "or". However, the efficiency decreases as the number of basic assumptions increases. De Kleer states that the algorithm is still practicable with up to 1000 basic assumptions [de Kleer 86, P. 152].

The disadvantages of ATMS include the treatment of uncertain knowledge and rules with exceptions. The problem is that in these cases it is not so easy to trace a conclusion back to the basic assumptions as it was in the examples considered so far. Further, the more basic assumptions there are, i.e., the more data is required for the problem description in the domain, the more complicated it becomes to trace all conclusions back to the basic assumptions.

## 9.3 Summary

A summary of the relative advantages and weaknesses of both approaches is given in Fig. 9.8. The main advantage of ATMS [de Kleer 84, 86] is that it is

| | JTMS | ATMS |
|---|---|---|
| Type of justification | direct justification | basic assumptions |
| Treatment of loops | complicated | simple |
| Treatment of exceptions from rules | simple | complicated |
| Treatment of uncertainties | in special cases simple | complicated |
| Comparison of different solutions | complicated | simple |
| Efficiency | depends upon degree of interdependence of knowledge base | depends on amount of basic assumptions |

**Figure 9.8** Comparison between JTMS and ATMS

possible to follow different solutions in different contexts independently of one another. This is useful for example in qualitative simulation (see Chap. 32), for which de Kleer actually developed ATMS. On the other hand, if one is interested in only one solution JTMS [Doyle 79, McAllester 80, Goodwin 82] should be more efficient, since ATMS must resolve intermediate solutions into their basic assumptions during problem solving. However, one must not overlook the fact that the general implementation of both approaches is fairly inefficient for large knowledge bases with a high degree of interdependence. An acceptable efficiency is seemingly attainable at present only for special cases, e.g., if no loops, or only those which can be pre-detected, are allowed (ITMS [Puppe 87]), or if the knowledge base is not too large.

# 10. Temporal Reasoning

The representation of time-dependent information has been practically neglected so far in expert systems, because the knowledge representation and the derivation strategies can become considerably more complicated. On the other hand there are many fields of application where temporal information is important. In diagnostics, for example, the change of a symptom with time often conveys more than the actual value, and in planning temporal boundary conditions must be fulfilled. At present the most promising approach to temporal reasoning in expert systems is to limit the representation to the most important temporal aspects of the domain.

There is a basic difference between systems which draw conclusions from time-dependent data and those which derive hypothetical future situations. The former can be regarded as temporal data bases with a suitable enquiry language, while the latter are simulation systems. In this chapter we will restrict ourselves to the representation and management of time-dependent data. Simulation is treated in Part IV.

In probabilistic reasoning statements are given a degree of uncertainty, and in non-monotonic reasoning derivations are associated with justifications. In an analogous way facts are qualified with a time value in temporal reasoning, e.g., "onset of thorax pains: three weeks ago". There are numerous aspects to be considered for the exact representation, amongst which the following may be named:

- *Base representation.* There are two types, one based on points (instants) and the other on time intervals. Both representations are equally powerful in principle, because an interval can be represented by a pair of points, and an point by an arbitrarily small interval. Nevertheless, the simplicity and elegance of application can be very different for different domains.

- *Exactness.* The representation of an exact time is simplest. In many domains, however, exact time information is not available. The inexactness may be specified qualitatively (e.g., the thorax pains began before the stay in hospital) or quantitatively (e.g., the thorax pains began three to four weeks before the stay in hospital).

- *Reference.* Three ways of relating one time specification to another may be distinguished: an absolute time scale and single or multiple reference

events. The simplest method is the use of an absolute time scale (e.g., for an exact time: the thorax pains began at 3.30 p.m. on 9-3-1988). The alternative is to stipulate reference events (e.g., three weeks before the stay in hospital). Reference events have the advantage that, unlike the absolute time scale, they do not require that all time specifications be completely ordered. If an unprecise time specification is described in relation to several reference events, the intersection of the various uncertainty intervals may be more precise than each individual relationship. However, the calculation of multiple reference events and the maintenance of consistency complicate the time representation considerably compared to using single reference events. A suitable implementation technique is the representation of time relations as constraints and the calculation of the most precise relationship between two events as constraint propagation. Since contradictions may occur when an uncertainty interval is restricted by further information, the use of a TMS may be necessary.

- *Units of time.* A representation with quantitative time specifications appear much more natural if not only the numbers but also the units (e.g., seconds, minutes, years) are also represented. Further, the choice of unit may also be used as a means of conveying inexactness (e.g., "a year ago" and "365 days ago" do not always mean the same).

The complexity of time representation determines of course how efficiently and with how much effort questions to time data bases can be answered. Typical questions are for example:

- Is a fact valid during a certain interval?
- Has a value or its rate of increase changed recently?
- What is the temporal relationship between two facts?

The various aspects of temporal reasoning are illustrated in the following sections by means of the diagnostics expert systems VM and MED2, which require the time representation for the evaluation of subsequent sessions, and also by means of the general time representation formalisms of McDermott and Allen.

VM and MED2 have a point-oriented basis representation and process mainly exact time data. VM uses an absolute time scale without units, whereas in MED2 times are specified by reference to other times, either as exact time specifications with units or as qualitative before/after relationships.

In the more powerful formalisms of Allen and McDermott inexact time specifications with multiple reference events are allowed, so that, as already indicated, they perform a kind of constraint propagation. One technique for increasing efficiency is the partitioning the network of time relationships into groups ordered in a hierarchy (clusters, see Sect. 10.3). The systems of McDermott and Allen differ in the base of the respective time representations (point or interval, respectively) and in the degree of inexactness (inexact quantitative or qualitative relationships, respectively).

## 10.1 Exact Time Relations: VM and MED2

One of the first time-processing expert systems was VM [Fagan 84], whose knowledge representation is based on MYCIN and which was developed for the surveillance of patients in the iron lung. VM controls the transitions between different types of respiration assistance according to the patient's condition as given by about 30 measured data, which must be stored together with the exact time of measurement. Since the measurements are made at regular intervals, no time units are needed; simple numbers on an absolute time scale are sufficient (see Fig. 10.1).

| Time elapsed (in minutes) | 69 | 59 | 58 | ... | 9 |
|---|---|---|---|---|---|
| Frequency of respiration (per minute) | 9 | 9 | 10 | ... | 9 |
| Blood pressure (in mm Hg) | 141 | 154 | 153 | ... | 150 |
| : | : | : | : | : | : |
| Time of day | 12.30 | 12.20 | 12.19 | ... | 11.30 |

**Figure 10.1** Time representation in VM

For the evaluation of time successions VM has, in addition to the "usual" rule predicates, special temporal predicates and functions. The most important types are:

- Fluctuation: This predicate is for monitoring changes of parameters, i.e., whether a measured value has changed more than a certain amount or not during a certain time interval. Thus the statement (fluctuation, blood pressure, change-in-both-directions, 15, 20, negation) in the precondition of a rule is expanded to true if the blood pressure has not changed by more than 15 mm Hg during the last 20 minutes.

- Time-expect: This predicate checks whether a parameter has a certain value during a given time interval.

- Expect: This function sets the tolerable or optimum range of a parameter according to the global condition of the patient as a reference for other rules.

A subsequent session with a new series of measurements is regarded as a new session in VM and evaluated completely from the start. Since there are only 30 parameters it is possible to do without a TMS.

MED2 [Puppe 87] is based on a time representation similar to that of VM. The following discussion centers on the differences between the two systems. In MED2 it is not required that the date and time are known for each symptom, but the user specifies times relative to a reference point, which may be either the pre-defined reference point "now" or one defined by the user. In subsequent sessions, the old "now" is automatically renamed "session $X$" and related to the

new "now". The value of a symptom remains valid until it is explicitly overwritten by the user, so that only the changed symptoms need to be entered in subsequent sessions. MED2 also allows unprecise time specifications such as "event 1 preceded event 2", but since any point is represented by a relationship to exactly one other point, this representation only requires a connection of before/after chains and no real constraint propagation. Time specifications are evaluated by MED2 with the following predicates and corresponding functions (see Fig. 10.2):

- Calculation of the temporal relationship between two points (predicates: before, after, simultaneous; with limited time interval if required).
- Calculation to determine whether the absolute value of a parameter has changed, which corresponds in most respects to the "fluctuation" of VM. To test the influence of a therapy on the system, MED2 can also specify an event which must have taken place during the change (e.g., did the temperature drop by at least one degree *after the medicine was administered*?).
- Calculation of the change in the rate of change of a parameter (e.g., test whether the difference between one measurement and the next has changed significantly).

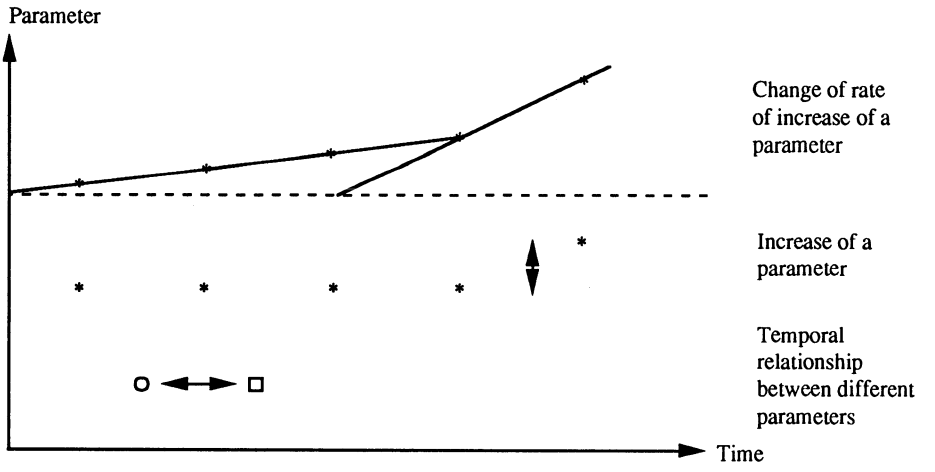



**Figure 10.2** Temporal predicates in MED2

In MED2 a subsequent session is treated not by a complete new calculation as in VM, but by specific correction of the conclusions on the basis of the old session and the ITMS (see Sect. 9.1).

## 10.2 Inexact Quantitative Time Relations: TMM of McDermott

A powerful systems for treating inexact reference events and multiple time relationships is the TMM (**T**ime **M**ap **M**anager, [Dean 87]) of Dean and McDermott.

Its function is the management of a temporal data base which may be accessed by other programs. The uncertainty in the time of an event is expressed as an interval in TMM. Units of time are not represented explicitly. An example for the representation of times in TMM is (elt(distance $T1$ $T2$) 0 10), which means that $T1$ precedes $T2$ by between 0 and 10 time units (elt = "element of").

The duration of the validity of a fact is represented by the two points "start" and "finish". The assertion that a fact is valid during a particular interval can be specified as follows:

1. (assert fact interval):
   This attaches a variable "interval" to the fact, and this variable is used in the following two statements.

2. (elt(distance(start interval)"reference point") number number)
   The start of the interval is related to another point. Any number of such relationships may be given.

3. (elt(distance(start interval)(finish interval)) number number)
   The duration of the interval is specified.

This formalism still does not enable a very vague statement such as "$T1$ precedes $T2$" to be represented correctly. For this purpose the symbols "pos-tiny" (a number greater than 0, but less than any positive number) and "pos-infinite" (a number greater than any other number) are allowed in addition to real numbers for specifying the limits of an interval. Thus (elt(distance $T1$ $T2$) pos-tiny pos-infinite), for example, means that $T1$ precedes $T2$.

In order to answer the question whether an assertion is valid during a certain interval, the TMM evaluates all temporal relationships which refer to the beginning or the end of the assertion. If only the beginning is known, it is assumed that the assertion remains valid indefinitely, i.e., the default value "pos-infinite" is used for the end. Since additional information may limit the duration of an assertion, conclusions may become invalid. The TMM uses the procedure (see Sect. 9.2) for the withdrawal of a conclusion.

A large time data base involves so many temporal relationships that it is not possible to evaluate them all as precisely as possible all the time (i.e., carry out a complete propagation of all temporal constraints). In the TMM this problem is solved by having the user specify which relationships are of interest and monitoring only these continuously. Another solution, which is explained at the end of the next section, is to organize the time intervals in a hierarchy.

## 10.3 Qualitative Relations: the Time Calculus of Allen

Like the TMM, Allen's [83] temporal calculus also serves for handling a temporal data base and provide other programs with an enquiry language. As the basic representation Allen's temporal calculus uses intervals, which may be related to

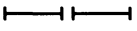
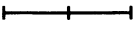
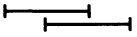
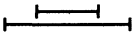
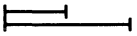
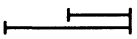
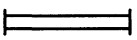
| Relation | | | Symbol | Inverse Symbol | Graphical Example |
|---|---|---|---|---|---|
| $X$ | *before* | $Y$ | < | > |  |
| $X$ | *meets* | $Y$ | m | mi |  |
| $X$ | *overlaps* | $Y$ | o | oi |  |
| $X$ | *during* | $Y$ | d | di |  |
| $X$ | *starts* | $Y$ | s | si |  |
| $X$ | *finishes* | $Y$ | f | fi |  |
| $X$ | *equals* | $Y$ | = | = |  |

**Figure 10.3** Allen's 13 relationships

other intervals by means of qualitative statements. The 13 disjunctive qualitative relationships between two intervals are shown in Fig. 10.3.

These operators are sufficient for expressing many temporal relationships very elegantly, for example Allen can express the statement "$B$ is valid during $A$" simply by "$B$...(during)...$A$." In the TMM the following statements are required:

1. (elt(distance(start $A$)(start $B$)) pos-tiny pos-infinite): "$A$ starts before $B$,"
2. (elt(distance(finish $B$)(finish $A$)) pos-tiny pos-infinite): "$B$ ends before $A$," and the implicit conditions
3. (elt(distance(start $A$)(finish $A$)) pos-tiny pos-infinite): "$A$ is an interval,"
4. (elt(distance(start $B$)(finish $B$)) pos-tiny pos-infinite): "$B$ is an interval."

Two intervals may also be connected by a group of basic relationships, e.g., "$A$ starts before $B$" is represented by $A$...(< m o di fi)...$B$, i.e., one of the relationships "before", "meets", "overlaps", "during-inverse" or "finishes-inverse" applies.

In Allen's temporal calculus as in McDermott's TMM the determination of the exact temporal relationship between two intervals is often complicated, because the chains of reference events can be long and may also show complex branching on account of the possibility of multiple reference events for an interval.

A simpler calculation is achieved by the method of cluster formation given by [Allen 83]. A cluster is a set of intervals which belong together. Within a cluster the reciprocal relationships between all intervals are calculated completely. In each cluster one interval[1] is distinguished by having relationships also to other

[1]In the following discussion the cluster is always named after this special interval.

clusters. The relationship between two intervals $A1$ and $B1$ from two different clusters $A$ and $B$ is not stored explicitly, but calculated in three steps: from A1 to the special interval $A$, from there to the special interval $B$ of the other cluster and from $B$ to $B1$. In large temporal data bases the clusters themselves can also be arranged in a hierarchy.

An example of cluster formation is shown in Fig. 10.4. The derivation of the temporal relationship between pre-school and primary school runs as follows: since pre-school and primary school belong to different clusters, we first find a path between the clusters (pre-school... meets... education) and then the relationship within the cluster (primary school... starts... education), which are combined to give (pre-school... meets... primary school).

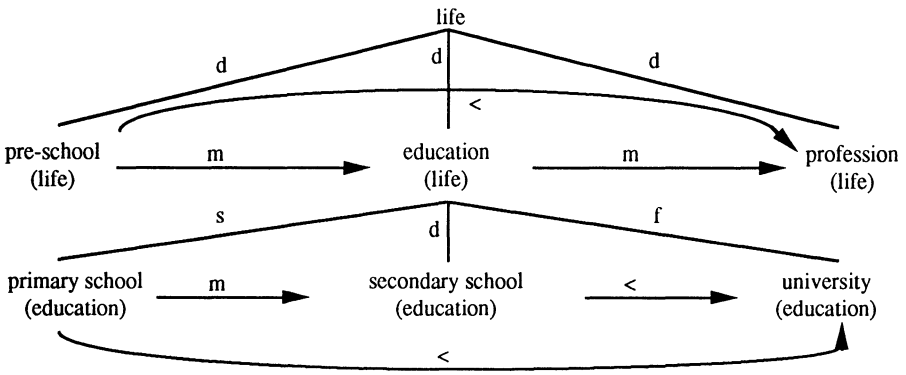



**Figure 10.4** The formation of clusters of time intervals

Allen's temporal calculus can be efficiently implemented [Vilain 82] and is sufficient for many applications in which quantitative temporal data are lacking, such as in qualitative simulation (see Chap. 32).

## 10.4 Summary

There are two types of system which handle temporal information: temporal data bases for the management and recovery of time-referenced data and simulation systems (see Part VI) for the prediction or explanation of events.

The most important characteristic of temporal data bases is the allowed degree of inexactness of temporal relationships. While simple, exact systems have already been realized in some expert systems (e.g., VM [Fagan 84], MED2 [Puppe 87]), the management of multiple, inexact temporal data is considerably more complicated, since conceivable reduction of the uncertainties by additional information must also be handled. For this purpose the techniques of constraint propagation are suitable. Two well-known approaches are TMM [Dean 87] and Allen's [83] temporal calculus. The latter has intervals as the smallest unit of representation, which simplifies the knowledge representation considerably compared to the point-based representation of the TMM, but is poorly suited for the

representation of inexact, quantitative temporal data. Temporal systems must also be capable of revising conclusions, for which a TMS is required, except in small domains as in the case of VM, for example. A good review of temporal reasoning can also be found in [Charniak 85, Sect. 7.4].

# Part III

# Problem Classes and Problem-Solving Methods

# 11. Previous Approaches to Problem Classification

Despite the great theoretical and practical importance which must be attached to a division of application domains into problem classes, this subject has been rather neglected in the past. In many introductory books, such as [Jackson 86][1], problem classes and problem-specific methods of solution are not treated. Even in the latest volume of the "Handbook of Artificial Intelligence" (1989), only basic techniques are described in the chapter on expert systems, and on the subject of problem classes it is stated:

> "We lack a robust taxonomy of problem types (among the most complete so far is Chandrasekaran 1986)[2], so the individual examples still provide a better characterization of the types of problem than general descriptions." [Buchanan 89, p. 183f].

In the 1970s, rule-based programming was regarded as the typical problem-solving method of expert systems, and early in the 1980s this was supplemented by approaches using objects and constraints (see Chaps. 5–7). A comprehensive, detailed description of these and other problem solving methods in artificial intelligence can be found in [Stoyan 91]. For many general problem-solving methods corresponding programming languages were developed, which may be classified under "general tools" in Fig. 2.1. Despite their usefulness compared to the conventional universal programming languages, it turned out that they can offer only limited assistance in the development of expert systems and in this connection contribute little to a differentiation required for the classification of problem types.

The difficulties encountered when trying to develop a good classification of problems arise from the fact that there are two quite different requirements to be fulfilled:

1. Users must easily be able to identify the classes of their problems.

2. For the problem classes it should be possible to make statements about how complex the corresponding expert systems are expected to be and to provide

[1] However, in the second edition (1990) a distinction is made between classification and construction.

[2] See Sect. 11.4.

assistance for their development. This requires the specification of suitable problem-solving methods, which should be implemented by corresponding problem-specific tools as a comprehensive aid for knowledge acquisition.

Since it is very difficult to satisfy both requirements, separate classifications are often developed and the relationships between them given. In order to distinguish between the two types of problem classes, we will speak of *problem types* in the case of user-specific classification with no recognisable connection with problem-solving methods, while the categories which are defined according to the criteria of the applicability of powerful problem solving methods will be called *problem-solving types.* When classifying into problem-solving types, negative statements are also meaningful, since there are problem categories for which no good problem-solving methods exist at present. However, such a statement is often given implicitly, when no suitable problem-solving method is known for a particular application.

A review of previous approaches for the division of domains into problem classes is given below. While the first approaches represent only the classification of problem types, increasing attention has been given more recently to the identification of problem-solving types. As the first problem-solving type Clancey [85] described heuristic classification (see Sect. 11.2). Generally, however, the classification according to problem-solving types is still very fragmentary, especially if the requirement of operationalization in tools is taken into account.

## 11.1 The Approach of Stefik et al. and Hayes-Roth et al.

The still most frequently cited example of a classification of problem types from the point of view of the user is the scheme given in the first general book on expert systems "Building Expert Systems" [Hayes-Roth 83, p. 14 and p. 82ff], a part of which (p. 82ff) had been published previously in [Stefik 82]. In [Hayes-Roth 83, p. 14] the "generic categories" of problem types shown in Fig. 11.1 are described. In [Stefik 82] the last four categories are not listed, but the others are

**Interpretation:** Inferring situation descriptions from sensor data
**Monitoring:** Comparison of observations to plan vulnerabilities
**Diagnosis:** Inferring system malfunctions from observables
**Prediction:** Inferring likely consequences of given situations
**Design:** Configuring objects under constraints
**Planning:** *Designing* actions
**Debugging:** Prescribing remedies for system malfunctions
**Repair:** Executing a plan to administer a prescribed remedy
**Instruction:** *Diagnosing, debugging* and *repairing* student behavior
**Control:** *Interpreting, predicting, repairing* and *monitoring* system behaviors

**Figure 11.1** Problem types according to Hayes-Roth [83. p. 14]

described in more detail by means of exemplary systems and typical problems: DENDRAL for the interpretation of mass spectrograms, VM for the surveillance of patients in the iron lung, MYCIN for the diagnosis of bacterial infection diseases, MOLGEN for planning experiments in genetics; there are no examples for design or prediction. The main part of the article is concerned with the discussion of techniques for the solution of "frequently recurring problems" of various categories, namely searching methods for large solution spaces and the treatment of plausible assumptions, time-dependent data and uncertain data. The techniques are illustrated by means of example systems, which are given in brackets in the following list:

uncertain reasoning (MYCIN, PROSPECTOR),
temporal reasoning with situation calculus (VM),
generate-and-test procedures (DENDRAL, GA1),
partitioning into subtasks (R1),
hierarchical reasoning with top-down refinement (ABSTRIPS),
least-commitment strategy (NOAH, Stefik's MOLGEN),
constraint propagation (Stefik's MOLGEN),
plausible reasoning with belief revision (EL),
pursuing several solution paths (HEARSAY II, SYN),
blackboard model (HEARSAY II),
knowledge compiling (RETE algorithm in OPS5, EMYCIN, HARPY).

This is just a simple list of techniques, used in expert systems, lying on widely different levels of abstraction. There is no classification from the user's point of view according to the problem types described above. The authors regard their approach as a basis for discussion, as they indicate at the end of the article:

> "The theory of building intelligent systems is far from complete, and the ideas expressed here are by no means universally accepted in the AI community. To wait for the ideas to settle and survive the test of history would preclude making a timely guide to current thinking."[Stefik 82, p. 171]

## 11.2 The Approach of Clancey

Clancey [85] developed the problem type classification of Stefik and Hayes-Roth further into a hierarchical scheme and distinguished between generic and compound problem types. Further, he gave the first detailed description of a problem-solving type, namely heuristic classification. We first give a general review of his classification of generic problem types (Fig. 11.2). Clancey emphasizes the distinction between problems of analysis (interpretation of a system) and synthesis (construction or modification of a system). In characterizing problem types he starts from the general idea of a system as something which produces an output from an input. The interpretation problem types are subdivided according to

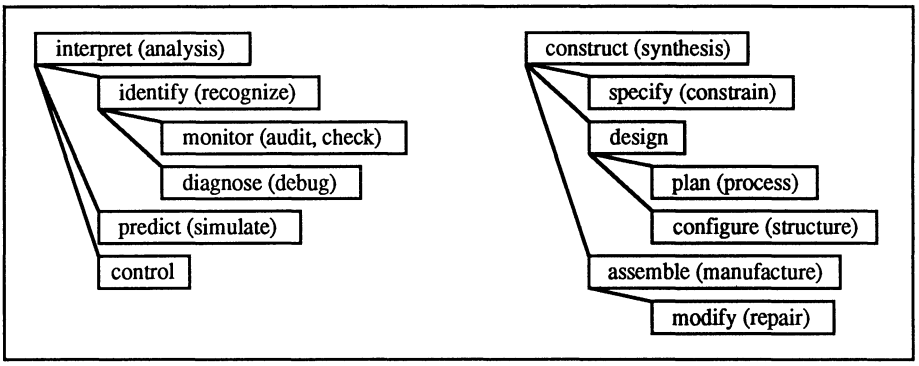



**Figure 11.2** Problem types according to Clancey [85]

which of the three factors input, system or output is deduced from the other two. In *identification* the (defective) system state is deduced from the output and the input; in *prediction* the output is derived from the input and the system description; and in *control* the input is deduced from the system description and the (required) output. The two subtypes of identification differ in that monitoring is for detecting system defects, whereas diagnosis is for tracing system errors to defective system components (or external conditions).

The subtypes of construction are specification, i.e., refinement of a system description, design and assembly, in which a system is physically changed, e.g., by carrying out repair instructions. The two types of design – configuration and planning – differ according to Clancey not so much in their internal structures as in their viewpoints: One can configure a travel plan to fulfill a certain purpose or plan a journey in which an agent is transformed from one state (i.e., place) to another.

With the exception of instruction, all the Hayes-Roth problem types correspond to one in Clancey's scheme. Instruction is regarded as a compound problem type. Other compound problem types are

System development: specification + design + assembly

System maintenance: monitoring + prediction + diagnosis + modification

However, Clancey made another, more important contribution. Comparing several expert systems from different domains, he found a common problem-solving method, independently of whether they worked with frames or rules. He called this problem-solving type "heuristic classification" and characterized it by means of an "inference structure diagram" (Fig. 11.3).

The essential feature of heuristic classification is that the solution is chosen from a given set of alternatives. Another feature is that the solutions are not derived directly from the raw data but via intermediate steps which are called data and solution abstractions. The data abstractions include the definitions of technical terms or qualitative generalizations. The solution abstractions describe a hierarchy, the final nodes of which are the solutions. The abstraction of the raw data can be accomplished by means of definite rules with data-controlled interpretation. The heuristic matching between the data abstractions and the

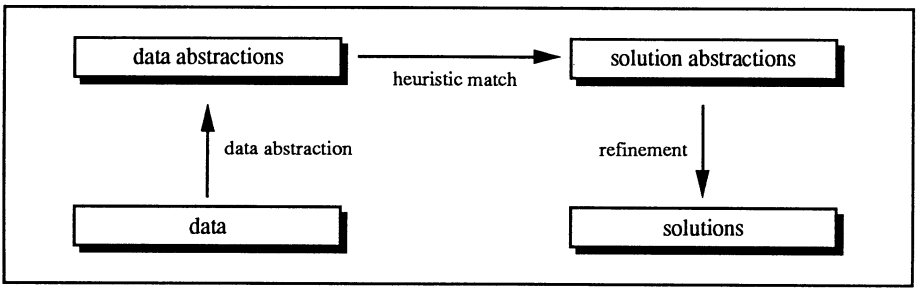



**Figure 11.3** Inference structure diagram of the heuristic classification

solution abstractions can be done with uncertain rules, and the refinement of solution abstractions to solutions with hierarchically structured rule packages.

Clancey distinguishes heuristic classification from simple classification, in which the solutions can be selected without uncertain rules or intermediate steps, and also from heuristic construction, in which it is not possible or expedient to list all alternative solutions. He mentions two variants of heuristic construction: the generation of the whole solution with the generate-and-test strategy (as in DENDRAL or in MYCIN's therapy algorithm) or the stepwise assembly of the solution from primitive components (as in HASP). The main difference between heuristic classification and heuristic construction is the necessity of providing a data structure for the latter in which the solution components can be stored in the correct order. This gives the scheme of problem-solving types shown in Fig. 11.4.

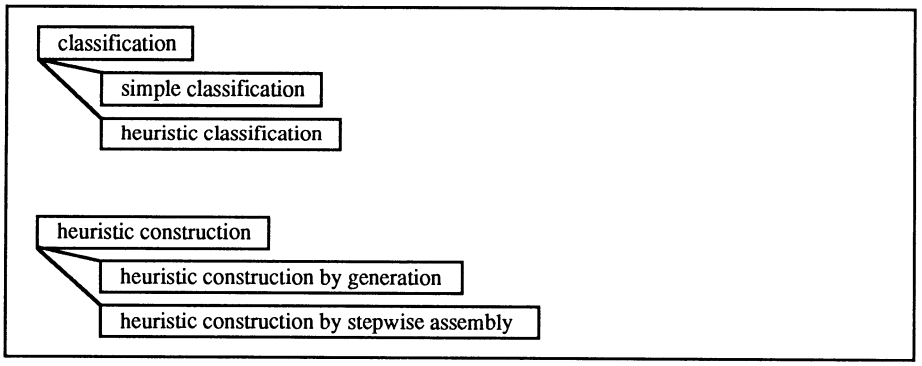



**Figure 11.4** Problem-solving types according to Clancey [85]

As problem-solving strategies for heuristic classification Clancey names data-controlled, goal-controlled or opportunistic (bidirectional) searching and the refinement of hierarchies. Further, he describes "causal process classification" as a subtype of heuristic classification, in which the heuristic matching is carried out in two steps: first patho-physiological conditions are derived from the data abstractions, and in the second step the diseases corresponding to these conditions are found. Clancey maintains that the model of heuristic classification can be impracticable in the second step. Since there are often many combinations

of patho-physiological conditions which point to the same disease, not all these combinations should be listed, but construction techniques should be used instead for this part of the task.[3]

Clancey lists the following techniques as problem-solving methods for construction: least commitment, following alternative paths, hierarchical refinement of solutions, backtracking, constraint propagation and debugging. Apart from debugging, which is not further explained, these are the techniques given in [Stefik 82] (see above).

Clancey's new contribution was the identification of the new problem-solving type heuristic classification, which he differentiated from simple classification and heuristic construction. However, the relationships between his problem types, problem-solving types and problem-solving methods remain unclear. On this subject he writes:

> "Relating the above methods for constructing solutions to problem tasks is beyond the scope of this paper. It is possible that the problem categories of Section 5 will be useful. Though they may prove to be an orthogonal consideration, as we discovered in distinguishing between classification and construction." [Clancey 85, p. 338].

## 11.3 The Approach of Breuker et al.

Breuker et al. [87,89] refined Clancey's two classification methods and combined them to one. For most types they give "interpretation models" in the form of Clancey's inference structure diagram (Fig. 11.3). They call the objects in their inference structure diagrams "meta-classes" (in Fig. 11.3 data, data abstractions, solution abstractions and solutions) and the relationships "knowledge sources" (in Fig. 11.3 data abstraction, heuristic matching and refinement). The main distinction in their classification is, as in Clancey's, whether a system is interpreted or constructed. As a connecting link between the two extremes they introduce a third main class, "modification", in which a given system is changed but maintains its basic structure. Since hardly any details are given for modification or for synthesis (construction), the most interesting part of their treatment is analysis (interpretation). The nomenclature is not entirely the same as in Clancey's scheme; for example, with Clancey "diagnosis" is a problem type and "classification" is a problem-solving type, whereas Breuker et al. use "diagnosis" as a subtype of "classification".[4] Also, it is not always clear whether problem types or problem-solving types are meant: Thus on the one hand interpretation models are given for those types which indicate problem-solving types, and on the other

[3]Clancey does not mention the contradiction that causal process classification is regarded as heuristic classification although the choice of problem solutions is insufficient.

[4]On the nomenclature: the word "diagnosis" has different meanings for different authors and is used as a subtype of classification (Breuker), a problem type (Hayes-Roth, Clancey) or a field of application for backward-chained rule systems (Harmon, see Sect. 11.5). We use diagnosis in the sense of classification to designate a problem-solving type.

hand it is said of the types of modification, for example, that they are often solved by classification techniques (e.g., selection of repairs or remedies), which indicates rather problem types from the user's point of view. New types are amongst others "assessment" and the division of "prediction" into the prediction of behavior and the prediction of values. Figure 11.5 shows only a selection of these types, omitting in particular those types which have hardly been described (above all in the area of synthesis).

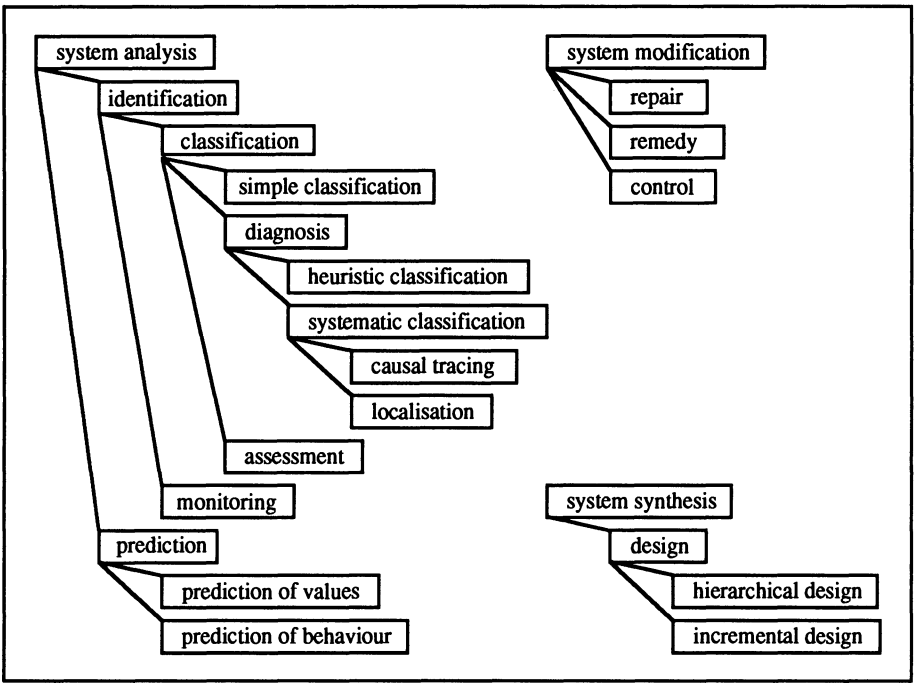



**Figure 11.5** A selection of the problem types and problem-solving types according to Breuker et al. [87]

The distinction Breuker et al. make between heuristic classification, which (like simple classification) he took over without change from Clancey, and systematic classification is that in the latter "a stronger algorithmic structure exists, since hypotheses can be eliminated without considering probabilities" [Breuker 87, p. 52]. The difference between the two subtypes of systematic classification is that in localisation there is a "part of" model of components which can be tested individually (example: the hardware diagnostics system HTE of Davis, see Sect. 8.4), whereas causal tracing uses cause-and-effect model of faults. The two subtypes do not differ in their structure.

Assessmant is a problem type from the user's viewpoint where the question is whether a case fulfills certain requirements (e.g., an applicant for a post). The result is a decision class (e.g., "suitable" or "unsuitable"). If the requirements and the attributes of the case are given explicitly, the problem solution structure corresponds to simple classification, otherwise the requirements and the attributes

must each be derived in a preparatory step (similar to data abstraction in heuristic classification). In [Breuker 87] the problem-solving structures of assessment and monitoring are very similar, since in both types a concrete case is compared with a norm.

In prediction a system description and an initial state are given. The prediction of values includes mainly the solution of sets of mathematical equations, e.g., in physics or optimization. If necessary a suitable set of equations must first be selected or constructed on the basis of the system description. In the prediction of behavior a series of possible future states is derived from the system description and the initial state. The system description consists of parameters with qualitative values (landmarks) and trends. The system of de Kleer and Brown and that of Forbus [AI Journal 84] (see Chap. 32) are cited as typical examples.

In all, the approach of Breuker et al. is very ambitious, but has not yet been developed very far. Their interpretation models with the inference structure diagrams can really be understood only if the problem-solving type is already known (e.g., Clancey's heuristic classification). On this subject the authors write[5]:

> "To the naive observer interpretation models may be no more than some sketchy flow diagrams with some highly abstract descriptions and some cryptic structures of predicates, standing for 'tasks'. Part of the depth has to be read in by understanding the descriptive tools; part of it has to be supplied by applying the models and refining these in a particular domain, where the "full glory" can be captured in greater detail." [Breuker 87, p. 1]

## 11.4 The Approach of Chandrasekaran

Chandrasekaran's group has also been working for some time on a scheme for problem classification, which is summarized in [Chandrasekaran 86,87]. They list the "generic tasks" shown in Fig. 11.6.

| | |
|---|---|
| Hierarchical classification | Classification with the establish-refine strategy |
| Abductive assembly | Diagnosis assessment by set covering |
| Hypothesis matching | Diagnosis assessment by probabilistc reasoning |
| Data base inference | Reasoning with raw data |
| Object synthesis by plan selection | Planning by refinement of skeletal plans |
| State abstraction | Prediction of states |

**Figure 11.6** "Generic tasks" according to Chandrasekaran [87]

In comparison with the problem classes which have been discussed so far, these tasks are on a lower level of abstraction. On the other hand they are more

[5]However, considerable progress has been made during the time of translation of this book, see, e.g. [Wielinga 92]

easily implemented in the corresponding tools, as has been achieved by Chandrasekaran's group. Solution strategies for concrete application problems should be describable by a combination of different generic types or their tools. In [Chandrasekaran 87] this is done for heuristic classification by means of a combination of the first four types: hierarchical classification as a global classification strategy, hypothesis matching for assessing the individual diagnoses, data base inference for deducing or requesting the symptoms and abductive assembly for checking whether the established diagnoses do in fact cover all observed symptoms.

The attractiveness of this approach results from the possibility of covering a broad spectrum of problem classes with a limited set of generic tasks and the corresponding tools. An evaluation requires, however, the analysis of very many different problem classes, and in [Chandrasekaran 87] this is only hinted at with the diagnosis example, which is an already well understood problem solution type. It is worth noting that the level of abstraction of Chandrasekaran's generic tasks with their associated problem-solving methods is intermediate between the abstraction level of the general problem-solving techniques for rules, frames, constraints, probabilistic and non-monotonic reasoning and that of the problem-solving types discussed so far, such as heuristic classification, since the latter is regarded as a compound type.

Chandrasekaran, like Clancey, was one of the first expert system research workers to recognize the significance of abstraction levels above that of the basic knowledge representations. He also started early with the development of tools for his generic tasks. However, his research program suffers from the fact that his tool-box is still too small, and not enough different high-level shells have as yet been developed to demonstrate convincingly the postulated multi-usability of the components. Thus in spite of the theoretical attractiveness of the idea of generic tasks for toolkits at a medium level of abstraction, it is still not clear how useful this concept is compared with the development of problem-specific tools for the problem-solving types.

## 11.5 The Approach of Harmon

The approach of Harmon [ESS 88] is based, so the author maintains, on an analysis of the actual application of expert system in the industrial practice. According to the statements cited in Sect. 3.3, his aim is to provide users with information about which tools have been used for which problem types so far. He comes to the conclusion that 90% of all expert systems belong to one of the following five problem types:

- procedural problems,
- diagnostics (debugging),
- monitoring,
- design problems,
- scheduling problems.

The procedural problems include those which can be solved by means of flow diagrams or decision tables. Harmon maintains that conventional techniques are also sufficient for this purpose, but that expert system tools are often preferred because they are easier to use and adapt. Diagnostics and monitoring are distinguished by the fact that the former is controlled mainly by backward chaining from the diagnoses to the data, and the latter by forward chaining from the data to the diagnoses (recommendations). Design problems (e.g., the configuration of computers) include the assembly of an object from components. Scheduling differs from design in that temporal constraints must be taken into account in addition.

Harmon also indicates which types of expert system tools can be used for solving each of the problem types, which are further subdivided into simple, medium and complex. The tools are classified as procedural, inductive, backward chaining rule-based, forward chaining rule-based and hybrid, i.e., object- and rule-based (see Fig. 11.7). Of course there is some overlap between the different areas.

| | |
|---|---|
| Simple procedural problems | Procedural and inductive tools |
| Medium and complex procedural problems | Conventional languages and CASE tools |
| Simple and medium diagnostic problems | Backward chaining rule-based tools |
| Simple and medium monitoring problems | Forward-chaining rule-based tools |
| Simple design problems | Forward chaining rule-based tools |
| All other problems (complex diagnostic and monitoring problems, medium and complex design problems, all scheduling problems) | Hybrid tools (object- and rule-based) |

**Figure 11.7** Suitability of the available expert system tools for the different problem types

Harmon makes the prognosis that there will be increasing use of problem-specific tools in the years to come.

Harmon's classification, like that of Stefik and Hayes-Roth (see Sect. 11.1), describes the problem types from the point of view of the user and is limited to those types which are most frequently encountered in practice. Harmon's prognosis concerning problem-specific tools indicates that he himself is not altogether satisfied with his relationship, "observed in practice", between general tools and problem types.

## 11.6 The Approach of McDermott

For some years McDermott's group at the Carnegie-Mellon University and the Carnegie Group have been developing a series of problem-specific expert system tools [Marcus 88]. In [McDermott 88] the aim of this development is stated to be that the problem-solving methods of the tools should represent steps in the direction of a classification of problem-solving types. McDermott lays special emphasis on the statement that a classification should be based on precisely defined

problem-solving methods, so that for every problem-solving type a tool can be provided which may be used by experts without the help of knowledge engineers. The work carried out so far has led to the implementation and application of the following tools:

- MORE is a tool for heuristic classification. It is remarkable that the knowledge acquisition component first collects knowledge about fault models and then transforms this semi-automatically into heuristic knowledge (see Sects. 15.4 and 17.4).
- MOLE is a tool for classification with fault models using the technique of set covering (cover and differentiate, see Sect. 17.4). Unlike the predecessor MORE, the problem-solving component works directly with fault models.
- SALT is a tool for heuristic construction with the propose-and-revise strategy (see Sect. 23.4).
- SIZZLE is a tool for case-based construction in dimensioning problems, e.g., dimensioning a computer system [Offut 88].
- KNACK is a tool for case-based construction which helps in writing reports. As input it requires a complete example report and knowledge about the concepts used in the report. From this KNACK can generate new reports by generalizing the example and creating new instances.

McDermott cautiously calls the problem-solving types on which the tools are based "data points" for a classification. He does not yet attempt a systematic analysis of problem-solving types. His contribution is the requirement that a classification of problem-solving types should be operationalized by means of problem-specific tools – a requirement which he has put into practice in a number of special examples. Compared to Chandrasekaran's tool-kit (see Sect. 11.4), the tools of McDermott's group are on a distinctly higher level of abstraction and are designed for independent use, but not for coupling.

## 11.7 Discussion

The above review shows that a satisfactory classification of problem-solving types does not yet exist. However, there are some indications, especially in the work of Chandrasekaran, Clancey, Breuker and McDermott, that there are at least a few problem-solving types with problem-solving methods which are nearer to the problem and therefore more useful than general problem-solving methods, but can still be described independently of the concrete domain. With Breuker they mainly play the role of interpretation models with which the knowledge engineer can better analyze verbal data when analyzing domains or communicating with specialists. McDermott attempts to operationalize them in the form of problem-specific tools which are intended for direct use by the expert. A detailed discussion

of the similarities and differences in the basic assumptions of the approaches of Chandrasekaran, Breuker and McDermott can be found in [Karbach 90].

According to McDermott, the important difference between strong, problem-orientated problem-solving methods and general, weak methods is that the former require only highly structured control knowledge, or none at all, since the control is laid down in the problem-solving method. On the definition of the control knowledge of a problem-solving method he writes: "The control knowledge of a problem-solving method includes (1) an algorithm which specifies when which knowledge is needed for the identification, selection and performance of actions and (2) all the knowledge which the method has for the selection of the action" [McDermott 88, p. 255]. An inflexible determination of the control limits the application of the method but enables strong assumptions to be made about the domain knowledge; the latter therefore has a regular structure with fixed, limited functionality, which simplifies knowledge acquisition considerably. In contrast, weak methods have a much broader application, but their simple control structure can be supplemented by relatively complex domain-specific knowledge. Since hardly any assumptions can therefore be made about the domain knowledge, weak methods offer little assistance in knowledge acquisition. As an example of domain-specific control knowledge, McDermott mentions meta-rules which control the selection of other rules. Domains such as that of R1-XCON, in which meta-rules may be unavoidable, cannot be managed with strong problem-solving methods. From this McDermott concludes that strong problem-solving methods are not available for every domain, but that, when they are available, their usefulness justifies giving them priority over the less specialized weak methods. From this, one can derive a specialization hierarchy of problem-solving methods in which each additional constraint of the knowledge range simplifies the knowledge acquisition and hence increases the strength of the problem-solving method. The difference between heuristic and simple classification may be regarded as an example of this, since the latter is a specialization of the former with elimination of uncertain knowledge.

# 12. Principles of Problem-Solving Methods

In this chapter we give a survey of the problem-solving methods described in detail in the following chapters and their basic principles. The search for problem-specific problem-solving methods and their operationalization in shells which experts can use independently corresponds to the approach of McDermott (Sect. 11.6 and 11.7). Compared to the few "data points" of McDermott, however, considerably more problem-solving methods will be described.

The term *problem-solving method* is used for an algorithm which determines how domain-specific knowledge is used for solving problems. In comparison, a sorting algorithm, for example, is parametrized on only one level, since it refers only to input data and not to domain-specific knowledge. Since the way in which knowledge is used imposes certain restrictions on the form of reprsentation[1], the specification of a problem-solving method must also include information about the knowledge representation. The domain-specific knowledge consists of problem characteristics, problem solutions or elements thereof and possibly auxiliary objects and relational knowledge. These objects are characterized by attributes, the contents of which are unknown at the level of the problem-solving method. The strength of a problem-solving method depends upon how far the representation and the function of the domain-specific knowledge are determined by a specification of the object types with attribute structures and their use.

*Weak (or basic) problem-solving methods* are open to a great variety of knowledge representations and/or functions. The have a broad range of applications, but for this reason they are of no great help in knowledge acquisition. Examples of basic problem-solving methods are: (1) The basic knowledge representations with their corresponding inference strategies such as (a) rules with forward or backward chaining, (b) frames with inheritance, attached procedures and transfer of messages or (c) constraints with local propagation (see Chaps. 5–7). The rule actions, messages or constraints may have widely differing functions (for example a rule action can effect a logical implication, a change of a situation, a dialog control or an internal program control). (2) General procedures such as the difference method in Planning (see Sect. 26.2), on which the General Problem

[1]Problem-solving methods and knowledge representations are like two sides of the same coin, since problem-solving methods require a certain determination of the knowledge representation, and knowledge representations are meaningful only with a view to associated processing procedures. The two concepts should not therefore be interpreted as though the structure of knowledge could be separated from its function; it is just that one aspect or the other may be highlighted.

Solver [Newell 72] is based, or the hypothesize-and-test strategy (Sect. 13.4) for diagnostics. The basic idea of the latter method can be combined with different types of knowledge: In case-based classification, for example, cases (*hypotheses*) are pre-selected from a data base and then compared (*tested*) in detail with some measure of similarity. Similarly, in heuristic classification *hypotheses* are set up by forward chaining of rules and then individually *tested*, in which process additional questions may be put to the user. In set-covering and functional classification also, *hypotheses* must first be generated, which are then *tested* in a simulation model. Thus although the hypothesize-and-test strategy has a relatively broad range of applications, several stronger problem-solving methods may be defined according to the type of knowledge.

In *strong problem-solving methods* the representation and function of the domain-specific knowledge are more or less fixed. They are therefore less flexible, but offer – where applicable – comprehensive assistance in knowledge acquisition. As was shown above for the example of the hypothesize-and-test strategy, strong methods can be regarded as specializations of weak methods. The more precisely the types of objects and relations in the knowledge representation are defined, and the further its function is restricted, the stronger the method is. Amongst the more important types of restrictions in our problem-solving methods are:

- Restriction of the use of variables.
- Definition of object types by extensive determination of their attribute structures.
- Determination of the communication paths between the objects (or object types) by the representation of the rules (and constraints) as attached procedures of the objects (or object types).[2]
- Global control, in many cases by agenda control. An agenda is an ordered list of tasks from which those to be executed are selected according to certain criteria, and to which new tasks can continually be added.
- Knowledge input mainly by instantiating given object types and specifying the relationships between object instances. This supported by graphical input forms.

The exact determination of the object types and their attribute structures is very problematical, because additional attributes are often needed for the special requirements of a domain. These attributes are unnecessary for the majority of the domains and make the problem-solving method more complicated. We will treat them as optional additions to the problem-solving method, which can be switched on or off as needed without substantially changing the basic algorithm. The determination of the exact object types and attribute structures requires

[2]The brackets denote special cases, which on account of their generality should if possible be avoided. However, this difficult in some problem-solving methods, e.g., the propose-and-exchange strategy (Chap. 24).

a great deal of experience with the particular problem-solving method. Since such experience is still scarce, we will generally describe only the basic structure. Additional mechanisms will be given only in the case of heuristic classification, which is particularly well understood (Chap. 18).

## 12.1 Survey of Problem-Solving Methods

The three main types of problem-solving methods which form the basis of our grouping are:

- *classification*, in which the solution is selected from a set of given alternatives,
- *construction*, in which the solution is assembled from given, primitive components, and
- *simulation*, in which it is determined how a given system model reacts to certain inputs.

In the grouping of problem-solving methods of Fig. 12.1 the three main types do not carry equal weight. In the "classification" group the subdivision is well elaborated and the methods are fairly strong. Under "construction", problem-solving methods have been given only for the sub-group "heuristic construction", and these, particularly the least-commitment strategy, are relatively weak. The subdivision of "simulation" is even more fragmentary. The differences in the completeness of the subdivision and in the strengths of the problem-solving methods reflect the present state of expert system research.

The motivation for this grouping structure is given in three introductory chapters surveying, respectively, classification (Chap. 13), construction (Chap. 21) and simulation (Chap. 29), and the problem-solving methods are described in detail in the subsequent chapters. For most of the problem-solving types the structure of a corresponding shell is outlined. Details are given concerning the representation, manipulation and acquisition of knowledge, and some examples are described. Since many problems are best solved by a combination of different problem-solving methods, the shells should be integrable. The reciprocal relationships are investigated in Chaps. 33, 34 and 35.

## 12.2 Restriction of Use of Variables

The view of expert systems as knowledge media is favored by the use of easily understandable problem-solving methods and a clear, if possible graphical knowledge representation for the knowledge holder (expert) and the knowledge receiver (user). These aims are endangered by an uncontrolled use of variables in a knowledge base. In problem-solving methods we will therefore avoid as far as possible the use of "reference variables", which point to variable objects, i.e., objects which

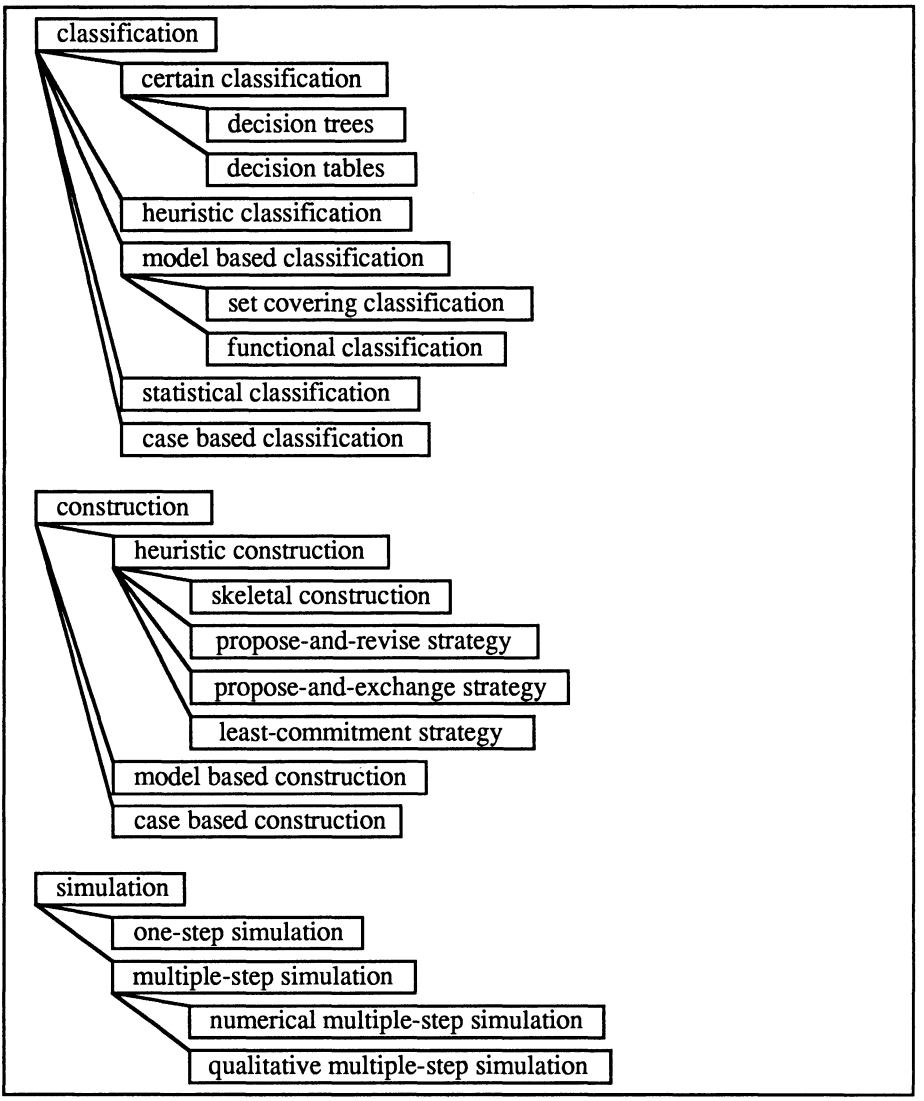



**Figure 12.1** Survey of problem-solving methods

change according to the context. "Container variables", on the other hand, which represent the changing values of a fixed object, are unproblematical, as long as the values are not themselves pointers to other objects.

The significance of reference variables and the techniques for eliminating them may be illustrated by means of the use of variables in first-order predicate logic. To make the distinction clear consider first a typical rule containing only container variables: "If the value of measurement 1 is 40 units greater than the value of measurement 2, then we have condition $Z$," or $g(v(M1), v(M2), 40) \rightarrow Z$, with $g$ = greater, $v$ = value of and $M$ = measurement. The predicates refer to the

values of objects ($M1$ and $M2$), where the values may vary but the objects are fixed. Reference variables, however, do not refer to fixed objects. Their meaning in predicate logic is determined by "for-all" or "there-exists" quantifiers.

In expert systems "for-all" statements can usually be transformed into a set of basic propositions without variables, since one normally considers only finite sets of individuals. The most important ways of achieving this are hierarchy formation, instance generation and implicit "for-all" quantification.

In hierarchy formation a new object is introduced which represents the base set of the "for-all" quantified variables. The new object is derived by means of rules; one rule is needed for each element of the base set and can be used in other deductions. Hierarchy formation requires that the right-hand side of a "for-all" quantified implication does not contain the same variables as the left-hand side.

*Example:* For all $c$ ($c$ in $c'$): $d(c) \rightarrow l$ and $v$. *Concrete example:* For all cyl. (cyl. = any cylinder in the car): damaged(cyl.)→ low-power and vibrations.
*Transformation:* $c_1 \rightarrow c''$, $c_2 \rightarrow c''$,..., $c_n \rightarrow c''$, $c'' \rightarrow l$, $c'' \rightarrow v$.
*Explanation:* $c''$ is a new concept, which corresponds to $c'$. It is derived from the $c_k$ and then used for other conclusions.

If both sides of an implication refer to the same "for-all" quantified variable, one can use static or dynamic instance generation, in which all instances are explicitly listed. In the static method the instances are pre-calculated in the knowledge acquisition phase (corresponding to the program compiling time), while in dynamic instantiation they are generated in the problem-solving phase (i.e., in the program running time). Static instance generation requires that the range of values is statically fixed, whereas dynamic instantiation can depend on other variables. Since dynamic instance generation may make a problem-solving method considerably more complicated, its use requires an exact analysis of the problem situation. If several variables of an implication are affected, the number of instances increases exponentially.

*Example:* For all $s$ ($s$ in $s'$): $b(s) \rightarrow d(s)$. *Concrete example:* For all s.p. (s.p. = any sparking plug in the car): black-sooty(s.p.) → damaged(s.p.).
*Transformation:* $b(s_1) \rightarrow d(s_1)$, $b(s_2) \rightarrow d(s_2)$, ..., $b(s_n) \rightarrow d(s_n)$.
*Explanation:* For each element $s_k$ of the range of values of $s'$ a special rule is produced. In static instantiation $n$ is fixed, so that for, e.g., four sparking plugs four rules are needed. In dynamic instantiation $n$ is taken over from another variable, e.g., when the user is asked how many cylinders the particular car has. If the answer is "two", only two instances are produced.

Implicit "for-all" quantification can be used when the problem is restricted to a partial problem in such a way that all formulas are quantified with the same variable, which always points to the same object. Then the quantification can simply be removed from the formulas and transferred to the partial problem as a whole. Thus if a sub-program exists for the partial problem, calls of this program with different parameters correspond to different instantiation of the variables.

*Example:* For all $p$ ($p$ in $p'$): $t(p) \rightarrow i(p)$. *Concrete example:* For all pat. (pat. = any patient): temperature(pat.) → infection(pat.).
*Transformation:* $T \rightarrow I$.

*Explanation:* If all formulas in a partial problem contain the same "for-all" quantified variable, e.g., the same patient with all symptoms, diagnoses and rules, this variable can simply be dropped in all the formulas. Different patients are handled by calling the sub-program with different input data. However, the characteristics of different patients can then no longer be directly compared.

Unlike "for-all" quantified variables, those quantified by "there exists" can hardly be eliminated. Their role in problem-solving methods usually consists in referring to objects indirectly via their properties. For their evaluation a function (Skolem function) must be available which supplies a suitable object on the basis of the required properties. The identity of the object can be pre-calculated in the knowledge acquisition phase only if the properties and the base set for selecting the object are given statically in the knowledge base. Normally, however, this is not possible. The "there-exists" quantified variables must then be treated with special mechanisms which are integrated in the problem-solving method and can make it considerably more complicated. As in the case of dynamic instantiation, a careful analysis of the domain is necessary to find out whether the indirect reference of objects through their properties is unavoidable.

A typical special mechanism, which is often used in problem-solving methods, is agenda control, where an object best fitting some given properties is selected by means of an agenda. Examples are situations such as "As the next object to be assigned, always select, from those which have not yet been assigned, the one which has at present the smallest number of free alternatives," or "From a set of hypotheses always test the one which is at present most probable."

The more one succeeds in avoiding reference variables in problem-solving methods, the easier it is to support knowledge acquisition with graphical methods, since static, "associative" relationships between object instances are generally much better suited for representation as graphical hierarchies or networks than variable relationships are. A disadvantage is the additional storage space needed for the transformation of pure "for-all" statements, but this is often necessary anyway, because rules which seem to be general may have to be put into a more concrete form on account of their many exceptions. In critical cases the number of instances can often kept low by using good generating heuristics in dynamic instance generation. A serious disadvantage is the limited expressive strength of "there-exists" expressions. This must be compensated by special mechanisms, such as agenda control, which depend upon the special features of the domain. The simplification of the problem formulation is therefore obtained at the cost of loss of generality. Only practical experience can finally show whether this is advantageous or not.

We believe, however, that it is highly probable that many complex applications can be successfully handled associatively, because there is evidence that human beings tend to solve problems on the basis of rich associations and special problem-solving techniques rather than with complex, general inferences. In chess, for example, grandmasters do not think so many more moves ahead than beginners, as many people think, but they assess the positions more realistically on account of their experience. In studies it has been estimated that grandmas-

ters know about 50,000 position types [Groote 65]. The situation is similar in other domains, as can generally be seen in the respective terminologies, in which all important concepts are explicitly named. The special terminology with its definitions and references can be regarded as a concept network. It makes possible efficient, associative problem solving with methods designed for the special domain.

The basic structure described above may be termed "associative problem-solving with semantic networks." In a "semantic network" all concepts are pre-defined, named nodes, and the known relationships are explicit, labeled links. "Associative problem-solving" means that in principle the solution is found by local propagation along the pre-defined links after certain nodes (the given problem attributes) have been activated.

The main advantage of using problem-solving methods which are similar to human thinking processes is that the knowledge acquisition is simplified, since the experts do not first have to translate their knowledge into a new code. Since the competence of expert systems is determined by the quality of their knowledge, which comes from experts, re-coding should be avoided if possible (unless the form of representation chosen by the experts is thought to be of poor quality). In logic one usually tries to express basic knowledge as concisely as possible in the form of axioms, which means that powerful inference techniques are required for proving theorems. This approach is not expected to be highly successful in most domains of expert systems. Instead, the complete terminology of the domain and the corresponding problem-solving methods should be used.[3]

## 12.3 Structure of Knowledge Representation

The knowledge representations should make it possible to encode the special terminology as naturally as possible. According to the metaphore of the semantic network, it is required to represent knowledge about concepts and about relationships. Concepts can be represented by frames, and relationships by inheritance hierarchies, attached procedures, rules or constraints. The main task of knowledge representation is the recognition of specialized, generic types of frames, rules or constraints of the particular problem solution type. For every solution type there are usually certain sorts of frames (e.g., symptoms and diagnoses in diagnostics) for which a characteristic attribute structure can be given (e.g., for symptoms: name, question text, type of answer, value range, actual value, etc.). The same applies to generic types of rules and constraints with typical predicates and actions. Concrete concepts and relationships of the domain are represented as objects by instantiating the generic object types, i.e., by giving values to the attributes. Since the rules or constraints generally refer to concrete frames, direct references can be made, which represent possible paths of communication between the objects. This is accomplished by giving each frame an attribute consisting of

[3]We suspect that, in fields where the formulation of concise axioms is expected to be advantageous (e.g., in many fields of physics), this has already been done.

a list of all rules and constraints which act on that frame. These attributes can then play the role of attached procedures, i.e., the rules are activated by adding or modifying a value of the frame. A rule is connected to all objects of its pre-condition through the attribute "meaning" (i.e., this attribute contains all the rules which draw conclusions from the object, like the facet if-needed of FRL in Sect. 6.1) and to all objects of its action part through the attribute "derivation" (i.e., this attribute contains all the rules of an object which derive its value, like the facet if-added of FRL in Sect. 6.1). In general three types of attribute may be distinguished: static attributes (e.g., name, value range), whose values are laid down by experts in the knowledge acquisition phase, semi-static attributes (e.g., meaning and derivation), which define communication paths between the objects and are derived from the static attributes, and dynamic attributes (e.g., the "actual value" of the object type "symptom" described above), which are set or derived during problem solving.

## 12.4 Structure of Knowledge Manipulation

Problem-solving methods are based on the exchange of messages between objects as in object-orientated programming [Meyer 88]. Rules and constraints are regarded as attached procedures of the objects to which they refer (see Sect. 6.2). Since the communication paths between the objects are pre-determined and stored as attributes, the inference process is very efficient and permits an exact, efficient withdrawal of values.

Frequently (partial) sequencing of the parallel exchange of messages through agenda control is desirable or necessary If the processing of a message is complicated and the object can potentially receive many messages, the efficiency can often be increased by first collecting the messages before the object reacts. Also, when questions to the user are necessary, the questions posed by different objects should be coordinated through an agenda. If the quality of the solution depends strongly on the order of processing, it should be easy to change the ordering criteria of the agenda (see e.g., the propose-and-exchange strategy, Chap. 24).

To realize the agenda control the exchange of messages must be modified. To some messages the object does not react directly, but sends a message to the agenda, on which the object is placed according to some priority criteria. When no more messages are active, an object is selected from the agenda, which sets off a new round of messages.

## 12.5 Structure of Knowledge Acquisition

The internal representation of knowledge is an object-attribute-value structure for all object types. One form of knowledge acquisition, which is simple to implement, is to fill in directly the corresponding frame, rule and constraint schemes. The great disadvantage of this method is that relational knowledge must be "lin-

earized", i.e., it must refer to a (randomly selected) object. This hampers the input, since references must be made continually in a complicated way to other objects, and also makes it difficult to obtain a general view of the knowledge base. Experts therefore often express their knowledge in graphical form, referring not only to the name but also to the position of an object. This principle should therefore also form the basis of knowledge acquisition, and should be made available by the knowledge acquisition components for the different problem solution types (cf. [Gappa 91]). The most important forms are tables for displaying the relationships between object pairs (Fig. 12.2), decision tables for connecting several objects (Fig. 12.3), fairly strict hierarchies (Fig. 12.4), networks for tangled hierarchies and cyclic graphs (Fig. 12.5), forms for the input of local object knowledge (Fig. 12.6), formula editors for the input of numerical knowledge (Fig. 12.7) and curves for the input of complex relationships between parameters (Fig. 12.8). From these main types different sub-types must be produced according to the different object types (e.g., different types of formulas or tables), and for each sub-type different instances for the concrete input of knowledge. In the descriptions of problem-solving methods contained in the following chapters, the main forms required for graphical knowledge in input are given in the sections on knowledge acquisition. A concrete idea of graphical knowledge acquisition is shown by our knowledge acquisition system CLASSIKA for heuristic classification (Sect. 15.5).

| | Object21 | Object22 | Object23 | ... |
|---|---|---|---|---|
| Object11 | × | × | | |
| Object12 | | × | | |
| Object13 | × | | | |
| Object14 | | | × | |
| ... | | | | |

**Figure 12.2** Tables for indicating relationships between pairs of objects

| | Rule1 | Rule2 | Rule3 | Rule4 | ... |
|---|---|---|---|---|---|
| Object11 | × | × | × | | |
| Object12 | × | × | | × | |
| Object13 | × | | × | × | |
| ... | | | | | |
| Object21 | | × | × | × | |
| Object22 | × | | | | |

**Figure 12.3** Decision tables for connecting several objects

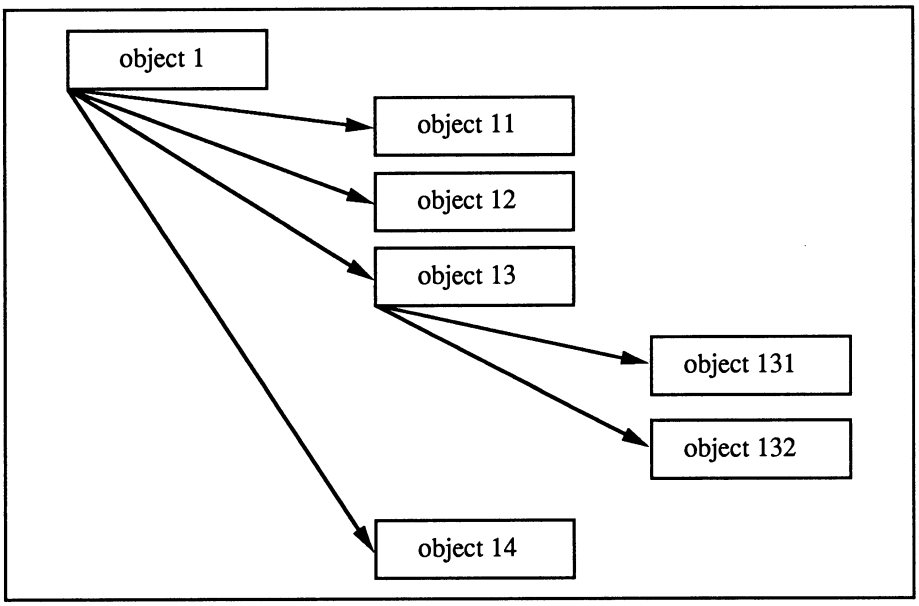



**Figure 12.4** Fairly strict hierarchies

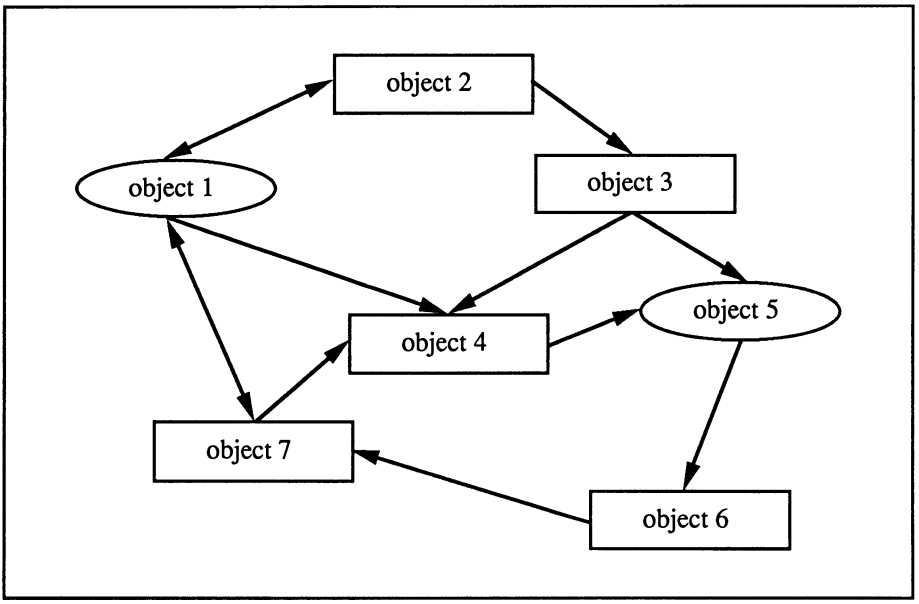



**Figure 12.5** Networks for hierarchies and cyclic graphs

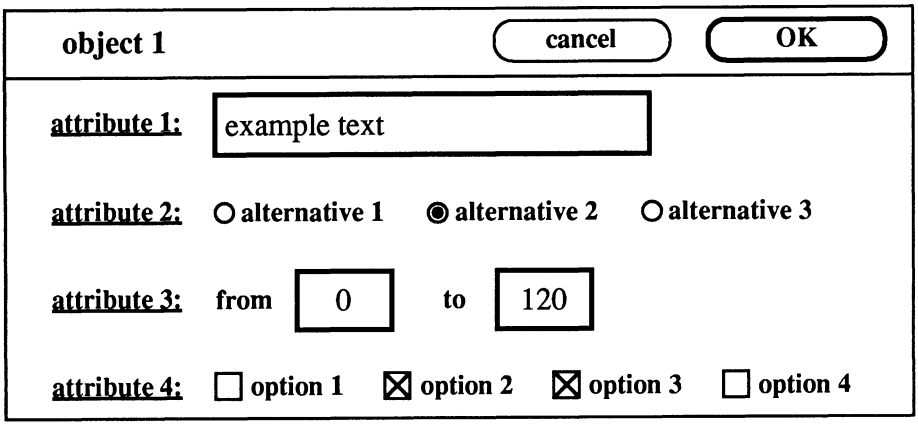



**Figure 12.6** Forms for the input of local object knowledge

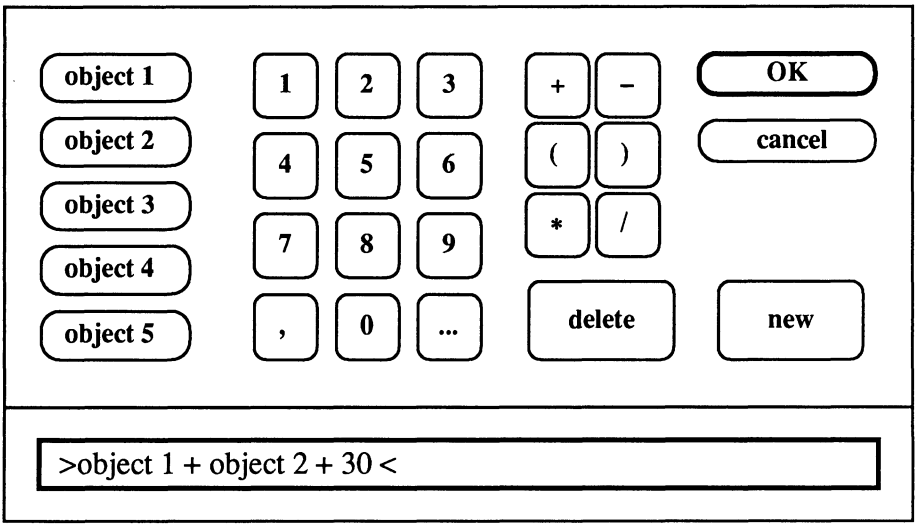



**Figure 12.7** Formula editors for the input of numerical relationships

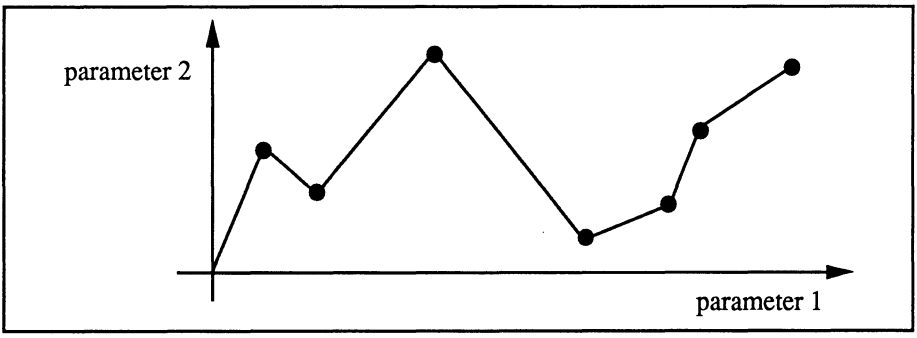



**Figure 12.8** Curves for the input of complex relationships between parameters

# Part IV

# Classification

# 13. Survey of the Problem-Solving Type Classification

The term classification (diagnostics) is given to the solution procedure for problems with the following properties:

1. The domain consists of two finite, disjunctive sets – one containing observations and the other problem solutions – and of typically uncertain, complex knowledge about the relationships between these two sets.

2. A problem is defined by a given subset of observations, which may be incomplete.

3. The result of the classification is the selection of one or more solutions to the problem.

4. If the quality of the solution can be improved by considering additional observations, one of the tasks of classification is to determine which additional observations are to be requested.

Figure 13.1 illustrates the basic principle of classification. The derivation of the solutions from the observations usually takes place via some intermediate steps, the "intermediate diagnostic structure." In this introductory chapter, starting from the great variety of domains, an attempt is made to achieve a classification of problem-solving methods via problem types and requirements for problem-solving methods.

## 13.1 Domains

In order to test whether a problem domain belongs to classification, the two sets of observations and solutions should be given explicitly. At first sight the requirement of an explicit list of all solutions seems quite difficult to fulfill in practice. Experience has shown, however, that once a domain which at first seems complex is well understood, one often finds a relatively small number of frequently recurring, stereotyped solution patterns, which are usually given special names by the experts on account of their importance. Solving problems is then reduced to recognizing these patterns.

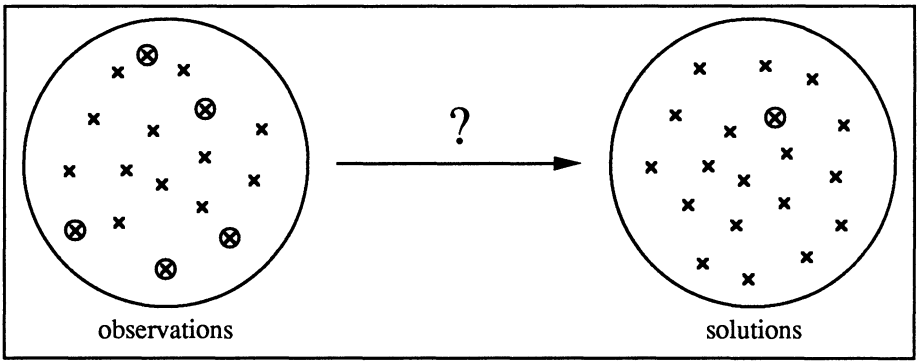



**Figure 13.1** Basic structure of classification with observations (symptoms) and solutions (diagnoses). A symptom can have different values. Classification can be regarded as a mapping of all possible combinations of symptom values on the power set of the diagnoses. In practice, however, only a few diagnoses – often only one – are sufficient.

As an illustration a list of domains with the corresponding sets of observations (O) and solutions (S) is given below.[1] The observations also include general basic data, which are not explicitly given as they are unspecific.

### 13.1.1 Technical Applications

a) Quality control:

- Engine diagnostics on the test stand. O: measured data with different loading of the engine and the observed symptoms such as noises, the color of the exhaust gas or escaping oil. S: defective components of the engine. Example: IXMO* [Puppe 85a, Premauer 86, Ernst 88].
- Gearbox diagnostics. O: measured data of the gearbox. S: Gearbox defects. Example: DAX* [Puppe 91].

b) Production process diagnostics.

- Circuit board production. O: visible faults, machine and process parameters. S: Suggestions for eliminating faults, e.g., exchange of defective machine components, adjustment of the process parameters. Example: LEDIS* [Achtstätter 89].
- Spray casting in the production of elastomers. O: visible defects of the rubber particles produced, process parameters, characteristic data of the machine and the raw rubber mixture. S: suggestions for eliminating faults. Example: EFFEKT* [Nedeß 88, Plog 90].

[1] Applications which were constructed using the tools MED1 or MED2 are marked *.

- Process diagnostics in automobile painting. O: visible defects in the paint, process parameters, characteristics of the paint material and the machine. S: Suggestions for eliminating faults. Example: LADIS*.

c) Repair diagnostics.

- Automobile engine diagnostics. O: complaints and observations of the driver, tests in the workshop. S: defective engine components. Example: MODEX* [Walser 91].
- Computer hardware diagnostics. O: complaints of the user, error messages of the computer. S: defective hardware components.
- Diagnostics of liquid chromatographs. O: characteristics of the chromatograph monitor. S: Solvent or apparatus fault.
- Vacuum pump diagnostics. O: Vacuum pressure values at different times, machine parameters. S: Defects of the pumps involved. Example: VADIS*
- Robot diagnostics.

d) Monitoring.

- Monitoring turbine machinery. O: Noises, vibrations, temperatures, results of oil analysis, etc. S: shut down or continue to run, faults in operation (e.g., over-loading), in the materials (e.g., oil) or the machinery (e.g., insufficient sealing from the power unit). Examples: TADIS* [Borrmann 88], TUMAD* [Keim 89].
- Monitoring power stations.

e) Network diagnostics.

- Diagnostics of electric power networks.
- Diagnostics of telephone networks.
- Diagnostics of computer networks.
- Diagnostics of water supply networks.

f) Other technical applications.

- Selection of tools in the construction of work pieces. O: geometrical properties of a part of the work piece, required precision, material. S: Selection of one of the available tools for constructing the part, iteration for all parts of the work piece.
- Data base tuning. O: characteristics of the data base monitors. S: Recommendations for optimizing the data base, detection of a general overloading of the computer or a poor data base design. Example: SIUX* [Dickmann 86].

### 13.1.2 Applications in Medicine and Environment

a) Medical diagnostics in the narrow sense.

- Diagnostics of chest pains. O: intensity of the chest pains, other accompanying symptoms, physical examination, technical investigation. S: Diseases of the heart, lung, etc. Example: MED1/chest pains* [Puppe 85b].
- Diagnosis of multiple sclerosis. O: anamnesis, findings, examinations. S: differential diagnosis, assessment of the gravity and prognosis. Example: MSX* [Nossem 89].
- Diagnostics in inner medicine. O: anamnesis, findings, technical investigations. S: known diseases. Example: INTERNIST/QMR [Miller 82, 86].
- Diagnostics of joint pains. O: intensity of joint pains, accompanying circumstances. S: rheumatic diseases. Example: Schewe's System [Schewe 90].

b) Choice of therapy in medicine.

- Determining the antibiotics therapy for bacterial diseases. O: anamnesis, early laboratory findings. S: antibiotics for the most probable causative organisms. Example: MYCIN [Shortliffe 76].
- Determining diabetes therapy. O: diabetes anamnesis, findings and tests. S: diabetes therapy elements such as type of diet, type and form of administration of insulin, etc. Example: DIACONS(*).

c) Interpretation of laboratory data.

- Evaluation of lung function test. O: test results (about 30 data). S: suspicion of lung diseases. Example: PUFF [Aikins 84].
- Evaluation of lipoprotein test. O: test results. S: Suspicion of metabolic diseases. Example: PRO.M.D/lipoproteins [Trendelenburg 86].

d) Patient surveillance in intensive wards.

- Surveillance of the iron lung. O: vital patient data such as pulse, blood pressure, respiration frequency, etc. S: mode of the mechanical respiration assistance. Example VM [Fagan 84].

e) Diagnosis of animal or plant diseases.

- Diagnostics of room plants.
- Diagnostics of milk cows.

f) Diagnostics of environment damage.

### 13.1.3 Application to Services

a) Banking applications.

- Advice on the purchase of shares or securities. O: sum to be invested, customer's preferences and readiness to take risks, investment period. S: options for investment.
- Granting loans.
- Checking credit card holders. O: statement of account, trustworthiness of customer, sum involved. S: OK/ not OK. Example: American Express system, which draws characteristic data from various data bases.

b) Applications in law.

- In the fields of work, environment or tax laws, for example.
- Fulfilling the regulations for handling asbestos.
- Fulfilling export regulations.

c) Catalog selection.

- Choice of data base. O: hardware conditions, amount of data stored in the base, type of use, performance requirements. S: data banks. Example: DABAS* [Mütter 89].
- Choice of computers.
- Choice of automobiles.
- Choice of books: O: customer profile. S: book titles. Example: GRUNDY.
- Advising in expert system projects. O: expected costs and profits, quality of the knowledge sources, knowledge fluctuation, etc. S: feasibility. Example: XPS advising* [Puppe 87, Sect. 4.3.1].
- Choice of expert system tools. O: problem type and special requirements of the domain, conditions of knowledge acquisition, hardware conditions, etc. S: expert system tools.

d) Product marketing.

### 13.1.4 Image and Object Recognition

a) Identification of plants.

- Identification of fungi. O: appearance, odor, feeling to the touch, description of the surroundings, season, weather. S: fungi. Example: PIDIAS* [Neis 88].

b) Recognition of objects on a conveyor belt.
c) Recognitions of particular features in image processing.

### 13.1.5 Other Applications

a) Recognition of geological formations.

- Searching for mineral deposits. O: rock types, vegetation, results of test borings. S: characteristic constellations. Example: PROSPECTOR [Gaschnig 82].

b) Selection of suitable objects for given locations.

- Selection of room plants. O: light intensity, temperature, humidity, sunlight, etc. S: room plants.

This detailed listing of various applications of classifications could be continued (see for example the surveys of expert systems in the German-speaking area in [Mertens 90] or of German diagnostics expert system projects in [Puppe 89]). It indicates the bandwidth of the problem-solving type classification, which extends far beyond medical and technical diagnostics. Almost all the classification systems listed above operate according to the principle of "heuristic classification" (see below).

## 13.2 Problem Types

In this section we attempt to map the great variety of examples on a few problem types from the point of view of the user.

- *Static fault-finding:* In a system the causes of observed, time-independent faults are sought, in order to derive suggestions for correction. O: observed symptoms and measurements. S: causes of faults and possibly suggestions for correcting them. Examples: Repair diagnostics of technical equipment, simplified process diagnostics, data base tuning, simplified medical diagnostics, therapy and laboratory data interpretation.

- *Dynamic fault-finding:* In a system the causes of observed, possibly time-dependent faults are sought, in order to derive suggestions for correction. O: objective symptoms and measurements, where in both cases values determined at different times may be present. S: defective components and/or possible correction of external influences. Examples: technical process diagnostics, complex medical diagnostics, therapy, monitoring and laboratory data interpretation.

- *Assessment:* The assessment of an object or a process according to a given norm. O: properties of the object or process. S: in principle only the two decision categories "positive" or "negative", the decision usually being made on the basis of weighted individual criteria. Examples: granting of loans (yes/no), feasibility of projects (yes/no), quality control (good/bad) and process monitoring (alarm/no alarm).

- *Multiple assessment:* The assessment of an object or process according to different given norms. O: properties of the object or process. S: definitions of different assessment criteria. Examples: Application of legal regulations to a case, selection of room plants for a given location, process monitoring with more than two alarm states.

- *Precedence selection:* Selection of an object on the basis of subjective wishes which are often unrealizable and must therefore be modified. O: wishes of the user. S: available products. Examples: investment advice, catalog selection, purchase of products.

- *Object identification:* The identification of a physical object as an element of a class of objects on the basis of observed characteristics. Here the solutions are naturally given object classes, in contrast to "multiple assessment", where they are usually much more precisely defined norms and definitions. O: observable properties of the object classes. S: object classes. Examples: identification of fungi, object and image recognition, recognition of geological formations.

The four classification problem types of Stefik and Hayes-Roth (Sect. 11.1) can be fitted in as follows: interpretation $\widehat{=}$ identification; monitoring $\widehat{=}$ assessment (simple or multiple); diagnostics and debugging $\widehat{=}$ fault-finding (static or dynamic). Admittedly, Stefik's and Hayes-Roth's "interpretation" also includes construction problems, as is shown by the cited example DENDRAL for determining molecule structure formulas, which we exclude from identification.

The value of the above classification is that it makes it easier to find a correspondence between problem types and domains. On the other hand, this grouping is only of limited help in the selection of problem-solving methods, since the latter depend more upon the observations and the nature of the available knowledge. Typical properties of classification problems are investigated below. They are sometimes, but not always, characteristic for certain problem types or can be excluded.

## 13.3 Analysis of Problem Characteristics

The simplest problem-solving method for classification would be a direct, unique correspondence between observations and solutions. This is of course rarely possible. The following difficulties may occur:

- *Uncertain knowledge:* The knowledge about the relationships between the observations and the diagnoses is uncertain, i.e., one observation may point to different solutions. Example: all domains.

- *Unreliable observations:* The derived solutions cannot be better than the reliability of the given observations. Three types of unreliable observations may be distinguished:

- *Uncertain observations:* The observations cannot be made with 100% certainty. Uncertain data on observations are much more difficult to formalize than uncertain knowledge, because the degree of uncertainty usually cannot be determined, and they may therefore constitute a barrier to the problem-solving ability which is hard to overcome. Examples: observations made with the human senses such as pains in medicine, sound quality in engine diagnostics, surface properties of fungi, etc.
- *Changeable (subjective) observations:* The observations are not objective facts, but user wishes, which may change during the problem-solving procedure. Example: precedence selection; in other problem types the observations are objective.
- *False observations:* The given observations do not agree with the actual facts. The attainable problem-solving ability can be impaired even more than in the case of uncertain observations. Since false observations can generally be detected only through contradictions, there should be as much redundance as possible in collecting the observation data, but this may decrease the acceptance of the programs. Examples: typical for assessment, possible in all domains.

- *Time-dependent observations* These observations are reliable, but may change with time. The changes often give valuable indications of the solution. Examples: dynamic fault-finding, process monitoring.

- *Incomplete observations:* At the start of the problem-solving process only a subset of the observations is given. If required, further observations can be collected, but this involves time and costs. As a consequence, the problem-solving method must foresee:

  - *Dialog control:* A cost/profit analysis must be made to find out whether additional observations should be requested, and if so which. Examples: all domains, except when all observations are collected in a standard procedure, as in the evaluation of experimental measurements.
  - *Belief revision:* In many cases (temporary) conclusions must be derived during the problem-solving process which may have to be revised when new facts become known. Examples: typical for fault-finding in dynamic systems, possible in all domains.

- *Abstraction of observations to solutions via intermediate steps:* The technical terminology often contains a rich vocabulary of concepts which are abstractions of the observations but not yet solutions. The two main types of such intermediate results are abstractions of observations and classes of solutions (see Sect. 11.2, Fig. 11.3). The use of intermediate results in the problem-solving method is almost always expedient. Examples: all domains; the more so, the greater the complexity of the domain.

- *Parametrization of observations and solutions:* Sometimes subsets of the observations and solutions consist of very similar elements which may easily be parametrized. Examples: faults in different cylinders in automobile engine diagnostics, faults in different machines of the same type in the process diagnostics of a production line.

- *Multiple solutions:* Sometimes there can be several solutions which are together responsible for the observations. Examples: multiple diagnoses in static or dynamic fault-finding, several applicable categories in multiple assessment; not possible in any other problem type (multiple solutions are not to be confused with competing solutions which have been given equal value).

- *Combined recommendations for several solutions:* Normally each solution has a recommendation for action. However, if multiple solutions are offered, or if less probable solutions are considered in addition to the most probable one, the recommendations should be adjusted to be compatible with one another. Example: fault-finding.

Some difficulties (e.g., the nature of uncertain knowledge) have a direct influence on the problem-solving method, while others (e.g., time-dependent characteristics) can be handled relatively well by means of additional mechanisms to a method. Such additional mechanisms will be considered in Chap. 16. The greatest difficulties are caused by uncertain observations, which, if they are very numerous, can cast doubt on the suitability of expert systems for the whole domain.

## 13.4 Problem-Solving Methods

The simplest procedure is to derive the solutions from the observations, possibly with step-by-step abstraction. This can be done with *forward chaining* of rules, in which all rules whose preconditions are fulfilled by the data set of given observations and derived intermediate results are eventually triggered. If the set of observations is incomplete, the user should be asked for the specific additional observations required. A suitable method for this is *backward chaining*, in which only those rules which are relevant to the actual goal are investigated and – if necessary – observations not yet included are requested. However, if no external hypothesis is given as a goal, all conceivable solutions must be checked in some given order (e.g., in order of decreasing *a priori probability*). This "blind search" can be improved by means of the *establish-refine strategy* for searching in hierarchical data structures: on a given level of the hierarchy a candidate is first established, and then its direct successors are investigated and the best of these is established, etc. A candidate can be established by direct enquiry in a decision tree or by backward chaining of many rules with the candidate as goal. The establish-refine strategy is efficient only if the knowledge can be given a hierarchical structure, and in addition it must be possible to establish a candidate

on every level, which is not always the case. The *hypothesize-and-test strategy* avoids this limitation by using forward chaining to find likely candidates and investigating the best of these with backward chaining. If several candidates have almost equal values, tests can be proposed which contribute most to differentiating between them. We thus have the following basic problem-solving methods for classification:

- forward chaining.
- backward chaining,
- establish-refine strategy,
- hypothesize-and-test strategy.

Forward and backward chaining are general inference strategies for evaluating rules. In heuristic classification these inference strategies are at the same time basic problem-solving methods, since they can be used as they are. The same can be said of the establish-refine strategy, which is also a very general technique of heuristic refinement. The hypothesize-and-test strategy is more specific for classification, and since it is very flexible, though simple, and can be used to simulate the other classification methods, it is the standard method for classification with uncertain knowledge. This was also confirmed for human problem-solving in psychological tests of medical decision-making by doctors [Elstein 78].

However, even the hypothesize-and-test strategy is general enough to be used for different types of knowledge. Like the other three methods it is therefore termed a "basic problem-solving method", which must be specialized to a strong problem-solving method by specifying a type of knowledge. Below we give a survey of the most important types of knowledge of classification.

If the knowledge is certain, *decision trees* or *decision tables* are suitable for knowledge representation. The former imply a good dialog control, but the latter are easier to change. Since each step is certain, not even the hypothesis-and-test strategy is needed, but a simple form of the establish-refine strategy is used for decision trees and forward or backward chaining for decision tables.

The simplest way to deal with uncertain knowledge is to qualify the rules with uncertainties, which are represented as numbers in the decision tables. However, this small change has important consequences, since a reckoning scheme for treating uncertainties is required and the best hypotheses should be followed with the hypothesize-and-test strategy. Further, numbers are a relatively weak way of representing uncertainties, so that many extensions become necessary. On account of its simple basic structure and flexible extendability by additional knowledge, this *heuristic classification* is by far the most successful method in classification expert systems up to the present.

Heuristic classification leads straight from observations to solutions, and that solution is selected which has accumulated the best evaluation. In (causal) *set-covering classification*, the knowledge representation works the other way round, i.e., from solutions to observations, and that solution is selected which explains

(i.e., covers) as many of the observations as possible. When set-covering classification is used for static or dynamic fault-finding, it is also called "classification with fault models", because the knowledge consists of a large fault model explaining how the diagnoses cause the symptoms, possibly via intermediate steps. An advantage of such fault models is that it is easier to recognize situations in which multiple diagnoses must be made, i.e., when no single diagnosis can explain all symptoms. A disadvantage is that hypothesis generation is difficult.

*Functional classification* distinguishes between the normal and faulty functioning of a system. For this reason functional models can be used in classification only for fault-finding in static and dynamic systems. A functional model describes the normally functioning system which does not evoke any symptoms. These are caused by faults (represented as changed behavior) in individual components, whose effect on the model is simulated. Usually the likely changes in behavior of the components are given as states (component fault models[2]). As in set-covering classification the predicted symptoms are compared with those observed, and the change which gives the best agreement is selected as the solution.

It is also possible to use statistical knowledge instead of heuristic, empirical knowledge, in which case the frequency with which an observation correlates with a solution is calculated from a case data base. The most important evaluation scheme for this *statistical classification* is Bayes' Theorem, although a number of conditions must be fulfilled for it to be applicable.

Another method for evaluating case data bases is *case-based classification.* With each case the set of observations and the corresponding solution are stored in the data base. For a new case, the case in the data base with the greatest similarity with regard to the observations is sought, and if the similarity is sufficient its solution is used. For this purpose some measure of similarity is needed, and additional knowledge generally required. Case-based classification has received much attention recently, since the knowledge acquisition can be achieved practically automatically when a case data base is built up. Figure 13.2 summarizes the main types of knowledge for classification and the requirements for each type.

The correspondence between basic problem-solving methods and knowledge types is given in Fig. 13.3.

The correspondence between problem types and knowledge types in Fig. 13.4 shows that the former impose hardly any restrictions on the choice of the type of knowledge. The only restriction is that classification with functional models is suitable only for fault-finding, since this is the only case where we have deviations of a system from its normal state.

The following description of problem-solving methods gives priority to the type of knowledge, and the hypothesize-and-test strategy is to be preferred for the more interesting knowledge types (heuristic, set-covering, functional and case-based classification). The problem-solving methods are therefore named after the knowledge type, which always includes one basic problem-solving method, however.

[2]These local fault models must be distinguished from the global fault models of set-covering classification.

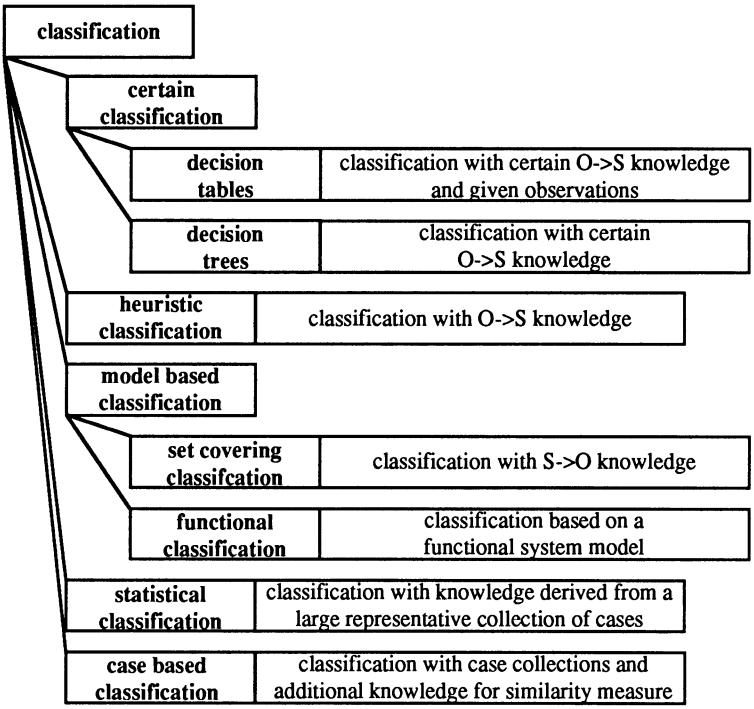



**Figure 13.2** Summary of knowledge types for classification

| | Tr | Ta | He | Co | Fu | St | CC |
|---|---|---|---|---|---|---|---|
| Forward chaining | | × | (×) | | | × | |
| Backward chaining | | (×) | (×) | | | | |
| Establish-refine | × | | (×) | (×) | (×) | | (×) |
| Hypothesize-and-test | | | × | × | × | | × |

**Figure 13.3** Suitability of problem-solving methods for different types of knowledge. Tr = decision trees, Ta = decision tables, He = heuristic classification, Co = set-covering classification, Fu = functional classification, St = statistical classification, CC = case-based classification. The brackets indicate less preference. Forward chaining is suitable for decision tables and statistical classification when all observations are given from the start. Establish-refine is applicable if the knowledge is structured in a strict hierarchy, which is the case for decision trees by definition. Otherwise the hypothesize-and-test strategy is to be preferred.

| | Tr | Ta | He | Co | Fu | St | CC |
|---|---|---|---|---|---|---|---|
| Static fault-finding | × | × | × | × | × | × | × |
| Dynamic fault-finding | × | × | × | × | × | × | × |
| Assessment | × | × | × | × | | × | × |
| Multiple assessment | × | × | × | × | | × | × |
| Precedence selection | × | × | × | × | | × | × |
| Object Identification | × | × | × | × | | × | × |

**Figure 13.4** Suitability of knowledge types for different problem types; abbreviations as in Fig. 13.3

# 14. Simple Classification

By simple classification we mean the evaluation of certain knowledge and certain data. The two main types are decision trees and decision tables.

## 14.1 Decision Trees

The internal nodes of a decision tree represent questions, the links alternative answers, and the "leaves" solutions (Fig. 14.1). The answer to a question determines the choice of the successor node.

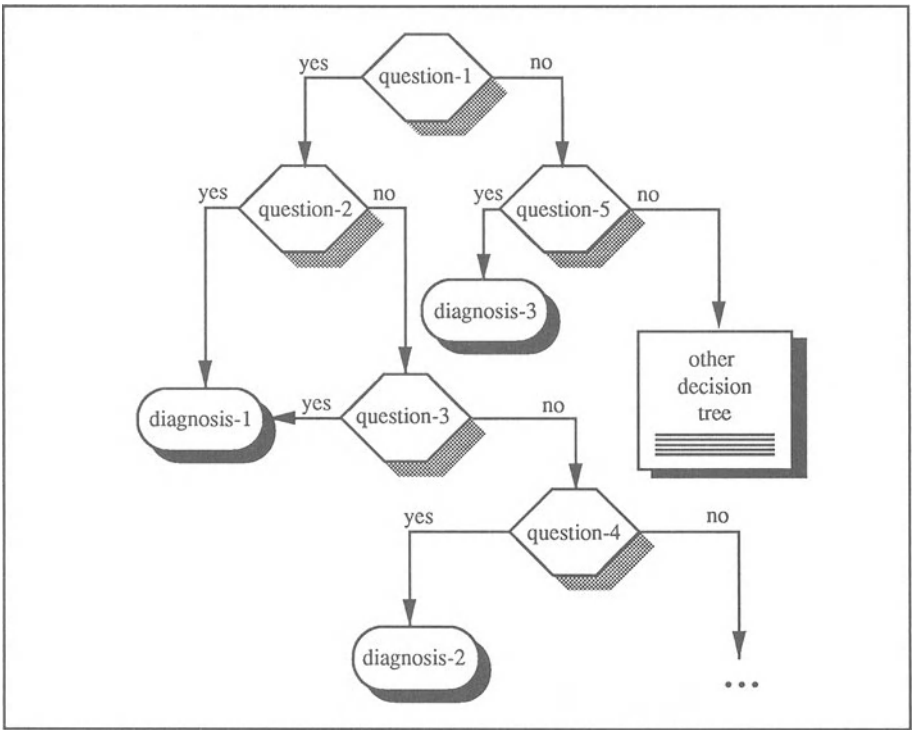



**Figure 14.1** Example of a decision tree

### 14.1.1 Knowledge Representation

The simplest representation would be a large, nested if-then-else instruction in a normal programming language. A structured form of representation has objects for the questions with attributes for text, explanations and value range and rules of the form "if question $x$ = answer $y$, then activate node $z$." A node may represent a further question, a solution, a contradiction or another decision tree.

### 14.1.2 Knowledge Manipulation

A decision tree is processed by alternately asking a question and evaluating the rules for determining the next question. When an end node has been reached, it represents a solution. For modularizing purposes a decision tree can be named and called by name. Plausibility checks can be achieved with redundant questions whose answers can lead to a contradiction.

### 14.1.3 Knowledge Acquisition

A simple form of knowledge acquisition is the input of questions as objects with the attributes name, question text, value range and possibly explanation and of attached procedures for the rules, e.g., in the form of branching instructions (answer 1 $\rightarrow$ question $x$, answer 2 $\rightarrow$ question $y$, answer 3 $\rightarrow$ question $z$). A graphical form of input would be to draw the decision tree on the screen: first a box is positioned, then the type (question, decision tree, solution, contradiction) is selected from a pop-up menu, and the links to other boxes are drawn and labeled with the alternative answers.

### 14.1.4 Discussion

Decision trees represent knowledge in a very concentrated form. They combine knowledge for dialog control and for the selection of solutions. The rules have only one explicit pre-condition, while all previously posed questions with their answers are implicit pre-conditions. Decision trees have the advantage that they are easily understood and – especially for small problems – compact. Their main disadvantages are that they are difficult to modify and may get very complex for large problems. The problem with modifying decision trees results from the fact that local changes may have global effects, since when one question is changed all succeeding nodes are affected.

## 14.2 Decision Tables

Decision tables can be regarded as a de-sequalized form of decision trees. Their rules are independent of one another and therefore do not give any dialog control.

Their knowledge representation and manipulation correspond to those of a commutative rule interpreter, where the rules represent implications (see Chap. 5). A graphical form of knowledge acquisition is to enter the rules in a table in which the rows contain questions with alternative answers, intermediate results and solutions. The pre-conditions of the rules, which must be combined with the "and" operator, are marked with crosses – or minus signs if they are negated (Fig. 14.2). The lowest cross (or some other symbol) in the column indicates the implication of the rule.

| **Decision table-1** | Rule-1 | Rule-2 | Rule-3 | Rule-4 | Rule-5 |
|---|---|---|---|---|---|
| Question-1 | × | × | × | – | – |
| Question-2 | × | – | – | | |
| Question-3 | | × | – | | |
| Question-4 | | | × | | |
| Question-5 | | | | × | – |
| Diagnosis-1 | × | × | | | |
| Diagnosis-2 | | | × | | |
| Diagnosis-3 | | | | × | |
| Decision table-2 | | | | | × |

× = yes
– = no
***empty*** = irrelevant

**Figure 14.2** Example of a decision table, corresponding to the decision tree of Fig. 14.1.

## 14.3 State of the Art

Decision trees and decision tables belong to the standard software engineering specification techniques [Fairley 85], which can be used also for many purposes other than classification. In expert system tools they are used, e.g., in the VAX Decision Expert.

# 15. Heuristic Classification

Heuristic classification is suitable for classification problems in which it is known from experience which observations – or combinations of observations – indicate intermediate or final solutions, and with what degree of certainty. Thus the basic object types are observations, solutions and rules of the form "observation indicates solution" ($O \rightarrow S$, with certainty $x$). That solution is selected which has the highest total score on the basis of the observations (Fig. 15.1).

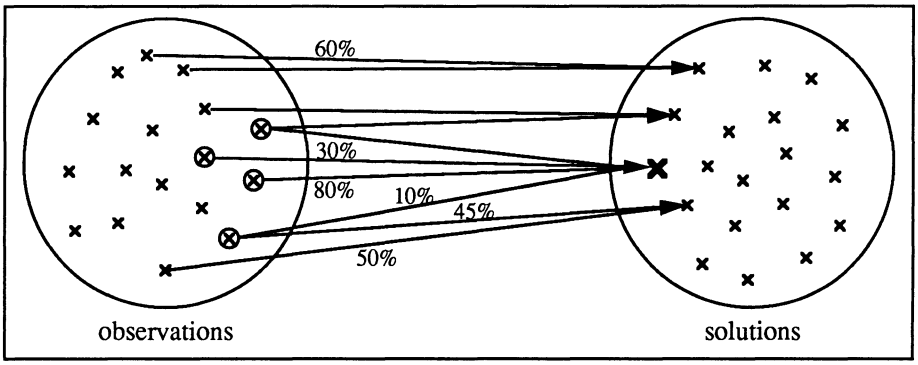



**Figure 15.1** Basic structure of heuristic classification

Heuristic classification differs from simple, or certain, classification in the use of uncertain knowledge, and from statistical classification (see Chap. 19) in that the certainty factors are estimated by experts and not calculated from a case data base. There is also a variety of reckoning schemes for evaluating these factors, on account of their different meanings. The certainty factors are also called evidences, scores or probabilities. There are two independent types of indications of the strength of the relationship between an observation and a solution: the *positive prediction value*, which indicates how strongly the presence of an observation supports the solution, and the *negative prediction value*, which indicates how strongly the absence of the observation speaks against the solution.[1] The

[1] In probability theory a further distinction is made between the positive prediction value ($P(S/O) \mathrel{\widehat{=}} O \rightarrow S$) and the specifity ($P(\neg O/\neg S) \mathrel{\widehat{=}} \neg S \rightarrow \neg O$) and between the negative prediction value ($P(\neg S/\neg O) \mathrel{\widehat{=}} \neg O \neg S$) and the sensitivity ($P(O/S) \mathrel{\widehat{=}} S \rightarrow O$). While heuristic classification uses the prediction value, statistical classification (Chap. 19) is based on specifity and sensitivity.

most important heuristic mechanisms for combining evidences are the simple addition of points with threshold comparison (INTERNIST scheme) and percentage reckoning (MYCIN scheme) (see Chap. 8).

Since heuristic classification is based on the experience of experts, and human beings find it difficult to keep track of many individual assessments for a solution, the technique of step-by-step abstraction from observations to solutions is necessary. In this process constellations of observations are combined to a single concept, which can be treated as one observation in assessing solutions. These concepts often correspond to terms from the special terminology of the domain. There are two types of abstraction (see Fig. 15.2): (a) the compression of raw data to simple data abstractions, which is usually based on concept definitions, i.e., on certain knowledge, and (b) the derivation of partial solutions or solution classes, which is based on uncertain knowledge. Examples of data abstraction are:

- Arithmetic calculations (e.g., annual milage of automobile = total milage divided by age of automobile).
- Abstractions from quantitative to qualitative values (e.g., in medicine: tachycardia = pulse rate over 90 per minute). The division of measured values into "too low", "normal" or "too high" is the basis of the classification in many domains.
- Combinations of single observations to local abstractions which do not yet have the status of global solutions (e.g., collecting together the various aspects of chest pains to classify them as "heart" or "lung" pains).

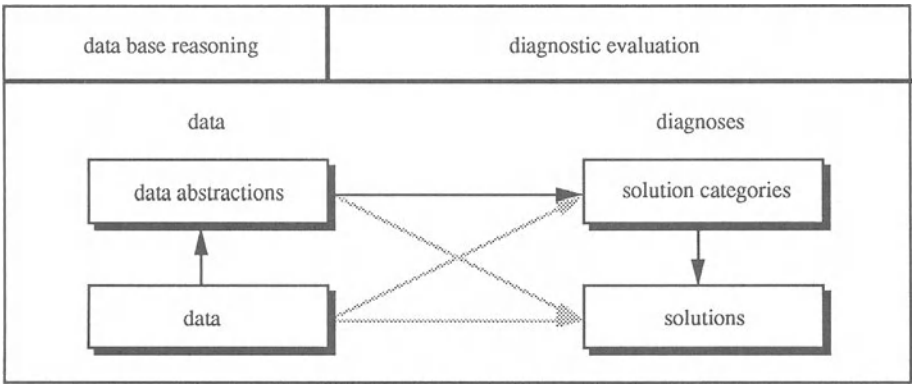



**Figure 15.2** Structure of step-by-step abstraction for diagnostics (cf. Fig. 11.3)

Partial solutions or solution classes are global, complex abstractions which are derived in the same way as the solutions but do not yet themselves constitute solutions, e.g.:

- "Infection" or "liver disease" in medicine, which describe a condition of the patient which must be further specified.

- "Fault in idling system" or "fault in the exhaust" in automobile engine diagnostics, which must also be refined.

The examples also show that in many domains there are several differently structured solution hierarchies, in which the classification principle may be fault localization (liver disease, exhaust), fault origin (infection) or faulty function (idling system) (see Fig. 15.3).

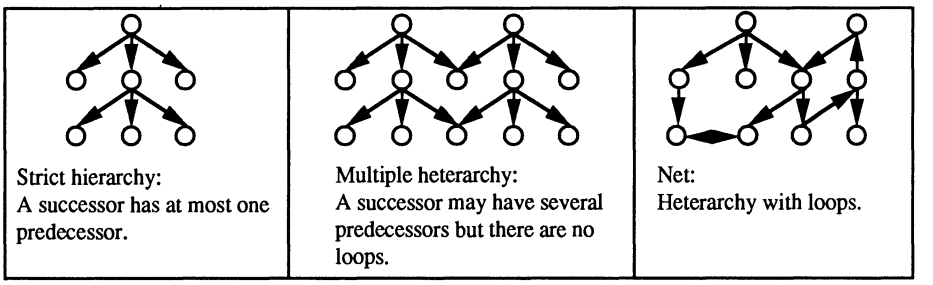



**Figure 15.3** Different types of solution hierarchies

If the derivation of a solution class is uncertain, one must consider how the uncertainties are to be propagated in the solution hierarchy. In this chapter we consider only the following approach: the certainty factor of a solution class is compared with a threshold value and with the certainty factors of competing solution classes to reach the decision solution class = "established", "uncertain" or "excluded". Only the established solution classes transfer evidence values to their successors. A less rigorous propagation of uncertainties through the hierarchy of solution classes is discussed in Sect. 16.1.

In domains in which all observations are given initially, the forward-chained processing of rules is a sufficiently good evaluating strategy. If the possibility of requesting additional data exists, however, the aim must be to achieve a maximum certainty with a minimum of additional questions. The most important questioning strategies are:

- *Backward chaining:* All rules are evaluated which can contribute to reaching the global goal. If some pre-conditions of the rules are unknown, they are derived recursively as sub-goals or requested from the user.

- *Establish-refine:* In a strict solution hierarchy a solution category is first established by backward chaining and then refined by attempting to establish a successor, which is itself refined, etc. This corresponds to the processing of decision trees, except that the internal nodes do not represent simple questions but solution classes which have been evaluated by backward chaining.

- *Hypothesize-and-test:* From the given observations hypotheses are generated by forward chaining and subsequently tested specifically by backward chaining. The cycle is repeated until, in a typical case, a hypothesis has been established (see Fig. 15.4).

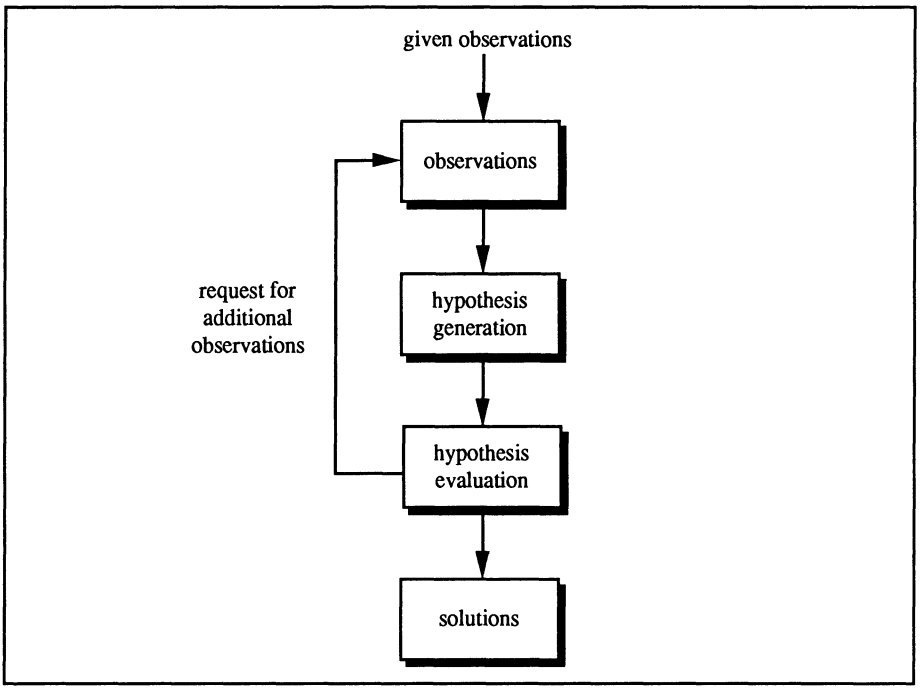



**Figure 15.4** Hypothesis-and-test strategy

The three dialog strategies differ in how they choose the next solution hypothesis to be tested: in backward chaining by exhaustive depth-first search, in establish-refine through a given solution hierarchy and in hypothesize-and-test by means of a separate hypothesis-generating phase in which the currently best hypothesis is determined. Once a hypothesis has been selected, it is completely investigated in all three strategies by processing all its rules and posing all the corresponding questions. In many domains, however, this is too complicated, since the procedures for testing the hypotheses can become very costly. It is therefore necessary to check whether it is worth testing a particular hypothesis, and if so for how long. For this purpose it is useful to attach tests to the hypotheses and to specify costs and profits for each test. A test contains a set of questions which are posed together like a small questionnaire and may be structured in a hierarchy. In each cycle of the hypothesize-and-test strategy one can then always choose and process the test which has the lowest costs for the most probable hypothesis.

In this survey we have sketched the mechanisms for treating uncertain knowledge, for step-by-step abstraction and for dialog control which will be explained in detail in the following sections. Figure 15.5 summarizes the typical assessment situation of a solution. Mechanisms for treating the remaining requirements of Sect. 13.3 are discussed in Chap. 16.

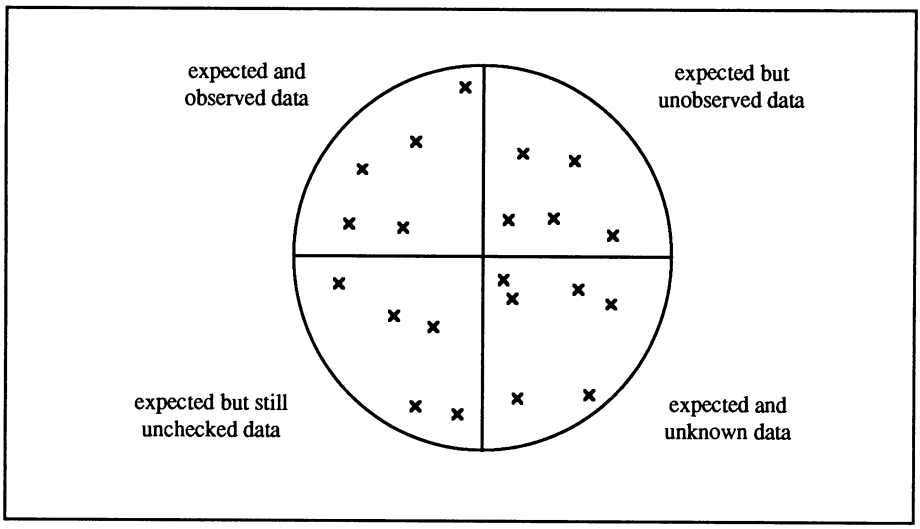



**Figure 15.5** Assessment situation in heuristic classification. The crosses represent observations expected for a given solution, which may be present (supporting the solution), absent (evidence against the solution), unknown (neutral) or not yet checked (requiring further investigation).

## 15.1 Knowledge Representation

The central elements of the knowledge representation are observations (questions), data abstractions, solution classes, solutions and tests for combining together observations and associated rules. Since data (observations) and data abstractions differ only slightly in their attribute structures, they are represented by the same object type. The same applies also to solutions and solution classes.[2]

1. Data and data abstractions differ only in their "derivation":
   - name
   - type (requested or derived)
   - value range (set of possible values)
   - meaning (s) (rules for deriving conclusions from the question)
   - derivation (s) (if type = requested: rules for their activation, if type = derived: rules for deriving the value)
   - tests (which use the data)
   - value (d).
2. Solution classes and solutions differ only in "successor" (none for solution) and "recommendations" (none for solution classes):

[2]Semi-static attributes, which can be derived from the normal (static) attributes by the knowledge acquisition component, are marked (s), and dynamic attributes (d).

- name
- recommendations
- *a priori* probabilities (general frequencies of the solutions)
- meaning (s) (rules for deriving conclusions from the solution)
- derivation (s) (rules for deriving the evidence of the solution)
- predecessor (s) (predecessor solutions, determined from "meaning")
- successor (s) (successor solutions, determined from "derivation")
- clarification (s) (tests for checking the solution)
- competitors (solutions with similar data)
- value (d) (calculated from *a priori* probabilities and derivation rules)
- status (d) (established, uncertain or excluded).

3. Tests

   - name
   - questions (list of observations to be checked in standard cases)
   - basic costs (general estimate of costs for performing the test)
   - costs (s) (rules for calculating the case-specific costs of the test)
   - profit (s) (use of clarifying hypotheses)
   - value (d) (costs-profit calculation)
   - status (d) (done, not done)

4. Rules

   - type (asking follow-up questions, test indication, solution assessment, derivation of data abstractions)
   - pre-condition (logical functions – with "and", "or" and "not" – of predicates referring to objects)
   - action (according to the type)

## 15.2 Knowledge Manipulation

The mode of operation of a problem-solving module for heuristic classification using the knowledge representation given in Sect. 15.1 is outlined below. In addition we introduce two new data structures: a solution agenda and a test agenda, in which the respective objects are arranged together with their current values.

The user starts a dialog by specifying the tests which are relevant to his problem and proceeds to answer the questions which are put to him. The answered questions send messages to all rules which are listed in their attribute "meaning". The rules react to the message by checking whether their pre-condition is fulfilled.

If so, they send a message to the object specified in the "action" attribute. If this object is a follow-up question, it is posed in the dialog. If it is a data abstraction, solution (class) or a question category its "value" is updated. While data abstractions can then send messages to the rules of their "meaning", the situation with solutions and tests is more complicated.

If a solution is suspected (as determined by a comparison of its total likelihood value with some threshold value) but not yet established, it sends a message to the solution agenda, where it is entered in the order of its likelihood. When the process of adding solutions to the agenda has been completed, the solutions with the best assessment on the agenda send messages to the test of their attribute "clarification". These tests react to the message by carrying out the profit-costs calculation to determine their "value" and sending a message to the test agenda. The test with the best overall value is determined and its questions are put to the user next in the dialog. Tests can also reach the agenda by being selected by the user as additional tests during or at the end of the dialog, or they may be activated directly by rules without any intermediate hypotheses. These tests are given priority.

If the assessment value of a solution rises above the threshold value, so that it becomes a candidate for being established, it sends a message to its "competitors", which report back their values. The candidate solution is established only if the difference in values is large enough. It then sends messages to all the rules of its "meaning", which thereupon check their pre-conditions as in the case of messages from questions. The process terminates either when there are no more tests in the test agenda or when some termination condition is fulfilled (e.g., the establishment of a genuine solution with no successors). The result then consists of the names and, if applicable, the "recommendations" of the established solutions. The most important steps can be summarized in the following algorithm:
Input: tests
Output: solutions with recommendations

1. TEST-AGENDA := input tests
2. *while* TEST-AGENDA not empty *repeat*
   - 2.1 Select the best-valued TEST not yet carried out ("status" = not done) from TEST-AGENDA, delete it from TEST-AGENDA and set its "status" = done.
   - 2.2 Process the individual data of TEST. These activate their rules (in "meaning") and can cause the derivation of data abstractions (which then activate their rules), the addition of test to TEST-AGENDA or the assessment of solutions or solution classes. The latter are included in SOLUTION-AGENDA.
   - 2.3 Evaluate the solutions (and solution classes) of the SOLUTION-AGENDA. Established solutions activate the rules of their "meaning" and are transferred from the SOLUTION-AGENDA to the RESULTS-LIST. Suspected but not established solutions transfer the tests from

their "clarification" to the TEST-AGENDA. Poorly assessed solutions are removed from the SOLUTION-AGENDA.

3. Output of the solutions on the RESULTS-LIST with their recommendations.

So far we have not made use of the possibility of inheritance of properties along the predecessor and successor relationships. Predecessors and successors are used for the ordered output of solutions and solution classes, in the explanation and in knowledge acquisition.

## 15.3 Knowledge Acquisition

On the basis of the scheme of heuristic classification given in Figs. 15.1 and 15.2, there are two complementary strategies: solution-oriented or data-oriented knowledge acquisition (see Fig. 15.6).

| 1. | Solution-oriented strategy | 2. | Data-oriented strategy |
|---|---|---|---|
| I. | Gather and structure solutions | I. | Gather and structure data |
| II. | Define solution profiles | II. | Add data abstractions |
| III. | Break up data abstractions | III. | Gather and structure solutions |
| IV. | Collect and structure data | IV. | Define solution profiles |
| V. | Add further knowledge (for evidence values, recommendations, dialog guidance, etc.) | | |

**Figure 15.6** Knowledge acquisition methods for heuristic classification

In the solution-oriented procedure the first step is to collect together all solutions which the system is to recognize and arrange them in a hierarchical structure. For each of these solutions a solution profile is drawn up, consisting of a list of data or data abstractions favoring the solution and a list of those speaking against it. Then the data abstractions are broken up into inquirable data, the data collected into groups and the questions structured hierarchically. The solution-oriented strategy is suitable above all for the rapid development of prototypes. Its disadvantage is that later modifications are difficult, since the data collected in this way is suitable only for deriving the solutions considered at the beginning. If additional solutions are added or if the domain is extended, it is generally necessary to change the data structure and the rules which are based on it.

This disadvantage can be avoided by using the data-oriented strategy. A necessary condition is that one already has a fairly good general view of the domain. All data which are known to be important are systematically collected and structured. For the derivation of solutions, profiles are drawn up on the basis of the data and data abstractions already collected, so that further solutions can be added without having to restructure the existing knowledge base.

Similar alternative procedures are used for the dialog control: starting from the tests and considering under what conditions they are indicated, or starting from the solutions and deciding which tests are needed to check them.

The second phase of knowledge acquisition is the formalization of the knowledge in the concrete knowledge representation. A graphical knowledge acquisition system is a suitable support for this phase. The procedure may be outlined on the basis of the knowledge representation described in Sect. 15.1.

First the expert enters names for the test, data, data abstractions, solutions and solution classes which he uses in a hierarchy. The hierarchical structure helps the expert keep track of the knowledge base. Then the expert enters local object knowledge for the names by filling in the attributes of the object types. Since the system knows the attributes required for each object type, pre-defined input forms can be used to assist the expert in this process. When all the concepts of the domain have been specified in this way, their relationships are entered. This is done mainly with two types of tables. If the pre-conditions of the rules have on the average only one proposition, tables are suitable in which the rows are the pre-conditions (e.g., data), the columns the actions (e.g., solutions) and the entries the assessments (e.g., evidence values) (see Fig. 12.2). Apart from the data-solutions tables this type is also suitable for the indication of tests (in the columns) by means of hypotheses and data (in the rows). If rules with several propositions occur, a separate rule input form must be used. If the majority of the rules have complex pre-conditions, however, this procedure would be complicated, and it is preferable to use the other type of table (see Fig. 12.3), which is suitable only for the evaluation of an object. In this case a rule is represented not by a single box but by a whole column, which makes it possible to specify combinations of propositions for each rules by means of corresponding entries. The evaluations of the rules is given at the head or foot of each column.

In order to keep the tables easy to read they should be as small as possible, which means above all a restriction of the number of rows. Typically a row contains an observation with its value range. If the value range is numerical and has not been reduced to a data abstraction, the intervals which are relevant for the assessment must be specified. An important role is played by the negations of a proposition, since it is used in heuristic classification for determining the negative prediction value of an observation. In order to avoid having to double the number of rows, it is suggested that the negation be built into the structure of the table. In the first type (one-observation-one-solution, Fig. 12.2) two columns may be reserved for each solution, one for the positive and one for the negative prediction values of an observation. In the second table type (Fig. 12.3) two types of crosses can be used, one indicating an observation and the other its negation.

In principle the tables are suitable for the input of all object types with the exception of arithmetic formulas for data abstractions, for which a formula editor (see Fig. 12.7) is used. As an alternative, rule input forms may also be provided. In order to reference objects when configuring a table or filling out a rule formula, they are selected from the concept hierarchies which were built at the start of the process. The values of an observation which are relevant for rule formulation can

be selected from its value range, whereas the intervals of numerical data must be entered explicitly. This can be avoided if numerical data are converted to data abstractions with qualitative value ranges.

## 15.4 State of the Art

Heuristic classification is one of the most highly developed problem-solving types, as is reflected in many industrial applications. Even the majority of the first expert systems in the mid 1970s such as MYCIN, PIP, EXPERT or INTERNIST solved heuristic classification problems. EMYCIN [van Melle 81], EXPERT [Weiss 84], MDX [Chandrasekaran 83] and MED1 [Puppe 83] were early expert system tools which used forward chaining, backward chaining, establish-refine and hypothesize-and-test, respectively, as problem-solving methods. The diagnostic intermediate structure (see Fig. 15.2) was recognized by Clancey [84] as being characteristic for heuristic classification (see Sect. 11.2). The technique of dialog control with tests was introduced in the classification shell MED2 [Puppe 87a,b]. Like the first expert systems, the first graphical knowledge acquisition tools such as ETS [Boose 84] and AQUINAS [Boose 87] were also developed for heuristic classification. In recent years the first shells for heuristic classification such as TESTBENCH and a tool from Coherent Thought [ESS 88] have become available commercially. The performance which can be attained is demonstrated by the graphical knowledge acquisition system CLASSIKA (see Sect. 15.5), with which even complex knowledge bases for heuristic classification can be developed and maintained by experts with little or no help.

## 15.5 Example: CLASSIKA

Our experience with the heuristic classification tools MED1 and MED2 has shown that the basic structure of heuristic classification, as outlined above, with abstraction, hypothesize-and-test strategy, a coarse scheme for diagnosis assessment and a few additional mechanisms (see Chap. 16) has proved successful, but the knowledge acquisition component of MED2, which operates directly on the object-attribute structure of the knowledge representation, is hardly suitable for experts. We therefore developed the new graphical knowledge acquisition system CLASSIKA [Gappa 89, Puppe 92], to make it possible for experts to construct knowledge bases on their own. The main steps in the construction of a knowledge base with CLASSIKA are illustrated below using the knowledge base MODEX [Walser 91] for fault-finding in automobile motors as an example. The first and most important step is the definition of the terminology, i.e., the names for the symptoms and diagnoses. For this purpose CLASSIKA provides two hierarchies (see Figs. 15.7 and 15.8). The hierarchies also allow an object to have more than one predecessor and are therefore tangled hierarchies (see Fig. 15.3).

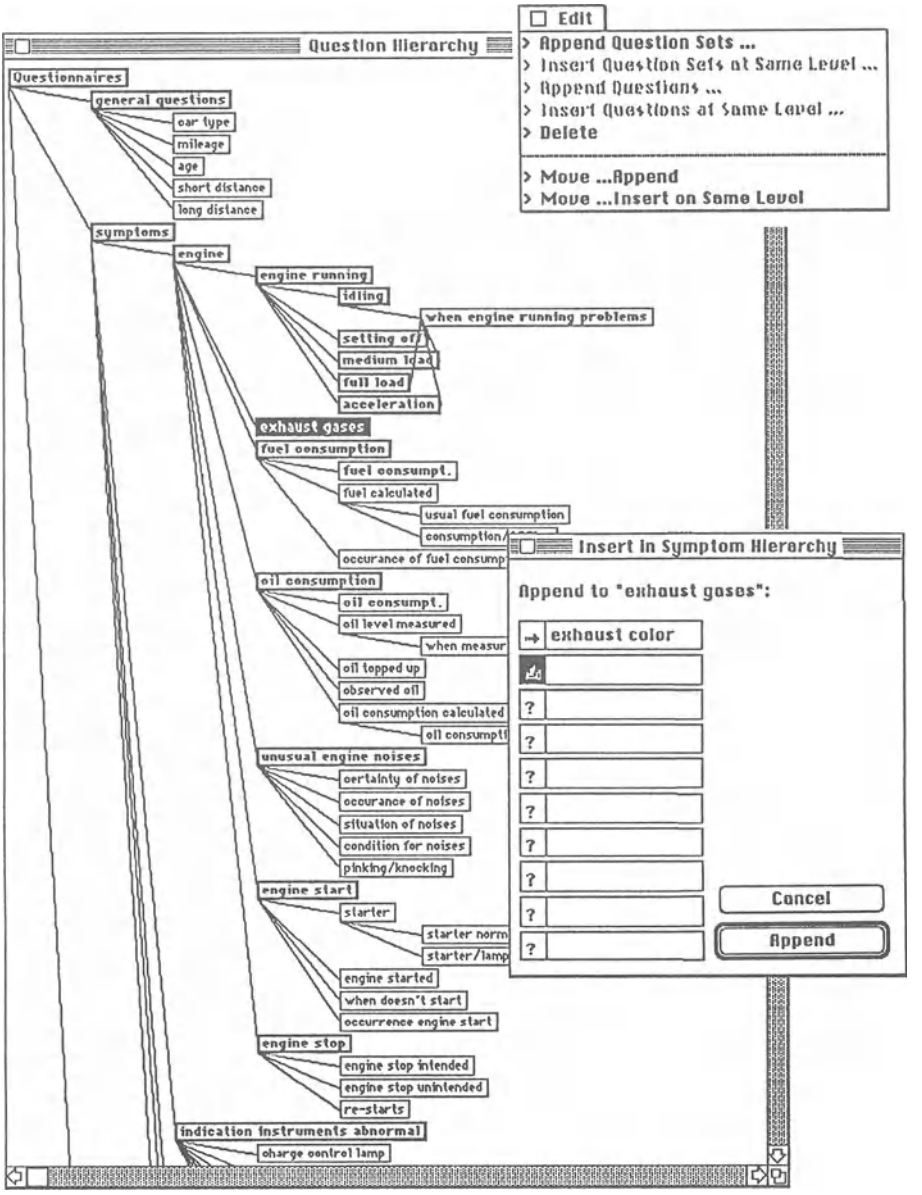



**Figure 15.7** Part of the symptom hierarchy of CLASSIKA. New concepts are entered by first selecting an object (exhaust) and then choosing an option "Append question classes" or "Append questions", etc. from the menu "Process", whereupon the insert window appears in which the new concepts may be entered. If the user closes the insert window with "Append", the new concepts are appended to the selected object in the hierarchy.

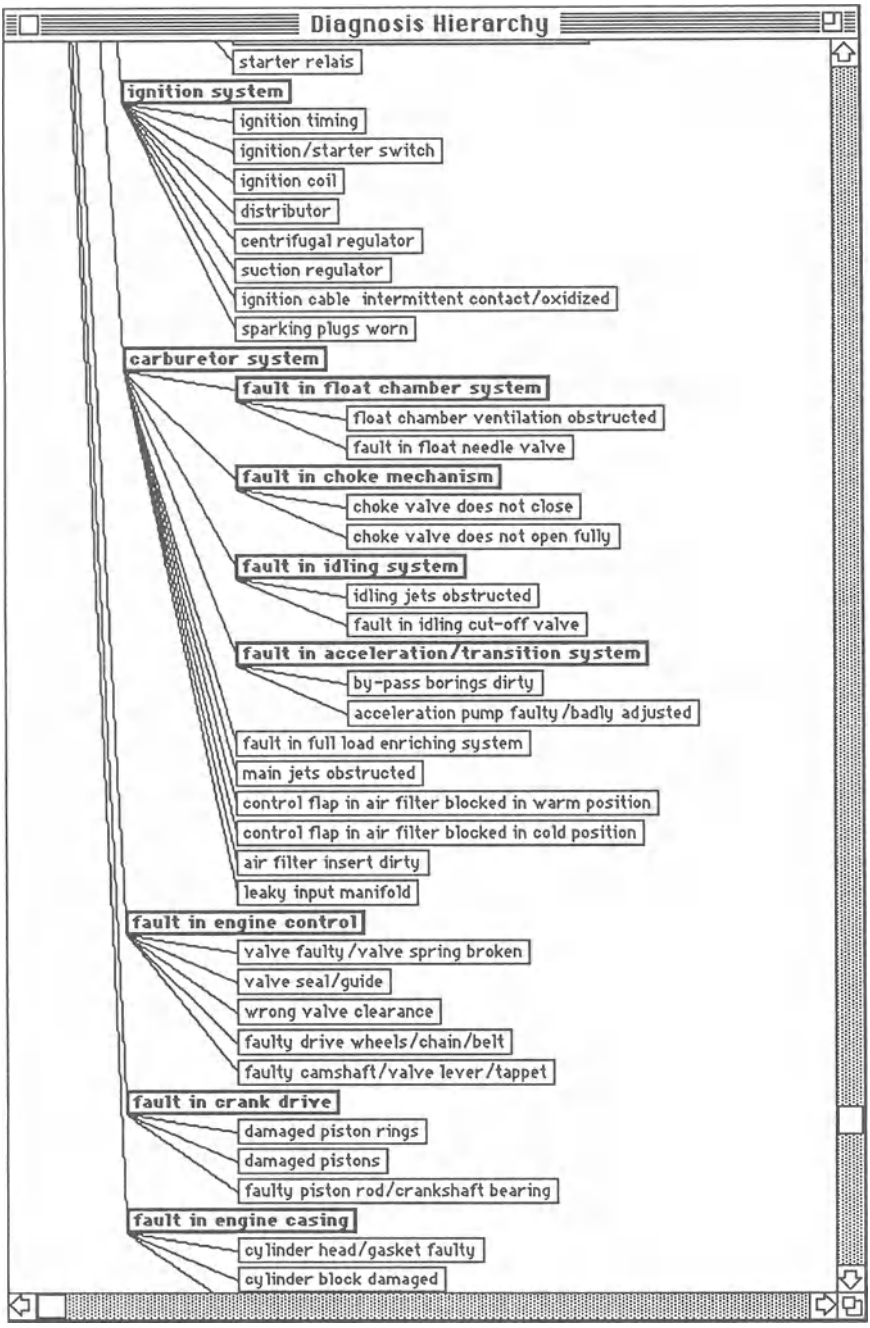



**Figure 15.8** Part of the diagnosis hierarchy, which has a structure equivalent to that of the symptom hierarchy

The next step is to enter local information for each object, i.e., to fill in the attribute structure. For this purpose the user must select one or more objects and then click on the item "form" in a menu, whereupon CLASSIKA displays a specific form according to the object type. The structure of this form indicates what values are expected for the attributes (Fig. 15.9).

Finally, the relational knowledge must be entered. In CLASSIKA this can be done with mouse clicks without using the keyboard. To enter rules between objects of a hierarchy, e.g., rules for follow-up questions or the derivation of data abstractions, the user double-clicks on the corresponding lines in the hierarchies with the mouse. This causes a rule form to appear (for many rules a table can be selected), containing the objects joined by the selected line in the pre-condition and the action, respectively, in which the user specifies the rest of the rule. Additional objects for complex rules can be entered in the form simply by selecting the objects in the hierarchy. The predicates and values of the pre-conditions can be chosen from local menus. Figure 15.10 shows the formula editor for rules with arithmetical calculations in the action part.

Most rules serve to derive diagnoses. To enter them, CLASSIKA provides two types of tables: one for deriving several diagnoses with simple rules, and the other for a single diagnosis with complex rules. First of all the user must enter the names of the rows and columns of the table by selecting the corresponding objects from the hierarchy. Then the table appears on the screen. In the first type of table the user enters a rule by first selecting a box and then choosing a probability class from the pop-up menu which appears (Fig. 15.11). In the second table type a rule is represented not by a box but by a column, and it is also possible to specify symptom combinations, contexts and exceptions to the rule (Fig. 15.12). In the first type of table complex rules are marked with "+" or "*" in front of the entry. In both types, the row headings may also be derived objects, e.g., diagnoses or objects with numerical values, in which case the relevant intervals must also be specified.

There are more tables for the evaluation of other objects, e.g., the selection of tests, data abstractions, etc.

The expert system so constructed can be used immediately. Figure 15.13 shows part of a dialog, in which the questions of the "exhaust" test are being answered, and an explanation of the established final diagnosis "ignition timing", which indicates amongst other things the triggered rules of the diagnosis.

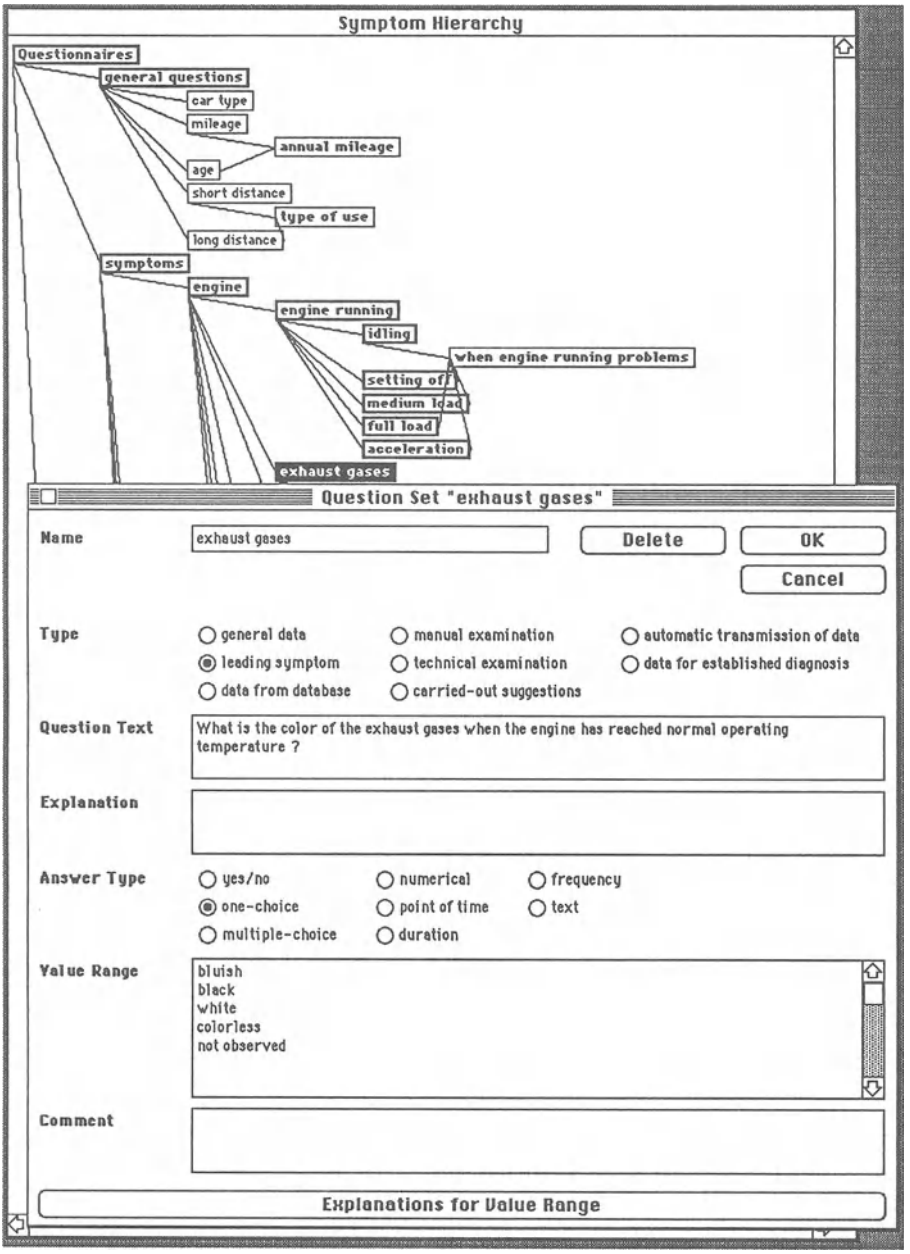



**Figure 15.9** Partly filled form for the question class "exhaust". There is a form for each object type (see Sect. 15.1)

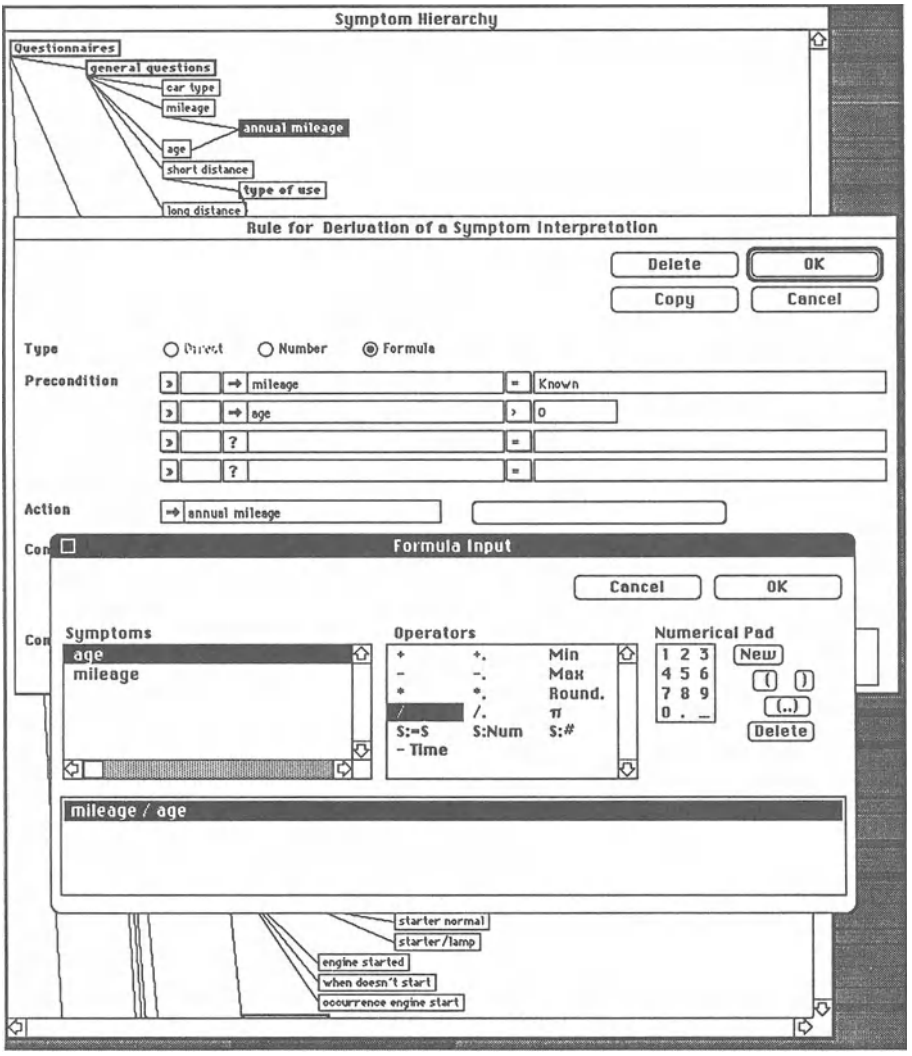



**Figure 15.10** Rule form with formula editor

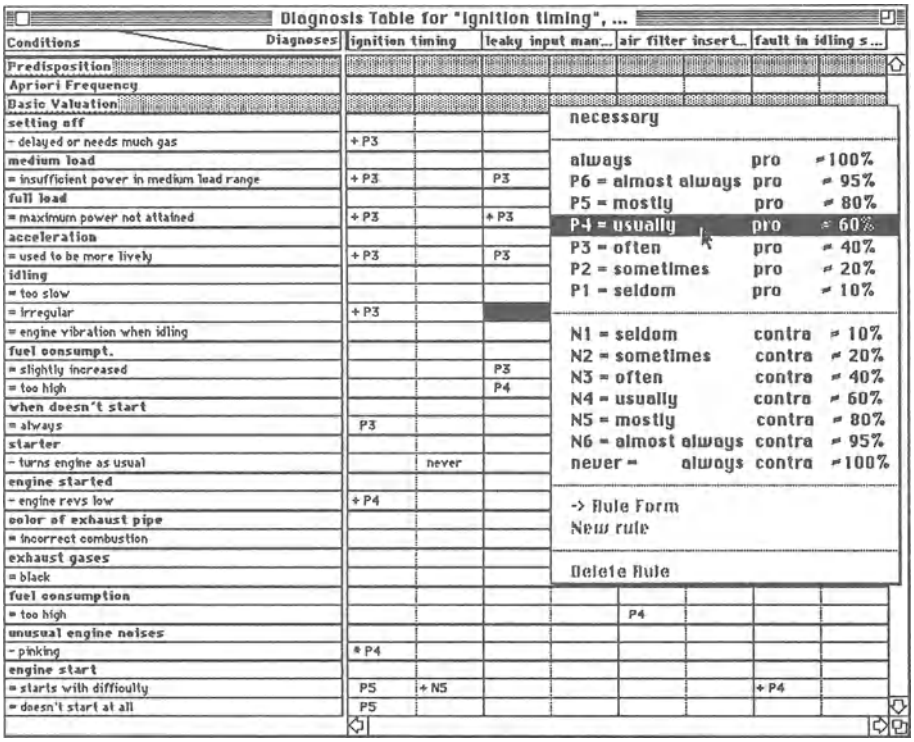



**Figure 15.11** Entering simple diagnostic rules in a table. In the table shown the rule "uneven idling indicates with a medium positive probability (namely 'p4') a leak in the intake manifold" is being entered with a pop-up menu. In the table two columns are reserved for each diagnosis. The first entry refers to the presence and the second to the absence of a symptom. The latter can supply important evidence for the exclusion of a diagnosis. If rules are more complex than can be represented in the table or if a table element conceals several rules, this is indicated by a "+" or a "*" in the corresponding box, and the rule can be edited by a double click in this box.

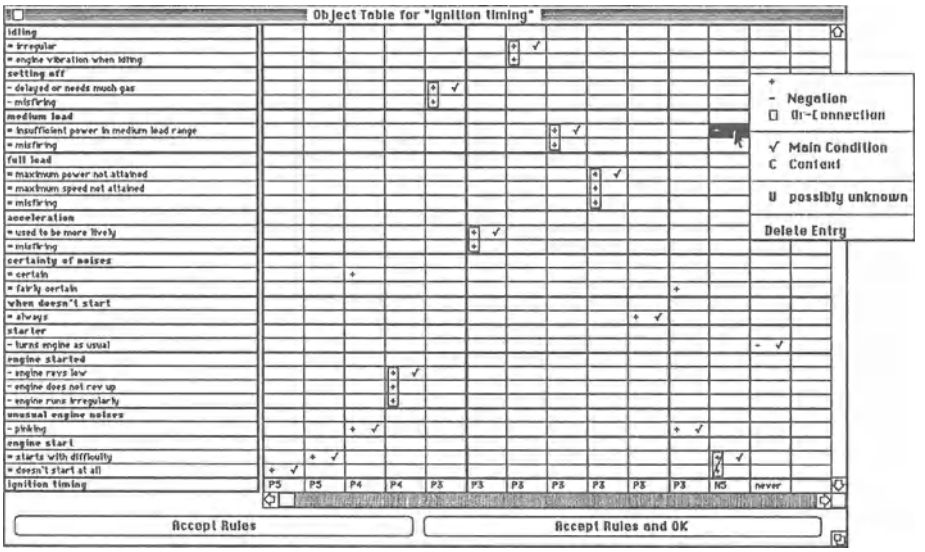



**Figure 15.12** Entering complex rules for a diagnosis in tables. The table shows the complete rule structure for the first diagnosis of Fig. 15.11 "ignition timing". Each column represents a rule, whereby the individual pre-conditions are connected by "and", and the evaluation of the rule is in the top row. A "+" in a field means that the symptom in the row is present, and a "–" that it is absent. A box round several entries means that they are connected by "or". Further, the propositions of the pre-condition can be divided into a main condition (marked with a tick), normal conditions (unmarked), context conditions (C) and exceptions (E). The main condition is included in the table type of Fig. 15.11 as the representative of the corresponding rule. The options are entered through the pop-up menu (top right), which appears when a box is selected.

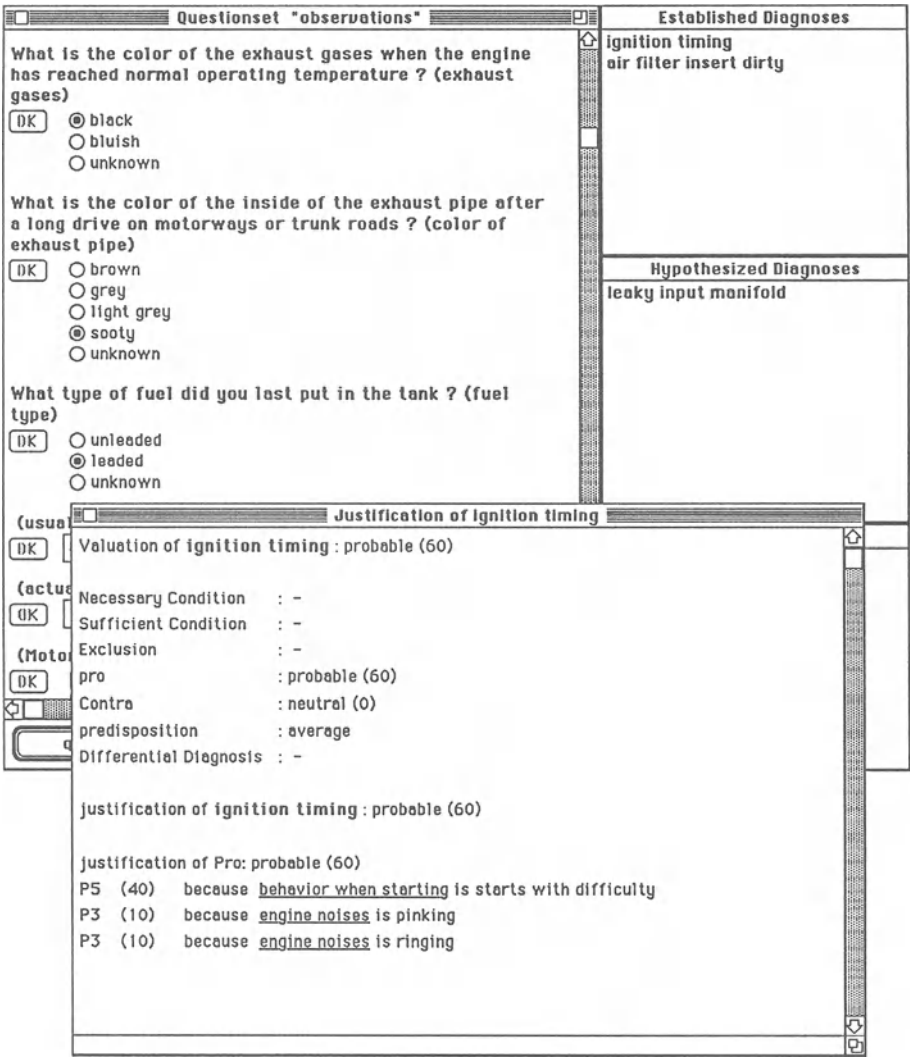



**Figure 15.13** Dialog situation from MED2, and below the explanation of the established final diagnosis "ignition timing"

# 16. Heuristic Classification: Additional Mechanisms

In the previous chapter we discussed how to deal with the central requirements of heuristic classification, namely the treatment of uncertain knowledge, step-wise abstraction and dialog control. In this chapter the mechanisms for the remaining requirements of Sect. 13.3 are described: the treatment of uncertain, subjective, potentially false, incomplete and parametrizable knowledge, of multiple solutions and combined recommendations for several solutions. The basic structure developed in Chap. 15 will be assumed.

## 16.1 Treatment of Uncertain Data and Solution Classes

### 16.1.1 Uncertain Data

In many applications, not only are the relationships between the observations and the solutions uncertain, but also the existence of the observations. With the knowledge representation described so far, it would be necessary to formulate explicit questions concerning the uncertainty of observations and corresponding rules for evaluating the answers, e.g., "How certain are you that observation $x$ is correct?", where the value range consists either of concepts such as "certain", "fairly certain", "uncertain", etc., or of percentages. If such uncertainties occur frequently, the number of additional questions and rules needed may become prohibitively large. As an alternative, one could provide such data as a standard in the knowledge base by giving each observation an implicit evidence value. In the dialog this would have the effect that the user could specify evidence values for all his inputs (either qualitatively from a pop-up menu or quantitatively), the default value being "certain" (100%). When evaluating the rules two kinds of uncertainty would then have to be combined: the uncertainty of the input, which comes from the user, and the uncertainties in the evaluation of the observations, estimated by the expert.[1] The standard reckoning procedure would be to multiply the evidences

[1] Since it is hardly possible to normalize human-estimated uncertainties and any two persons generally have different norms, no great accuracy may be expected from the calculations. One can even go so far as to dispense with the evidence estimates for the observations on account of the calculation difficulties. Instead, the expert could take into account typical uncertainties in the observations when determining the evidence values for the rules.

of the pre-condition (observations) with those of the action (evaluation). If several observations are combined through the "and" operator in the pre-condition of the rule, the individual evidence values are multiplied. In the case of the "or" operator they are treated in the same way as the combination of evidences from different rules. Another way of providing for the interdependence of the observations in a rule pre-condition, which was used in MYCIN (see Sect. 8.4), is to take the minimum for the "and" operator or the maximum for the "or" operator.

### 16.1.2 Propagation of Uncertainties of Solution Classes

A problem which is handled with similar mechanisms, but is different from the semantic point of view, is the treatment of the uncertainties of solution classes. In Chap. 15 we assumed that a solution class has only the three likelihood states "established", "uncertain" and "excluded", where "excluded" is the negation of "established" and nothing may be deduced in the "uncertain" case. The likelihood state was determined by comparison with threshold values and the values of competing solution classes. Instead of using the likelihood states, the further calculations can be performed with the primary uncertainties of the solution classes derived from rules, similarly to the use of user-estimated observation uncertainties in Sect.16.1.1. The problem of differences in the assessment standards between experts and users does not occur here, since the solution assessments are not estimated, but calculated from rules. On the other hand, there is an efficiency problem, since when new observations are entered the evidence values of solution classes must be changed by triggering the corresponding rules, and these changes must be transmitted to all direct and indirect successors of the solution classes. A compromise between many numerical evaluations and only a single likelihood state "established" would be to have a few additional classes (e.g., "probable", "highly probable" and "certain" as gradations of "established"). Then a small change in the basic evaluation which does change the likelihood state would not have to be propagated.

Another problem in the propagation of the uncertainties of solution classes is the danger of over-estimating the evidences if justification loops of the form "A increases the probability of B, and B increases – possibly via intermediate steps – the probability of A" are used. In two-element loops (A→B→A) the circular and non-circular parts of the evidences of the two solution classes concerned can be calculated relatively easily (see [Pearl 85, 88]), but the complexity of such calculations increases drastically as the number of elements increases. An example of the propagation of the uncertainties of solution classes can be found in PROSPECTOR; it is based on the idea of Bayesian networks (see Sect. 19.5).

### 16.1.3 Following Different Alternatives Simultaneously

An alternative to the above mechanisms is to work with certain observations and/or solutions and to represent the uncertainties by following the two alternatives (that a particular feature or solution class exists or does not exist) as

two assumptions parallel to one another and comparing the final results. The consequence for the knowledge representation is that all dynamic attributes of objects and rules can take several values for the different assumptions instead of just one. Suppose, for example, that two alternatives for an observation are to be compared, and let the values for the assumptions $A1$ and $A2$ be $x$ and $y$, respectively, then "$(A1xA2y)$" would be stored as the attribute "value", instead of just "x" as before. The rules for deriving conclusions from the observations also have a correspondingly extended attribute "value", e.g., ($A1$ triggered, $A2$ not-triggered). The same is true for tests, data abstractions, etc. Only for those objects which are completely unaffected is it sufficient to have a single entry, which is then independent of any assumptions. If it is required to branch again at another object, e.g., the two alternatives "established" and "uncertain" for a solution class, two new assumptions are introduced ($B1$ established $B2$ uncertain) whose consequences are followed like those of the first pair. If the different assumptions meet at a single object, all combinations must be considered ($A1B1$, $A1B2$, $A2B1$, $A2B2$). The same is true of course also for the two agendas for hypotheses and tests, and for any other global variables of the program. In addition, a new global data structure for the generation and global management of assumptions is needed. The dialog control must also be determined, e.g., by specifying a dominant assumption or combination of assumptions for the selection of tests and follow-up questions.

For following different assumptions simultaneously the ATMS technique ([de Kleer 86], see also Sect. 9.2) is suitable. Realisations can be found in several general expert system tools such as KEE and ART under the name multiple worlds or views.

## 16.2 Treatment of Subjective Observations

In the treatment of subjective observations, which represent preferences of the user, the problem is not just to find either a solution which fulfills all wishes or no solution at all. If there is no such solution, the user should be told which partial solutions come nearest to it, and which preferences would not be fulfilled. For this purpose two different assessments of a solution are needed: (1) whether the solution satisfies all the wishes of the user and (2) how attractive the solution is. The first assessment is categorical, but for the second the satisfied and unsatisfied preferences must be weighted according to their significance and combined with general considerations such as the cost of a solution. If no solution satisfies all preferences, the user can ask which wishes would have to be abandoned for the most attractive solutions. Case-comparing classification (see Chap. 20) is also very suitable for this precedence selection.

## 16.3 Recognition of False Observations

The treatment of uncertain observations was discussed in Sect. 16.1. This section is concerned with the detection of obvious errors in the input data. Such errors can arise through simple operating mistakes, inattentive observations, intentional input of false data or defective measuring instruments. In general, errors can be detected by means of redundant data input. Priorities must therefore be set when constructing the knowledge base: in order to avoid errors due to lack of attention the amount of data input should be kept to a minimum, whereas the recognition of intentionally false data or instrument failure requires as much redundancy as possible.

In order to avoid the effects of fatigue, as few questions as possible, with as few alternative answers as possible, should be asked. The number of questions can be reduced by arranging the questions in a hierarchy in which a follow-up question is put only when all pre-conditions have been fulfilled. The number of alternative answers per question can be reduced by means of constraints, such that under certain conditions some alternatives are eliminated or pre-selected. In the case of numerical data also, constraints can be used to restrict the value intervals as far as possible. Such constraints can be entered by means of a table of the type shown in Fig. 12.3 for combinations of observations.

The same constraints are also suitable for the second case, i.e., for the subsequent detection of false input data. Additional questions with their alternative answers are put, and contradictions are registered with the constraint network. Then the user can be asked to resolve the contradictions by correcting at least one of the answers concerned. This is of course not possible when the contradictory input data consists of automatically registered measurements. In this case it is advisable to have a diagnostic system for detecting faulty instruments by comparison of measurements, and to use the results to suppress the evaluation of the data from these instruments. The basic rule scheme could have the following form: a measurement implies some conclusion, *except* when the instrument is faulty. The use of exceptions has the advantage that it is normally assumed that the instrument is working correctly, as long as there is no explicit evidence to the contrary. However, this means that some mechanism for revising conclusions (see Sect. 16.5) is required. An example for the implementation of these techniques can be found in MED2 [Puppe 87, Sect. 4.2.2.2].

## 16.4 Treatment of Time-Dependent Data

In order to handle time-dependent observations the knowledge representation must be extended in such a way that temporal data can be stored, and rule predicates for evaluating the temporal data must be provided. On the knowledge representation side questions asking at what time (or in which interval of time) an observation was made are needed. Since the value of an observation may change with time, the history (i.e., the sequence of values) must also be stored.

An attribute "history" could have the following structure: time-1 value-1, time-2 value-2, etc., i.e., at time-$x$ value-$x$ was observed. These data could be collected in the dialog by registering the time in each session and storing the value entered at the last session together with the time of that session to the attribute "history". The evaluation of temporal data is based on the one hand on predicates such as "before", "after" or "simultaneous with", which refer to two times, and on the other hand on predicates such as "increasing", "decreasing" or "constant", which refer to the history of an observation. Other predicates may describe specific aspects of temporal behavior which are important for the domain, such as an increase in the rate of change of a value or a characteristic body temperature curve. A detailed discussion of temporal reasoning is given in Chap. 10.

## 16.5 Belief Revision

If one cannot wait until all data have been entered to derive solutions, it may happen that conclusions become invalid in the following situations:

- The user changes some input data for one of the following reasons: (1) a mistake has been made, (2) the user's wishes have changed, (3) contradictions must be resolved, (4) time-dependent data have changed.
- New observations change the evidence values of solution classes from which conclusions have already been derived.
- Exceptions to rules which have already been triggered become known, so that the rules must be withdrawn.

In addition to the change in the status of the object directly affected, a chain reaction of consequences may result. The aim of all revision techniques is then to produce the state which would have arisen if the changes had been taken into account from the beginning of the problem-solving process. An efficient technique is to have the affected objects send a message to all their rules, which thereupon check their status. If a triggered rule has to be withdrawn, it sends a message to the object of its action to subtract the evidence value transferred by the rule. These objects then check whether the subtraction leads to a change of status, in which case they in turn send corresponding messages to their rules, and so forth. In this technique the withdrawal of rules is treated like normal "firing" with the opposite sign for the evidence. However, justification loops (see Sect. 16.1.2) require special treatment in order to avoid the multiple activation of changed objects.

## 16.6 Treatment of Parametrized Data and Solutions

So far we have assumed that rules refer to concrete objects, i.e., the pre-condition of a rule consists of propositions about the values of certain data, and the action

changes the status of an object. Unlike first-order predicate logic there is no "for-all" quantor which refers to sets of objects. However, a necessary condition for classification is that the sets of observations and solutions must be finite. In theory, therefore, any quantified proposition can be expressed as a set of non-quantified propositions about individual objects (see Chap. 12).

The need for quantified rules arises in classification when there are many similar observations and/or solutions, e.g., for an automobile with many cylinders, a robot with many similar joints and motors or a production line with several identical machines. A typical quantified rule would be, e.g., "For all $x$ ($x$ is a cylinder): if (compression in $x$ too low) then ($x$ probably defective)." In such cases we say that the observations and solutions are parametrized. In the case of twelve cylinders there would be twelve instances of the observation "compression", the solution "cylinder defective" and the rule. The advantage of the quantified rule compared to the formulation of all instances is that it simplifies the knowledge input and saves storage space. The disadvantage is that a more complex rule interpreter is needed.

However, there are good reasons for retaining the object-orientated inference mechanism used so far: it is efficient and can easily be extended by means of additional mechanisms. At the knowledge acquisition level we can nevertheless allow the use of expressive quantified rules, from which the corresponding instances of non-quantified rules are automatically generated by the knowledge acquisition component. For efficiency reasons the instances should be generated at the time of knowledge input, but if necessary this can be done during the dialog, e.g., by means of questions such as: "How many machines of a certain type are there in the production line?" The answer would lead to a multiplication of the individual questions, rules and solutions needed for one machine with their corresponding attributes. Then the newly formed questions are put to the user in the dialog. If there are attributes which are the same for all instances (e.g., question text and the value ranges), they can be inherited from the "generic" object which served as a pattern.

## 16.7 Treatment of Multiple Solutions

Multiple solutions occur above all in fault-finding, because several independent causes may be present at the same time. Since the different causes usually evoke different symptoms, from which they can be recognized, the detection of multiple solutions appears at first sight to be unproblematical. The difficulties arise from the fact that even when the fault has only a single cause there might be several likely hypotheses. If the best hypothesis is always selected in such cases, then multiple solutions can never be found.

This problem can be overcome if not all solutions are regarded as competitors from which the best is chosen, but only those which have relatively similar symptoms. The similarity of solutions can be derived by the knowledge acquisition component through a comparison of their rules or may be given directly

by the expert. Several established solutions which are not marked as competitors of each other can then be interpreted as genuine multiple solutions. Two examples of systems which distinguish between independent and competing solutions are INTERNIST, in which the competitor relationship is derived (two solutions are competitors if the less probable solution cannot explain more of the observations of the particular case than the more probable one), and MED2, where the competitor relationship is laid down statically in the knowledge base.

Another problem is the possibility that the symptoms of different, independent causes may overlap, thus masking the normal appearance of the solutions. If such situations are not taken into account in the rules, it may happen that the actual causes are given too low a probability. In heuristic classification such overlaps can be properly treated only with the aid of special additional knowledge, e.g., exceptions or additional conditions of rules, new rules or even new solutions representing combinations of overlapping causes. A more favorable problem-solving method for this case is causal classification (see Chaps. 17 and 18), in which the effects of fault sources are represented explicitly, so that overlapping can be simulated.

## 16.8 Combined Recommendations for Multiple Solutions

Normally each solution contains a recommendation for action, which is indicated to the user or automatically carried out by the expert system when the solution is established. However, if several solutions are under consideration (whether as multiple solutions or because of remaining uncertainties), one would often like to have a single recommendation which takes into account all possibilities, rather than several independent pieces of advice. This can be accomplished by means of a special algorithm or a configuration technique (see Part V) which is called after completion of the actual classification and combines the possible solutions, their probabilities and the costs for carrying out their respective recommendations to a single overall recommendation. The result may be a flow chart showing the order in which repairs or tests are to be carried out or outlining the strategy for selling a product, or it could be a combined recommendation such as a wide-band antibiotic covering all bacteria which have not been excluded.

# 17. Set-Covering Classification

Set-covering classification is suitable for classification problems in which the solutions (causes) evoke particular symptoms (effects) – possibly via intermediate states – with a relatively high reliability. In the simplest form the knowledge representation consists of observations, solutions and rules of the form: solution causes observation ($S \rightarrow O$). The probability of a given solution or group of solutions is greater, the more observations are explained, i.e., covered, according to its rules and the smaller the number of non-observed features which can be derived from it. The basic structure is illustrated in Fig. 17.1. When set-covering classification

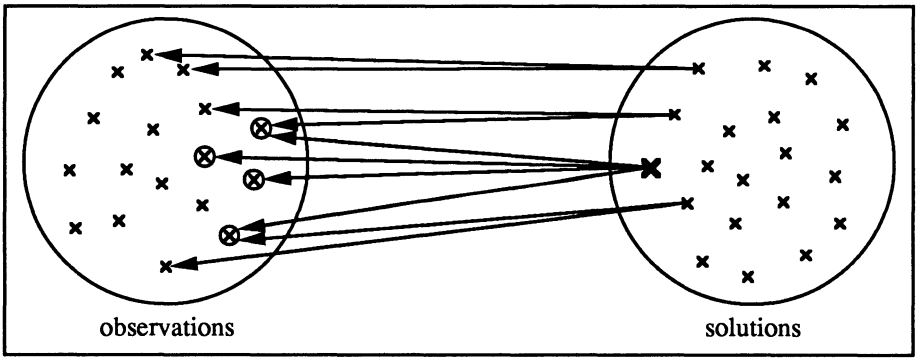



**Figure 17.1** Basic structure of set-covering classification

is used for fault-finding it is called classification with fault models. Two examples of fault models from the technical and medical fields, respectively, are shown in Figs. 17.2 and 17.3. Set-covering is not only suitable for fault-finding, however, but for all domains in which the solutions may be described by characteristic sets of problem features.

In comparison with the heuristic rules "observation indicates solution", the causal rules are usually easier to formulate, but more difficult to evaluate.[1] They are simple because they directly map the causality principle, and evidence values are less significant. The difficulties in the evaluation arise from the fact that causal rules ($S \rightarrow O$) are not directly applicable when the observation $O$ is given.

[1]This has nothing to do with the distinction between forward and backward chaining, where a given rule may be activated in two different ways, whereas here two different rules $A \rightarrow B$ and $B \rightarrow A$ are used.

Therefore hypotheses must first be generated. A simple, but inefficient way of doing this is to suspect all possible solutions (causes) for all observations (effects). In a second step all hypotheses are checked to see whether all observed and no non-observed effects can be derived from them by the causal rules. The differentiating ability of the fault model depends on how refined it is.

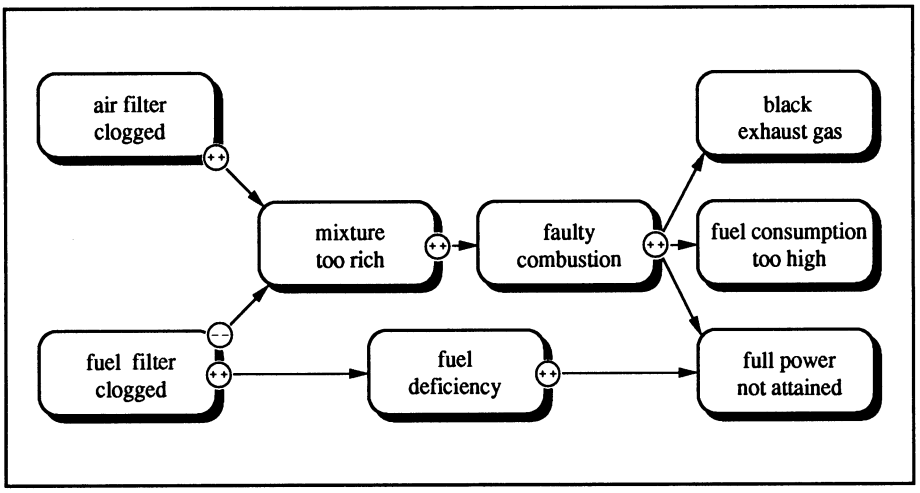



**Figure 17.2** Fault model for clogged air filter and gasoline filter in an automobile. *Formal explanation:* Each state has a degree of gravity, the value of which is passed on to successor states through the rules, which are represented as links. The circles on on the rules contain symbols for the rule relationships, where ++ means direct and -- inverse proportionality. *Explanation of contents for air filter:* A clogged air filter reduces the amount of air, which, if the fuel flow is normal, leads to a rich mixture. This results in faulty combustion, which is revealed by black exhaust gases, a high gasoline consumption and failure to attain maximum power. For comparison the corresponding heuristic rules are given below with evidence values:

| | |
|---|---|
| If exhaust black | then air filter clogged; evidence value: medium |
| If consumption too high | then air filter clogged; evidence value: low |
| If max. power not reached | then air filter clogged; evidence value: low |

The degree of refinement depends on the number of states (observations, solutions and intermediate states) and also on the expressive power of the knowledge representation. The latter can be increased by attaching parameters to the states, e.g., gravity, duration, begin, onset (sudden or gradual). The rules of a state then take account of its parameters and predict not only the successor states but also their parameters. An attractive property is the possibility of combining parameters to investigate the consequences of different causes on a state. In this way the consequences of multiple, overlapping causes can also be predicted, which is hardly possible in heuristic classification. An important special case of parameter relationships is the treatment of negative feedback loops (control circuits), which eventually reduce the gravity of their cause.

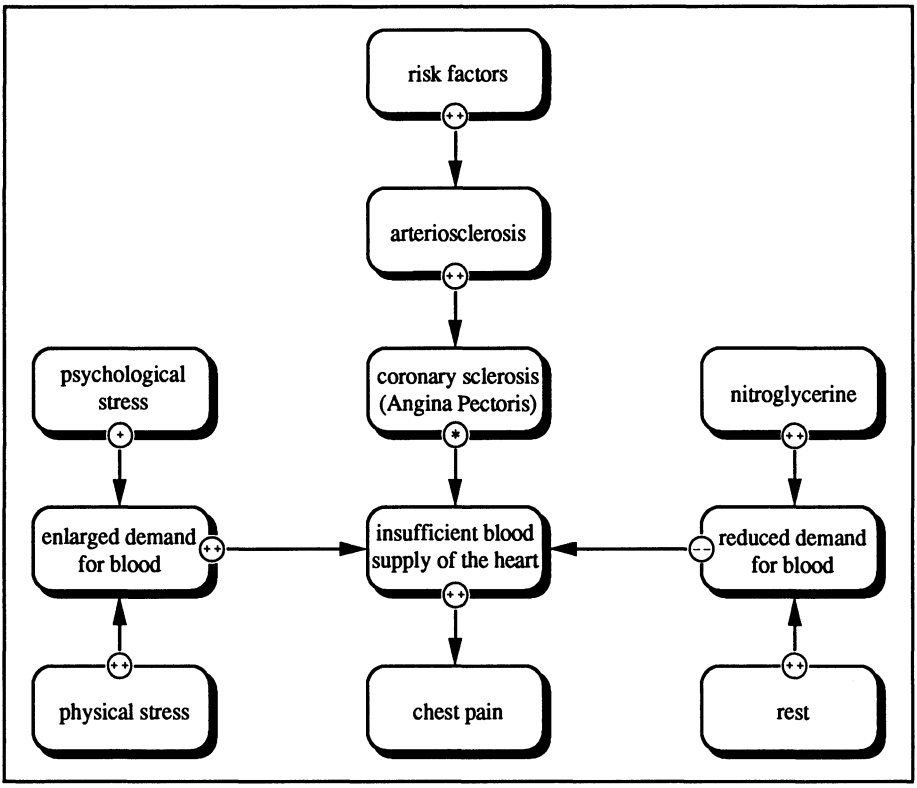



**Figure 17.3** Fault model for angina pectoris. *Formal explanation:* Each state has a degree of gravity, the value of which is passed on to successor states through the rules, which are represented as links. The circles on the rules contain symbols for the rule relationships, where + and ++ represent weak and strong proportionality, respectively, -- strong inverse proportionality and * the necessary condition for other relationships to become effective. *Explanation of contents:* The risk factors (overweight, lack of exercise, smoking, diabetes, high blood pressure) lead eventually to arteriosclerosis (hardening of the blood vessels), which affects in particular the coronary arteries supplying the heart. The damaged coronary arteries, which constitute angina pectoris, can deal with a normal blood supply but cannot satisfy an increased demand for blood due to physical or psychological stress. This is felt subjectively as chest pain. The demand for blood can be reduced by rest or by medicines such as nitroglycerine. The corresponding heuristic rules with the evidence values would be:

| | |
|---|---|
| If number of risk factors=high | then angina pectoris; evidence value: low |
| If pain-increasing=physical stress | then angina pectoris; evidence value: medium |
| If pain-increasing=psych. stress | then angina pectoris; evidence value: low |
| If pain-reducing=rest | then angina pectoris; evidence value: medium |
| If pain-reducing=nitroglycerine | then angina pectoris; evidence value: high |

The disadvantages of great refinement with many states, parameters and rules are decreasing run-time efficiency and increasing complexity of data input. They can be avoided by representing the knowledge in different levels of refinement and using the highly refined levels only when needed. There are two refinement concepts: the simpler form is rule refinement where the states are retained as they are and only the rules are refined by adding intermediate states (e.g., instead of $A \rightarrow B$ the refined form $A \rightarrow C$, $C \rightarrow B$). State refinement is more complex: a single state is replaced by a set of fine states (e.g., "liver disease" by more precise states such as "acute hepatitis", "cirrhosis", etc., or "faulty ignition system" by "faulty timing" or "worn sparking plugs"). Then all rules of the coarse state must be replaced by the more precise rules of the fine states.

## 17.1 Knowledge Representation

The simplest form of knowledge representation has only one object type "state" with the attributes "causes" and "effects", each consisting of a list of states, and a dynamic attribute "status" with the values "established", "unknown" or "excluded". Starting from the observed final states (i.e., states without effects), a set of initial states (i.e., states without causes) is sought which can explain the final states through their "effects". The derivation may go through any number of intermediate states (i.e., states with causes and effects).

A more complex knowledge representation is obtained if the relationships also include combinations of states, in which case they are best represented as separate objects, namely rules. In addition, uncertainties can be taken into account to express the fact that a state may have effects which are usually, but not always realized. It is then necessary to distinguish between the potential "effects" of a state and those which actually occur in a particular case. The knowledge representation must then be extended to include the dynamic attributes "certainty factor" and "actual effects" and the attached procedures "significance" and "derivation", which contain the rules connected with a state.

So far the knowledge representation is not very different from that of heuristic classification, except in the rule semantics.[2] A new element is the parametrization of the states. For this purpose we introduce a new object type "parameters", and each state now bears, as an additional attribute, a reference to its parameters. Each parameter is associated with exactly one state, and has the attributes "value range", "value" and "value components". Typically the parameters represent properties of the states such as gravity or duration. The rules can refer to them, e.g., a state A causes a state B, and the parameters of B are given by those of A, taking into account any factors (i.e., other states) which strengthen or weaken them. If an effect has several causes, their contributions are collected in the attribute "value components", and the "value" is determined from them by simple addition or according to the instructions of special combining rules.

[2]In this chapter we will not discuss mechanisms for data pre-processing or dialog construction and control, since they are very similar to those of heuristic classification.

Finally, it should be possible to specify different abstraction levels of states. For this purpose an attribute "generalization" is provided which indicates whether the state should still be represented on the next higher level. The more abstract model is then automatically calculated from the more detailed model by leaving out those states in which the "generalization" attribute is not set. Since the different levels generally have different rules, care must be taken in problem solving to ensure that the rules of different levels are not mixed.

This gives the following attribute structure:[3]

1. States
   - name
   - causes (set of states)
   - effects (set of states)
   - meaning (s) (rules for deriving the effects)
   - derivation (s) (rules which can cause the state)
   - parameters (set of parameters of the state)
   - status (d) (established, unknown, excluded)
   - certainty factor (d) (for calculating the status)
   - generalization (indication of whether the state is to be retained in simplified models).
2. Parameters
   - name
   - value range (set of possible values)
   - state (to which the parameter belongs)
   - combining rules (s) (rules for calculating the value from the value components)
   - meaning (s)
   - derivation (s)
   - value (d)
   - value components (d) (components with their causes).
3. Rules
   - pre-condition
   - action

[3]Semi-static attributes, which can be derived from the normal (static) attributes by the knowledge acquisition component, are marked (s), and dynamic attributes (d).

## 17.2 Knowledge Manipulation

Causal models can be used in three different ways:

1. Simulation mode: The initial states (causes) are given, and it is required to find the final states (effects) resulting from them.

2. Consistency mode: Both the initial and final states are given, and it is required to determine whether – possibly with what probability – the former caused the latter.

3. Diagnostic mode: The final states (effects) are given, and it is required to find the initial states (causes) which can explain them.

The three forms are closely interconnected: the diagnostic mode includes the consistency mode, and the latter includes in turn the simulation mode.

The central algorithm for the simulation mode consists of a forward-chained rule interpreter with agenda control. Starting from the initial states their rules are activated via the attribute "meaning". They calculate parameter values for new states. This is continued until parameter values have been determined for all states. The agenda control serves to increase the efficiency. Since a given state may be influenced by several other states, the rules which derive consequences from this state should not be triggered until all influences on this state are known, since continual revisions would otherwise be necessary. For the agenda control a criterion is required for determining when a state has been completely calculated, i.e., does not depend directly or indirectly from any other state in the agenda. For this there are various compromises between the amount of calculation needed and the above-mentioned avoidance of revision, e.g., pre-calculation of dependences or a dependence test between two states over one, two or more steps.

Special problems are posed by feedback loops, in which the change of a state is amplified (positive feedback) or attenuated (negative feedback) via intermediate states or may even oscillate. Unless special measures are taken, an endless loop results. Since the treatment of feedback loops reduces efficiency, the first question to ask is whether they are necessary. Technical systems, such as an automobile engine, can often be modeled without any feedback loops, which is not the case for biological systems. Feedback, if present, is usually negative and causes a reduction of the original disturbance. Non-convergent, positive feedback loops generally have catastrophic effects because of exponential growth of the values. Since such behavior is indicative of errors, it is important that it should be recognized before the system "explodes". This can often be accomplished by imposing some (arbitrary) upper limit on the value range. The two most important methods of treating feedback loops may be described as follows:

1. The feedback loop is approximated by comparing the new value of a parameter with the old value after each run through the loop and terminating the process if the absolute difference is less than a given threshold value. This procedure will always terminate for a negative feedback loop, since the

effect becomes smaller with each pass and eventually approaches zero. By changing the threshold value different compromises between precision and efficiency can be chosen. Positive feedback loops, on the other hand, have the property that the difference between the old and new parameter values increases with each pass. They can be terminated by means of an upper limit to the value range.

2. In the case of negative feedback with linear relationships between the parameters the "strength" $\alpha$ of a feedback loop can be pre-calculated and the resulting attenuation determined from the following formula: If $\Delta x$ is the primary change in the value of a parameter, the effect of the loop is to reduce this to $(1-\alpha)\Delta x$. Since non-linear relationships can often be approximated by functions consisting of linear segments, and since positive feedback loops are rare, this procedure can often replace the first, less efficient method.

The basic version of the simulation algorithm, excluding the calculation of a selection criterion for the agenda and the treatment of feedback loops, may be summarized as follows:
Input: initial states with parameter values
Output : final states with parameter values.

1. Put initial states on AGENDA

2. *while* at least one state on AGENDA has effects *repeat:*
Select a state on AGENDA which has effects and whose components have been calculated as completely as possible, calculate its parameter values from the components, delete it from AGENDA, evaluate the rules from its "meaning" and put the effects on AGENDA.

3. Output the contents of AGENDA.

In the simulation mode we assumed that the effects of a state and the cumulative effects of several influences are certain. If the effects do not always occur, they can be given evidence values, e.g., state-1 causes state-2 "mostly" with gravity-1, but also "rarely" with gravity-2. In the simulation mode this information could be used to branch to different alternatives. However, this can easily lead to an explosion in the number of combinations of alternatives, so that it is often better to restrict the choice and take, e.g., only the most probable alternative. A more exact approach for combining gravities and probabilities is fuzzy logic [Zadeh 73].

While the additional flexibility of evidence values is difficult to exploit in the simulation mode, it plays a central role in the consistency mode, in which the initial and final states are given. Here the task is to determine whether there is a consistent explanation of the input, and this may perhaps be possible only under the assumption of some relatively improbable causal relationships. A simple procedure for the consistency mode is to carry out simulations with the initial states until the final states have been derived. The most probable relationships are tried first, and then the alternatives in order of decreasing probability. For maximum efficiency it should be possible to recognize as early as possible when a simulation

is inconsistent, and to modify the last simulation without having to start again from the beginning. To fulfill the latter requirement a reset procedure (TMS, see Chap. 9) is required. The early recognition of the inconsistency of a simulation is favored by proceeding in both directions, i.e., also working backwards from the final states to determine the intermediate states required for their derivation. This is of course possible only if there is not too much branching in backward reasoning.

Backward reasoning is easy if only one cause exists for a given state, since in that case the parameter values of the cause are generally unequivocal. If there are many causes, there are usually many combinations of the parameters of these states which produce the given effect. Unless some causes can be eliminated by additional information or the single-fault assumption can be made, a continuation of the backward reasoning in such cases becomes very inefficient. Backward reasoning up to the point where it becomes inefficient (because of too much branching) results in the production of one or more "basic models" which trace the given final states back to a few intermediate states which are necessary for their derivation. The simulation now needs to be carried only as far as these intermediate states, which simplifies the consistency test. From this we obtain the following basic algorithm for the consistency mode:
Input: initial and final states with parameter values
Output: statement about the consistency of the input

1. Calculate by backward reasoning one or more basic models for the given final states. The backward reasoning should be terminated when a continuation would result in too many basic models.

2. Carry out a simulation with the given initial states in which at first only the most probable alternative rules are used.

3. *While* the result of the simulation is not a basic model, and there are rules which have not been tried, *repeat*
   Select the next most probable alternative rule and modify the simulation accordingly.

4. *If* the result of the simulation is a basic model,
   *then* output: certainty factors of the rules as assessment of the consistency
   *else* output: no consistent interpretation of the input data.

In the diagnostic mode we have the additional problem of generating hypotheses. The latter are tested with the consistency mode, and the best is selected. A frequently used, relatively efficient method of generating hypotheses is to calculate for each final state a "conflict set" containing all its potential causes (initial states with assignment of parameter values) and then to form a "solution set" which contains at least one element of each conflict set. The set of all "minimal" solution sets – i.e., those which do not contain another solution set as a subset – represents all possible causes. Single-element solution sets are potential causes which can explain all final states with a single initial state. In this method of

generating hypotheses the parameter values of the states are ignored. Theoretically therefore, all parameter combinations must be tested for each state, which increases the number of hypotheses considerably. The parameter values to be tested can be determined simply by backward reasoning from the parameters of the final states provided that there is only one way from at least one initial state to a final state, i.e., no two paths from $A$ to $C$ via two different intermediate states $B$ and $D$. For the efficient treatment of multiple paths the total influence between pairs of states connected by multiple relationships can be pre-calculated.

The most obvious method of generating hypotheses – reasoning backwards from the final states, taking into account the parameter values – is usually inefficient on account of the high branching rate. It is suitable above all for seeking a single cause when multiple paths between initial and final states also exist. (Pre-calculation does not help much in this case, because no potential causes are known when reasoning backwards).

Finally, a third way of generating hypotheses is to simplify the model and then use one of the above two methods. The simplification is done by eliminating some of the states and expressing the relationships on a higher level of abstraction. Which states are to be eliminated is determined by the attribute "generalization" of the states in the knowledge base. The abstract forms of the relationships can then be generated automatically. In all, two (or more) models are used: the main (detailed) model is used for testing the hypotheses, and the simplified model formed according to the "generalization" attributes is use for generating hypotheses.

Thus the central algorithm of the diagnostic mode consists of the following steps:

1. Generation of hypotheses by one of three methods:
    a. Calculate the minimal solution set of initial states and estimate their parameter values.
    b. Reason backwards from the parameter values of the final states, using the complete model.
    c. Reason backwards from the parameter values of the final states, using the simplified model.
2. Test of hypotheses: carry out a consistency test with every set of initial states generated in step 1, with parameter values.
3. Differential diagnostics: select the set which gives the "best" model. Possible selection criteria are: maximum certainty factor, if different models give complete explanations; difference between predicted and given final states, if no model fits completely; the number and *a priori* probabilities of the postulated initial states, if it is necessary to decide between alternatives which are otherwise similar.

## 17.3 Knowledge Acquisition

The procedure for knowledge acquisition is similar to that for heuristic classification (see Sect. 15.5):

1. Naming and sorting the states,
2. input of local attributes and parameters for the states,
3. exact specification of the rules between the states.

As graphical aids, networks are suitable for the first step (see Figs. 12.5, 17.2 and 17.3), and forms for the second. Since the rules usually correspond to links in the network, simple rules can be defined directly when constructing the network through symbols on the links (like the circled symbols in Figs. 17.2 and 17.3 indicating the strength and sign of proportionality). For entering more complex rules pre-defined forms are provided.

## 17.4 State of the Art

Fault models have been investigated above all in medical diagnostics. Pioneer work was done by Ramesh Patil of MIT with his system ABEL for diagnosing acid-base and electrolyte disturbances [Patil 81,82], which is still of interest today. As knowledge representation ABEL uses fault states (called "pathophysiological states" in medicine) which have as parameters gravity and duration. From the input data, which include the blood analysis results such as sodium concentration and pH-value, ABEL instantiates final states with their parameters and then attempts to relate these states to one another in a basic model (the patient specific model = PSM). Finally, initial states (diagnoses) are sought for those final states which are not explained in the PSM. ABEL can make use of various simplifications in comparison with the above treatment: Since the interactions between the pathophysiological states are mostly known as mathematical equations, the degrees of gravity can be represented as quantitative absolute values, propagated and if necessary added. Since the domain is relatively small, and there are no intermediate states between diagnoses and unexplained states in the PSM, ABEL can form all solution sets and test their effect. As assessment criterion ABEL does not use certainty factors, but tests how well the diagnoses can account for all the unexplained states in the PSM with the correct parameters and whether they predict other states which were not observed. ABEL has a well elaborated system of abstractions of the quantitative model to several levels; these serve only for explanations, they are not required for generating hypotheses. Provision is also made for feedback loops in ABEL, but the exact method of calculation is not described in the literature.

A fault model which also deals efficiently with feedback loops was developed by Long [88] for modeling the blood circulation system. It contains states (e.g., systolic blood pressure) with gravities expressed as deviations from the normal

state (on a scale from $-3$ to $+3$). In order that the numerous feedback loops can be treated efficiently they are pre-calculated, non-linear relationships being approximated by linear segments, so that the above special formula can be used. In Long's system the diagnosis takes place in three steps: First, using a simplified model, a causal chain from the initial states via intermediate states to the given final states (symptoms) is determined. Then a medicine is sought, in simple tables, which influences one of the states in the chain. This state should be as near as possible to the beginning of the chain, but it is often not possible to influence the initial states themselves. The tables have rows of the form: "Medicine-1 reduces state-1 strongly ($-3$), state-2 moderately ($-2$) and increases state-3 slightly ($+1$)." In Long's system the tables are relatively small, so that the best medicine can be selected directly. For more complex tables, however, a variant of the propose-and-revise strategy for parameter configuration (see Sect. 23.5) is recommended, in order to find as far as possible the optimum medicine configuration. In the third step the detailed effects of the proposed medicines are simulated with the complete model, since they can only be described roughly in the tables. It may turn out that the effects are much smaller than expected on account of the feedback loops, or unexpected side effects may occur.

Further research work on less complex fault models is described in [Reggia 83a,b], [Steels 85], [Kahn 88] and [Eshelman 88]. In [Reggia 83a,b] only simple fault models having states without parameters and without intermediate states[4] are used. The problem-solving method is an algorithm for set covering, which attempts to find a set of diagnoses which explains all observed states. The term "set-covering classification" is taken from Reggia. In [Steels 85] a simple fault model is introduced as a basis for learning heuristic rules. MORE [Kahn 88] uses a somewhat more complicated fault model for the semi-automatic knowledge acquisition of heuristic rules. In MORE the gravities of the states are not represented, but in a causal rule of the form $D \rightarrow S$ a distinction is made, on the symptom side, between symptoms, symptom attributes and symptom conditions. Symptom attributes represent refinements of the symptoms (if a symptom $S$ has three refinements $S_1$, $S_2$ and $S_3$, then the rule "$D \rightarrow S$" can be refined e.g., to "$D \rightarrow S_1$") and symptom conditions indicate the contexts for the occurrence of a symptom (if there is a condition $K_1$ which favors the occurrence of $S$, "$D \rightarrow S$" can be refined, e.g., to "$D \rightarrow S$ in the context $K_1$"). Symptom attributes and conditions are used to decrease the uncertainty of the rules generated. In MORE's successor MOLE [Eshelman 88] a fault model with intermediate states, but without gravities, is used directly for problem-solving, whereby the concepts of symptom attributes and symptom conditions are not taken over explicitly from MORE.

A review of fault models, particularly in medicine, can be found in [Patil 88].

[4] In [Peng 90], intermediate states are also treated.

## 17.5 Example: FEMO

FEMO [Goos 89,90] is an example of an expert system tool using set-covering classification. Its knowledge representation and the techniques used for knowledge manipulation are those described in Sect. 17.1 and 17.2. A text file is used for knowledge acquisition; a graphical editor is being developed. The automobile engine was used as an example for the application, an we will use this example to explain the way FEMO operates. Part of the knowledge base is shown in Fig. 17.4.

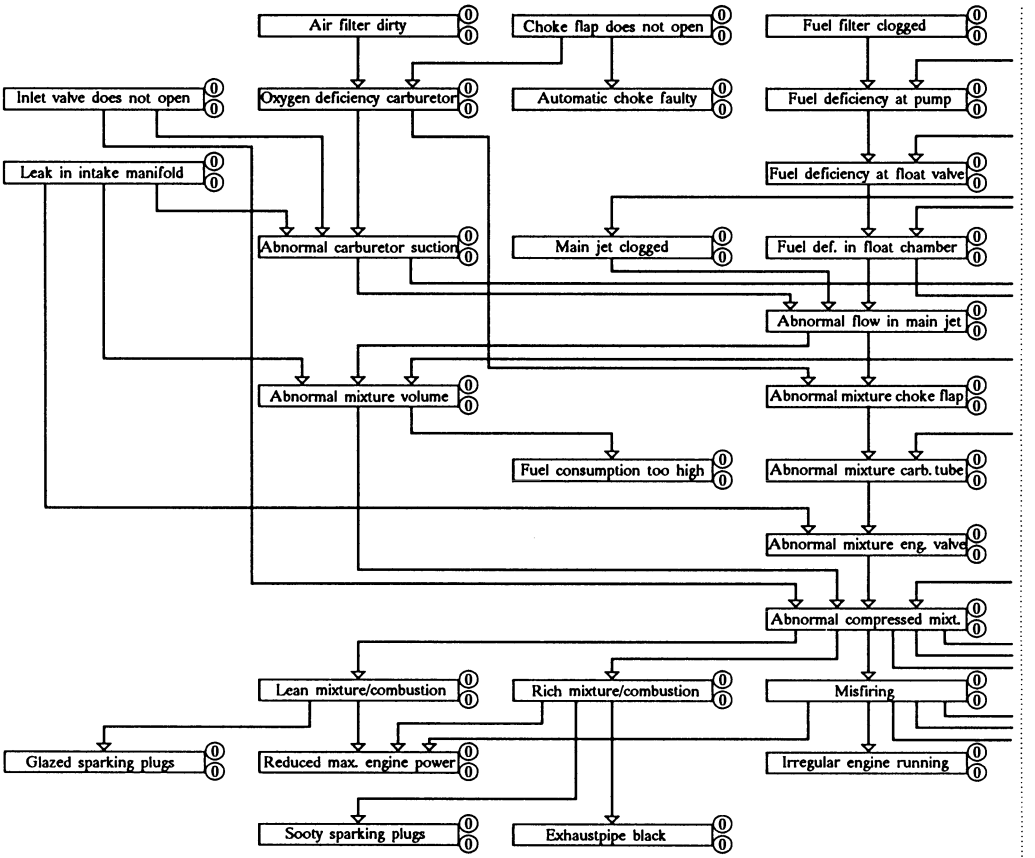



**Figure 17.4** Part of the fault model of the automobile engine from FEMO

A small section of the knowledge base (Fig. 17.5) may serve to illustrate the knowledge representation. Each state has two parameters, gravity and duration, with either four or seven qualitative values. The state "misfiring", for example, has the values "very frequent" (3), "frequent" (2), "seldom" (1) and "none" (0) for the gravity and "long" (3), "medium" (2), "short" (1) and "just observed" (0) for the duration. With other gravities low values ($-1$ to $-3$) can occur in addition to the high values (1 to 3) and normal (0). The current values of the parameters of each state are indicated in the small circles at the top right (gravity) and

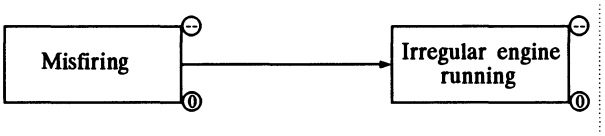



**Figure 17.5** Section of the knowledge base of Fig. 17.4 with the specific parameter assignments

bottom right (duration), where "−−" means gravity "2". The arrows between the states represent the rules between the parameters, e.g., *if* (misfiring established and gravity $\geq$ 2) *then* (establish uneven running and set its parameters equal to those of misfiring). If a state is affected by several influences, the gravities are added and the minimum duration is chosen, unless the expert has specified special "combining rules" for the parameters (e.g., for the influence of the states "leak in intake manifold", "inlet valve does not open" and "oxygen deficiency carburetor" on the state "abnormal carburetor suction" in Fig. 17.4, not the sum of the gravities is taken, but the maximum). The duration plays a role in those effects which require time to produce consequences (e.g., the air filter must be damaged for some time before dirt is deposited in the cylinder).

The fault model is used in the three modes described in Sect. 17.2 (simulation, consistency and diagnostic mode). The input consists of specifications of initial and/or final states with gravity and duration. An example of the diagnostic mode following the input "reduced maximum engine power" with the gravity "greatly reduced" (−2) is illustrated in Fig. 17.6, which shows FEMO's explanation of the first of the two diagnoses found, namely "fuel filter clogged" with gravity "slightly" (−1). The other diagnosis was "main jet clogged" with the gravity "slightly" (−1).

Special features of FEMO are the ability to deal with negative feedback loops by both methods described in Sect. 17.2, the specification and treatment of global operating conditions (e.g., normal load or idling), which in the form of contexts can control the activation of rules, and the automatic abstraction of the model when some states are removed from the basic model (the rules of the eliminated states are transformed approximately to the remaining states).

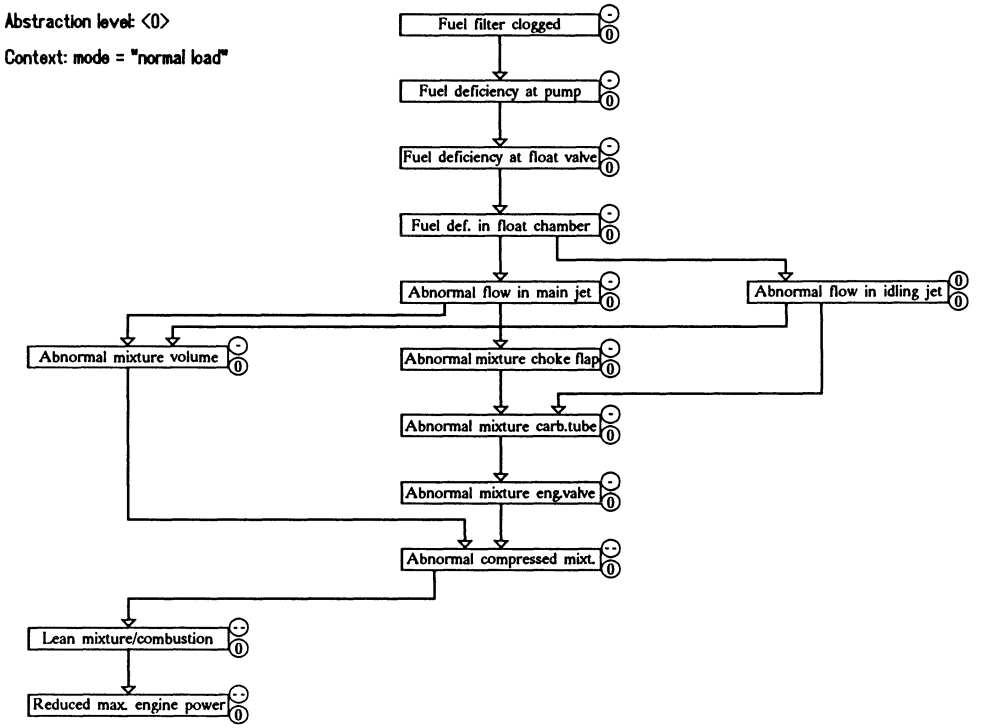



**Figure 17.6** Output of the explanation for the symptom "reduced maximum engine power" with the gravity "greatly reduced" (−2, represented as "−−" in the small circle at the top right of the state) and the duration "just observed" (0 in the bottom right circle)

# 18. Functional Classification

Functional classification is suitable for fault-finding in systems whose normal and defective structures can be described by materials and components with states, and whose normal and abnormal behavior can be derived from the description of the behavior of the components. A functional model describes how the input materials are converted into the output materials in a system. Examples are the gasoline engine, which converts air and gasoline into motive power and exhaust gases (Fig. 18.1), the blood circulation for supplying oxygen to the body (Fig. 18.2) and a circuit for addition and multiplication, which has numbers as input and output (Fig. 18.3).

The components are usually fixed parts and the materials moving parts of the system. The behavior of the components is described by a set of rules specifying how they transform the input into the output material. The materials, which may represent the transport of matter, energy or information within the system, are characterized by a set of attributes whose values are specified by the user or derived by the components. The component "air filter" in the automobile engine, for example, sets the value of the attribute "contamination" of the material "air-2" to "clean", while the other attributes of "air-2" are taken over unchanged from "air-1". If the air filter is clogged, however, the "pressure" of "air-2" is lowered, and the lower value is passed on to other components, which finally leads to symptoms, i.e., to abnormal values for the output materials.

In functional models every discrepancy between the expected and observed values of materials is a symptom (observation), and every abnormal behavior of a component is a diagnosis (solution). However, not every logically possible abnormal behavior is also physically possible (see Sect. 18.4). For this reason, and also to reduce the complexity, the typical faults of a component can be defined as its states and the corresponding abnormal behavior given by specific rules.

In principle, the attribute values of the materials correspond to measurable parameters of the real system. However, since we are interested mainly in the discrepancies, functional classification uses the deviations of the values from their normal states rather than the absolute values. Thus the value range includes such descriptions as "value too high, normal, or too low", where the deviation can be given qualitatively, with different gradations, or quantitatively. The advantage of working with changes of values instead of absolute values is a considerable reduction of the complexity of the model, since in the fault-free state all values

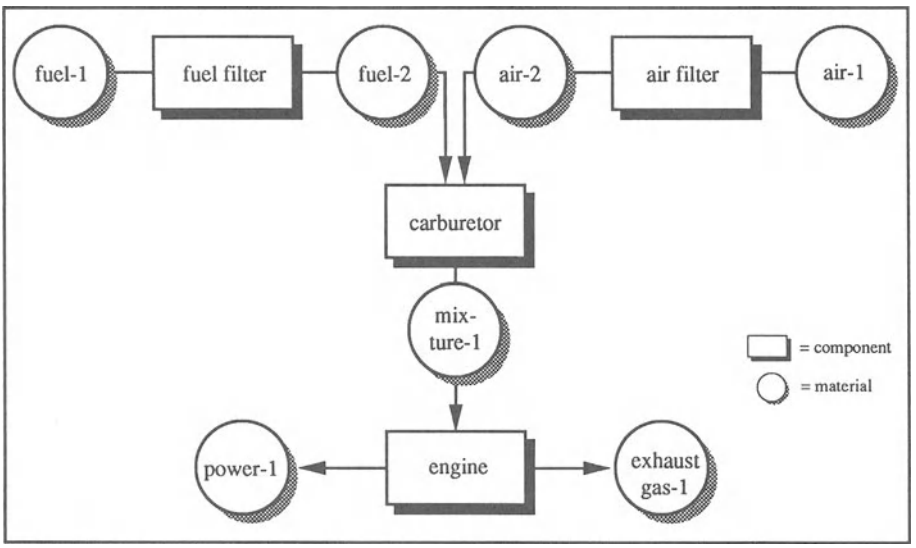



**Figure 18.1** Simple functional model of the gasoline engine. Explanation:

| | | |
|---|---|---|
| Materials: | air-1, air-2: | type air with attributes contamination {clean, dirty} and pressure {normal, too low} |
| | fuel-1, fuel-2: | type fuel with attributes contamination {clean, dirty} and pressure {normal, too low} |
| | mixture-1 | type mixture with attribute ratio {too rich, normal, too lean} |
| | exhaust-1 | type exhaust with attribute color {colorless, black} |
| | power-1 | type power with attribute intensity {normal, too low} |
| Components: | air filter in different states {normal, clogged, torn} with the rules: | |
| | normal: | air-2.contamination ::= clean<br>air-2.pressure ::= air-1.pressure |
| | clogged: | air-2.contamination ::= clean<br>air-2.pressure ::= air-1.pressure - 1 |
| | torn: | air-2.contamination ::= air-1.contamination<br>air-2.pressure ::= air-1.pressure |
| | Similarly for the other components | |

are "normal", and only the deviations need to be propagated. The approximate absolute values can be determined if for each attribute the normal absolute value and a scheme for determining the quantitative effects of the deviations on it are given. If the normal value depends on individual system specifications it can be derived by means of corresponding "normalizing rules" (see Sect. 18.5).

Whether the value changes are given quantitatively or qualitatively depends upon the accuracy of the model. Generally quantitative formulas are not available for all system relationships, either because the knowledge is lacking or because too many variables are involved which would be too difficult to measure. On the other hand, qualitative formulas have the disadvantage that the addition of several effects to one value may lead to uncertainties. If a value is increased (+)

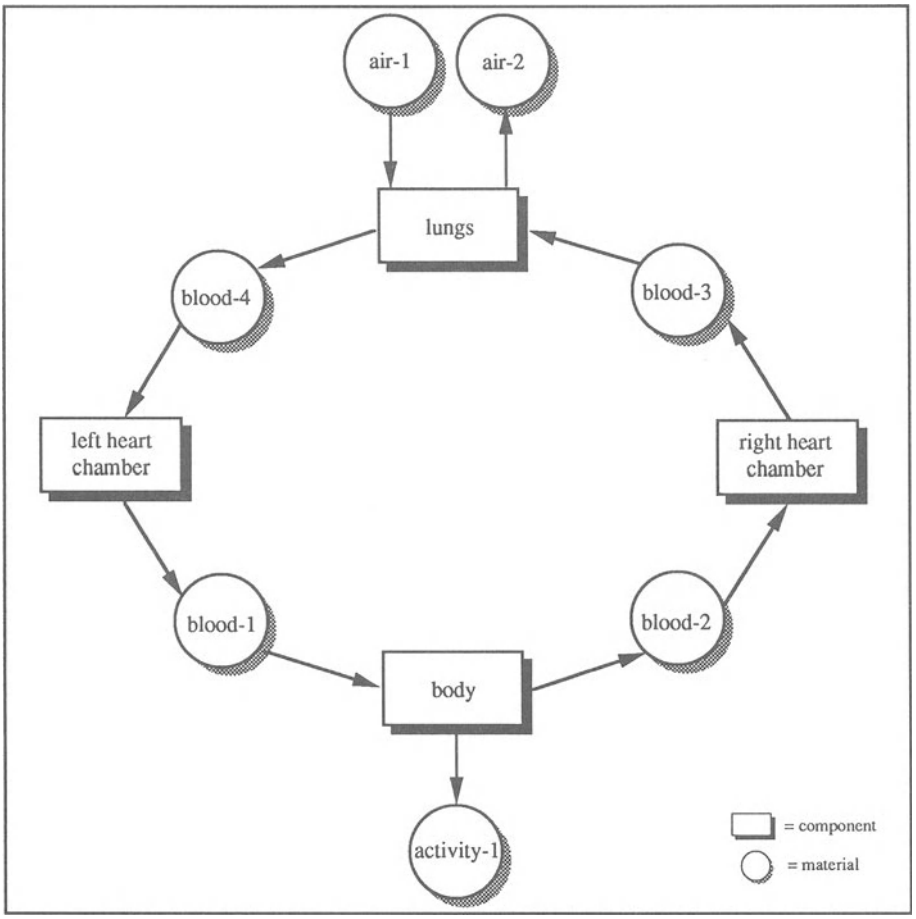



**Figure 18.2** Simple functional model of the blood circulation. Explanation:

| | | |
|---|---|---|
| Materials: | blood-1, blood-2 | type blood with attributes oxygen-content {high, |
| | blood-3, blood-4: | medium, low} and pressure {high, medium, low} |
| | air-1, air-2 | type air with attribute oxygen-content {normal low, low} |
| | activity-1 | type activity with attribute intensity {normal, low} |
| Components: only the normal behavior is given: | | |
| | left chamber: | blood-1.pressure ::= high |
| | | blood-1.oxygen-content ::= blood-4.oxygen-content |
| | body | blood-2.pressure ::= low |
| | | blood-2.oxygen-content ::= low |
| | | activity-1 ::= blood-1.oxygen-content(normal ← high; |
| | | low ← medium, low) |
| | right chamber | blood-3.pressure ::= high |
| | | blood-3.oxygen-content ::= blood-2.oxygen-content |
| | lungs | blood-4.pressure ::= low |
| | | blood-4.oxygen-content ::= air-1.oxygen-content (high |
| | | ← normal; medium ← low) |

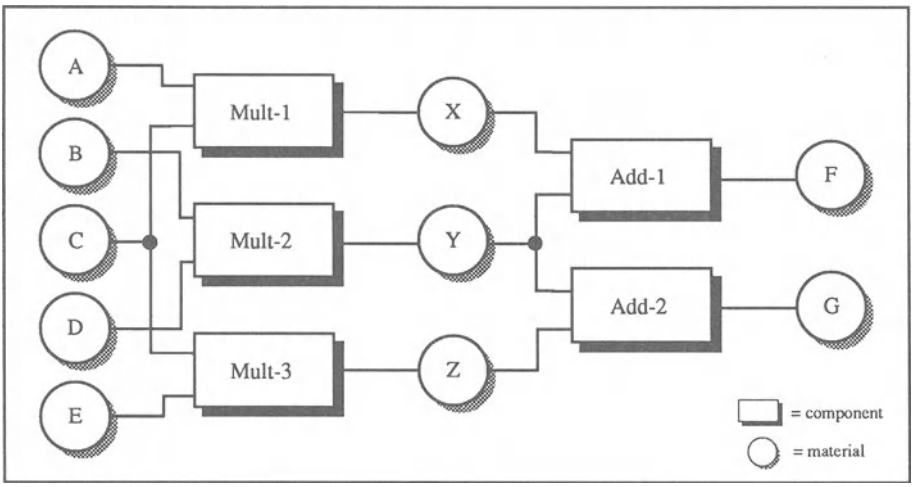



**Figure 18.3** Functional model of a simple circuit. Explanation:

Materials: A,B,C,D,E,X,Y,Z,F,G: type number with attribute value.
Components: mult-1, mult-2, mult-3: type multiplier; behavior: input-1 × input-2 = output
add-1, add-2: type adder; behavior: input-1 + input-2 = output.

by one influencing factor and lowered (−) by another, for example, qualitative arithmetic gives: + and − is not 0, but indeterminate, all three values $\{+, 0, -\}$ are possible, and the system must branch into all three directions. A solution of this problem requires additional knowledge, of which there are various forms:

1. Inclusion of knowledge about the order of magnitude of the effects, which may then be compared. If one effect is an order of magnitude larger than the other, the second can be neglected.

2. Mapping the gradations onto numbers, e.g., much too high = 3, too high = 2, high = 1, normal = 0, low = −1, etc., and using normal arithmetic. This also makes it possible to calculate the effects of feedback loops (see Sect. 17.2).

3. The use of quantitative formulas when needed, i.e., the formulas for quantitative relationships – if available – are stored, but used only to resolve conflicts.

Different levels of abstraction can be represented by describing a component on the higher level only by its behavior rules and on the more detailed level by its internal structure with sub-components and materials. The input and output materials of the component on the higher level must agree with those of the group of sub-components on the lower level, since they form the interface to the other components.

## 18.1 Knowledge Representation

Functional models can be represented by a network of components and materials, where the materials also constitute the connections between the components. The components contain mainly rules for handling the materials. If it is required to specify faulty functions of the components, the components are given states (normal states and faulty states) which each have their own set of rules. The materials are characterized by a set of attributes which are represented by parameters. Since there are usually many instances of a given material or parameter type, these types should be defined as generic objects from which the instances inherit the basic structure. A parameter has an absolute normal value and its actual value, which represents the deviation from this normal value. We thus obtain the following attribute structure:[1]

1. Components
   - name
   - input materials
   - output materials
   - states (set of possible states)
   - normal state
   - current-state (d)
2. States
   - name
   - rules (for treating the input materials)
   - *a priori* probabilities (indication of frequency of the state; optional)
3. Material types
   - name
   - parameter types (list of all parameter types)
   - instances (list of materials)
4. Materials
   - name
   - material type
   - pre-components (set of components where the material is the output)

[1]Semi-static attributes, which can be derived from the normal (static) attributes by the knowledge acquisition component, are marked (s), and dynamic attributes (d).

- post-components (set of components where the material is the input)
- parameters (attributes of the material)

5. Parameter types
   - name
   - material type
   - value range
   - units
   - instances (list of parameters)
6. Parameters
   - name
   - parameter type
   - material
   - normalizing rules (s) (for calculating the absolute normal value of the parameter)
   - derivation (s) (rules for deriving the value of the parameter)
   - meaning (s) (rules for drawing conclusions from the parameter value)
   - combining rules (s) (for processing value components)
   - normal value (d) (absolute normal value of the parameter, calculated with the normalizing rules)
   - value components (d) (deviations from the normal value calculated with the derivation rules)
   - value (d) (calculated from the value components with the combining rules)
7. Global system parameters
   - name
   - value range
   - normalizing rules (s) (for calculating the absolute normal value of the parameter)
   - value (d) (current value, initially given by the user)
8. Rules
   - type (normal or deviation)
   - pre-condition
   - action

If the "superstructure" (components, material types, materials, parameter types, global system parameters) is removed, the remaining core contains only states and parameters which at first sight correspond to the knowledge representation of the fault models. However, in fault models the states and parameters describe only the fault propagation, whereas in functional models they bear more information: not only the modeling of the faulty behavior but also the representation and propagation of normal values. Since most of the states are "normal" even in faulty systems, this requires a higher complexity. The "superstructure" is therefore needed to impose a clear structure on the much greater number of states and parameters in functional models. We have not considered assigning evidence values to the rules in functional models – although this is possible just as for fault models – since this would even further increase their complexity.

## 18.2 Knowledge Manipulation

A functional model can be used for a variety of purposes: for hypothesis generation, starting from the observed deviations of the materials and looking for the components which are responsible, for hypothesis testing, starting with given initial states and component faults and simulating their effects on the model, or for explanation, starting from given component faults and abnormal material values and looking for a consistent explanation in the model. All modes exist also in case-covering classification with fault models (Chap. 17), from which the basic procedure can also be taken (Sect. 17.2). The difference consists in the more detailed knowledge representation with many more objects, properties and relationships, since the aim is to describe not only the fault propagation but the complete function of the system. It follows that the hypothesis generation phase is less efficient. If the hypotheses are not given externally by the user or another program, it is often necessary to simplify the boundary conditions for their generation, e.g., by treating only those components or parameters which can be either "in order" or "not in order". If necessary, simple boundary conditions can be produced by abstraction.

Hypothesis testing by simulating the given input data with the network differs hardly at all from that used in fault models. It corresponds to a forward-chaining process in which the input materials activate the corresponding components, where they are converted to the output materials according to the given states and rules. This process is repeated until all components have been dealt with. If a component has several input materials, it should not be activated until all input parameters are known (cf. the agenda mechanism and the problem with feedback loops in Sect. 17.2).

The result of the simulation is a set of predictions about the effects of the assumptions which can be compared with the actual observations. We thus obtain three sets of symptoms: 1) predicted and observed, 2) predicted, but not observed, 3) observed, but not predicted. The first set is evidence for the correctness of the

hypotheses, the second is suitable for indicating additional tests and the third is evidence against the hypotheses.

The basic algorithms for evaluating functional models are as follows:

I. Simulation mode:
Input: values of input materials, states of components, global system parameters.
Output: values of output materials.

1. Put all components which are connected to the input materials on a global list AGENDA.

2. *While* AGENDA not empty *repeat*

   2.1 Select a component from AGENDA – as far as possible one which does not depend on any other component in AGENDA. From its input materials (if any are unknown, assume that they are "normal") calculate the output materials. Delete the component from AGENDA.

   2.2 *If* there are other components connected with the output materials, *then* put these components on AGENDA
   *else* put the output materials on RESULTS-LIST.

3. Output materials from RESULTS-LIST.

II. Diagnostic mode:
Input: values of input and output materials, global system parameters.
Output: States of components.

1. Finding discrepancies: Using the given global system parameters calculate the values of the output materials on the assumption that all components are in their normal states and compare the results with the given values. Every difference is a discrepancy. This step is not necessary if the input data are already given as deviations from the normal state.

2. Hypothesis generation: Determine for each discrepancy a "conflict set", which contains all components directly or indirectly involved in the calculation of the discrepancy together with their possible states. Form "minimal solution sets" of all conflict sets (see Sect. 17.2).

3. Hypothesis testing: Each solution set is a hypothesis which is simulated by changing the corresponding states of the components.

4. Differential diagnostics: For each simulation compare the predicted and observed states. Criteria for the comparative assessment of different hypotheses are: a) derivation of all observed discrepancies, b) derivation of no discrepancies not observed, c) smallest set of defective components (which may be weighted with the *a priori* probabilities).

Additional mechanisms used in classification with fault models (such as the weighting of alternative causal relationships with evidence values, the treatment

of feedback loops and the abstraction of the model for more efficient hypothesis generation) could in principle also be used here. However, they have not been included in the algorithms because they would lead to a considerable increase in complexity.

## 18.3 Knowledge Acquisition

In the field of knowledge acquisition functional classification has the great advantage over fault models and heuristic classification that no heuristic knowledge may be required at all. Instead, the normal functioning of the system is modeled, and this is usually known in sufficient detail in the case of technical systems. The knowledge input begins with the definition of the material and parameter types and the construction of a graphical network with named components and materials (as instances of the material types). Labeled boxes or abstract pictures (ikons) can be used as graphical symbols for the components, and circles for the materials. These symbols can be selected from the edge of the screen as in many drawing programs, named, and moved into position with the mouse. Then the mouse may be used to draw connecting lines representing links to objects already defined. The result is a graph like those shown in Figs. 18.1–18.3. After the initial model has been constructed, the local knowledge is entered by activating the object (mouse click) and filling in the form which appears.

## 18.4 State of the Art

The aim of the work of de Kleer, Forbus and Kuipers [AI Journal 84] on qualitative reasoning was to determine the behavior of a system from the structure and the behavior of its components. The functional classification described here has a different approach: a description of the behavior of the system is assumed, and from the various possible behaviors the one which agrees best with the symptoms (observed features) is selected. The work of de Kleer, Forbus and Kuipers will be discussed in connection with qualitative multiple-phase simulation (Chap. 32).

One of the first systems for functional classification to be published was HTE by Davis [84] for hardware diagnostics. The components are adders, multipliers, etc., and the materials have only one parameter with a numerical value range. The components can only have the state "in order" or "not in order", the output being any number in the latter case. Since the relationships involved are exact, Davis was able to represent the absolute values of the parameters, which are changed by the components according to the rules of arithmetic. Further, HTE can perform forward or backward reasoning with equal efficiency, which considerably simplifies the generation of hypotheses. The first step in HTE is to find discrepancies by forward reasoning from the input values and comparison with the output values. For each discrepancy a set of potential diagnoses, i.e., components with abnormal behavior, is then generated by backward reasoning. The suspected components

are checked by setting their outputs to values which cause their direct discrepancies to vanish. With these assumptions a simulation is performed by forward reasoning. If no new discrepancies occur, the abnormal components constitute a consistent explanation for the observed faults. The faulty components can then be investigated in a similar way on a lower level of abstraction (the bit level), in order to find out which subcomponents are responsible for the fault. A special feature is the possibility of introducing additional components in the model, in order to simulate faults which would otherwise be difficult to describe locally, such as bridges between neighboring conductors. However, the hypothesis generation for this purpose is very complicated.

Another important system is GDE (general diagnostic engine) by de Kleer and Williams [87], which was further developed by Struß and Dressler [89] and de Kleer and Williams [89]. The most important field of application of this system is also circuit diagnostics. As in HTE, a component has only the two states "in order" or "not in order". The special feature of HTE is the use of an ATMS [de Kleer 86] for deriving hypotheses (conflict sets) for discrepancies. By this means it is relatively easy to generate multiple causes.

Struß and Dressler [89] have shown with GDE+ that the performance of GDE can be considerably improved with the aid of explicit knowledge about the behavior of a faulty system, because this allows many logically possible faults to be excluded on physical grounds. They give the example of a circuit consisting of three lamps $L1$, $L2$ and $L3$ connected in parallel to a battery. If $L1$ and $L2$ do not light, the most plausible hypothesis is that these two lamps are defective. However, GDE would generate a number of other hypotheses such as "the battery and $L3$ are not in order", which must be interpreted as follows: The defective battery is the cause of $L1$ and $L2$ not lighting, and $L3$ is not in order, because it lights in spite of the fact that the battery is defective! Such absurd hypotheses are avoided in GDE+ by making use of knowledge about faulty states. De Kleer and Williams [89] went a step further with SHERLOCK, in which the faulty states are given *a priori* probabilities in order to be able to select the best of a number of physically possible faulty states, or the best test for confirmation. In these further developments of GDE the attachment of gravities to the faulty states, which would complicate the model considerably, has not been considered.

A general description of diagnostics with functional models can be found in [Hamscher 87] and [Reiter 87]. Reiter distinguishes only between "normal" and "abnormal" components and does not consider gravities or knowledge about faulty behavior. He gives an algorithm for generating hypotheses for the causes of discrepancies which is based on a contradiction proof with a general theorem prover.

## 18.5 Example: SIMUL

SIMUL [Hnyk 88] is an example of an expert system tool for restricted functional classification; its knowledge representation corresponds largely to that described

in Sect. 18.1. The restriction is that hypotheses cannot be generated by SIMUL, but must come from outside the system, e.g., from the user or via a connection with the heuristic classification system MED2 (see Sect. 15.4). At present knowledge acquisition is carried out with an interactive editor, as yet without graphical support. The automobile engine will be used as an example application for explaining how SIMUL works. A general view of the main model used for the automobile engine is shown in Fig. 18.4. For the two components "engine" and "carburetor" of the main model there are submodels which are not shown in Fig. 18.4.

The input for SIMUL includes possible faulty states of the components (e.g., "air filter partly clogged" or "battery flat") and values of the global system parameters which characterize the current mode of operation of the engine, e.g., gear {neutral, 1–4, reverse}, engine temperature {cold, warm}, engine state {running, stopped} or choke {pulled, not pulled}. The output consists of symptoms – with gravities – weighted according to their significance, e.g., "no ignition in combustion chamber when starting", and their causal derivation from the faulty states. The knowledge representation may be explained on the section of the knowledge base shown in Fig. 18.5. The "air pipe" is a connection between two components and carrier of a material, namely air (a connection may possibly carry more than one material). The material "air" is described by the parameters "pressure", "flow rate", "degree of contamination", "oxygen content" and "temperature". The parameters have a uniform value range with seven numerical values representing qualitative steps from very low (−3) via normal (0) to very high (+3). These values represent changes from the normal state.

The normal value is either specified directly (e.g., air contamination := 100 ppm) or calculated with the aid of the global system parameters, e.g., if (engine-temperature = warm) then (air.temperature := 30 degrees centigrade); if (engine-temperature := cold) then (air.temperature := 20 degrees centigrade); if (engine-state = running) and (gas-control = idling) then (air.pressure := 0.9 bar) and (air.flow-rate := 1 m/s). The normal value is not needed for fault propagation; it serves only for a rough interpretation of the results: When the air.pressure is "slightly low" (−1), for example, this means a pressure reduction of 10–30% with respect to the normal value of 0.9 bar.

The components have a behavior description with rules for converting the input parameter values into the output parameter values. In SIMUL each component has a normal state and various faulty states. The air filter, for example, has the states "normal", "clogged", "broken" and "low oxygen". Each state has its own rules, e.g., for the air filter in the normal state "output-air.temperature := ($prop input-air.temperature)", where the function $prop transfers any increase or decrease of the input parameter unchanged to the output parameter. If the air filter is clogged, the following rule is activated: "output-air.flow-rate := ($reduce input-air.flow-rate)", which corresponds to a reduction of one step (−1) in the input parameter. The reduced flow rate can be detected in the pre-conditions of the rules of other components and thus cause specific effects. For combining

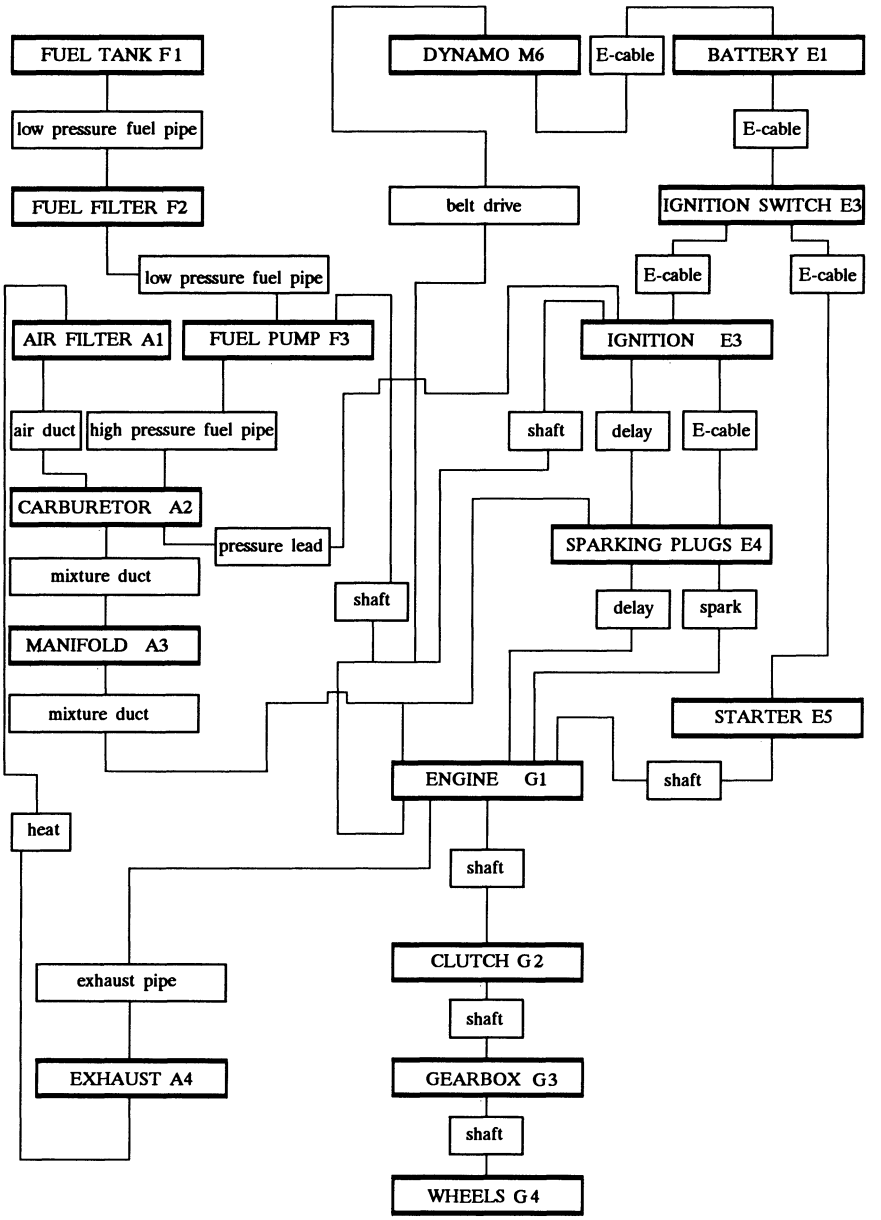



**Figure 18.4** Main model of the automobile engine in SIMUL. The diagram shows the names of the components (in double boxes) and the connections (in single boxes). Connection types have been instantiated to connections, which are in turn carriers of materials. With the knowledge representation of Sect. 18.1 the connections must be replaced by their respective materials.

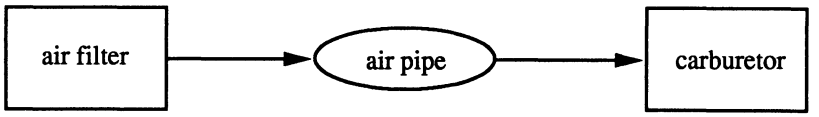



**Figure 18.5** Section of the knowledge base shown in Fig. 18.4

different effects on a parameter SIMUL offers the expert the two basic functions addition (e.g., $-1 + -1 \rightarrow -2$) and maximum (e.g.,$-1 + -1 \rightarrow -1$).

Different levels of abstraction can be represented in SIMUL by replacing a component on one level by a submodel, i.e., a set of components on a lower level. The input and output parameters of the submodel must agree with those of the component. When the submodel is activated, the simulation is no longer carried out with the behavior description of the original component, but with the new components of the submodel. This means that the parameter values calculated on the higher abstraction level are replaced by those calculated on the lower level.

# 19. Statistical Classification

Statistical classification is suitable for classification problems for which a large, representative collection of successfully solved cases exists. The basic idea is similar to that of heuristic classification, except that the relationships between observations and solutions are not estimated by experts but extracted from case data bases with statistical methods. This has the great advantage that the knowledge is less subjective. The limitations of statistical classification result from the fact that a number of conditions must be fulfilled before it can be used. Actually it is possible to use the same approach even if the conditions are not fulfilled, but we would then be dealing with a heuristic method, since the objectivity is lost.

Statistical classification is based on a variant of Bayes' Theorem (see Sect. 8.1), which is used to find the most probable solution for a given set of observations. The conditions for its applicability include the independence of the observations from one another, the completeness of the solution set, the reciprocal exclusion of the solutions, the representativity of the case collection and a sufficient number of cases for each alternative solution. When the first condition is not fulfilled it is sometimes possible to pre-process the observations in such a way that groups of interdependent observations are combined to single independent observations. An incomplete solution set can often be completed simply by adding a solution called "others".

## 19.1 Knowledge Representation

In the simplest case the knowledge representation has the form of a table (array), in which the columns are labeled with the solutions, the first row contains the "*a priori* probabilities" and the remaining rows are labeled with the observations. The entries are the *a priori* probabilities $P(S)$ of the solutions in the first row and the conditional probabilities $P(O/S)$ for the observations in the remaining rows.

If this is transformed into the object-attribute-value representation we have the two object types "observation" and "solution" with the following attributes:[1]

[1] Dynamic attributes are marked (d).

1. Observations:
   - name
   - probability table (a list of all solutions $S_i$ with the conditional probabilities $P(O/S_i)$ for the observation $O$.
   - value (d) (present or absent).
2. Solution:
   - name
   - *a priori* probability
   - probability table (a list of all observations $O_i$ with the conditional probabilities $P(O_i/S)$ for the solution $S$; redundant to the probability tables of the observations)
   - raw value (d) (result of the numerator in the formula of Fig. 19.1)
   - value (d) (result of the quotient in the formula of Fig. 19.1)

Note that in this simple form of knowledge representation the observations only have the value range yes/no, and there is no diagnosis hierarchy. In order to be able to handle numerical observations as well, the data must be pre-processed (see Chap. 15) to convert raw data into yes/no interpretations for the observations. This is true also for symbolically scaled observations (e.g., too low, normal, too high, much too high), which are interdependent, since they cannot simultaneously have the values "too high" and "much too high" and so must also be simplified to yes/no observations.

## 19.2 Knowledge Manipulation

The knowledge manipulation consists of a simple calculation according to a variant of Bayes' Theorem (Fig. 19.1), using the actual observations $O_1 \ldots O_m$ as input. The formula gives the relative probabilities of the alternative solutions from their *a priori* probabilities $P(S)$ and the conditional probabilities $P(O/S)$. If it is required to take into account also the absence of observations, the for-

$$P_r(S_i/O_1 \& \ldots \& O_m) = \frac{P(S_i) \cdot P(O_1/S_i) \cdot \ldots \cdot P(O_m/S_i)}{\sum_{j=1}^{n} P(S_j) \cdot P(O_1/S_j) \cdot \ldots \cdot P(O_m/S_j)}$$

**Figure 19.1** Bayes' Theorem for calculating the relative probability $P_r$ of a solution $S_i$ from a set of $n$ solutions assuming the observations $O_1 \ldots O_m$

mula can easily be modified in such a way that the factor $P(O/S)$ is replaced by $(1 - P(O/S))$ when the symptom is absent.

## 19.3 Knowledge Acquisition

The knowledge acquisition involves the construction and evaluation of large case data bases. Each case in the data base contains a set of observations and the correct solution. The required probabilities are calculated as follows: The *a priori* probability of a solution is given by $P(S) = (\textit{frequency of the solution})/(\textit{total number of cases})$, and a conditional probability by $P(O/S) = (\textit{frequency of the observation-solution pair})/(\textit{frequency of the solution})$. An example is given in Sect. 19.6.

## 19.4 State of the Art

The application of Bayes' Theorem is one of the oldest classification methods and is widely used, especially in medical diagnostics. A comprehensive introduction is given in [Lusted 68]. A frequently cited example for a successful Bayes program is described in [de Dombal 72]. This program differentiated between seven (later nine) different diagnoses for acute abdominal pain, and in a one-year clinical test was found to be right more often than the doctors. However, it turned out that the statistical correlations found in one hospital (in Leeds) did not apply in another (in Copenhagen), where the program was drastically less successful, due to different pre-selections of patients

A difficult problem in knowledge acquisition – apart from maintaining the conditions for the formalism and the representativeness of the case data – is the interpretation of zero as the value of a conditional probability $P(O/S)$. Since multiplication by zero would give a total probability of zero for the corresponding solution, the presence of that particular observation would definitely exclude the solution – only because the observation was not made in connection with the solution in one of the cases already in the data base. Lusted [68] suggests replacing zero by a very small probability in such situations, but this is also unsatisfactory. The only correct way of avoiding this dilemma is to evaluate sufficiently large data bases in which no chance zeros occur. This means, however, that rare solutions for which only a few case examples exist cannot be statistically classified.

On account of the numerous difficulties encountered in the correct application of Bayes' Theorem (independence of the observations, completeness and mutual exclusion of the solutions, representativeness of the case data, existence of enough cases, even for rare solutions), there are few successful Bayes programs. Below we discuss a few variants aimed at facilitating the use of Bayes' Theorem. They mostly involve a certain loss of exactness, so that there is a continuous transition to the "heuristic" methods of treating uncertainties, such as those used in INTERNIST (Sect. 8.3) or MYCIN (Sect. 8.4).

## 19.5 Variants of Bayes' Theorem

Most users cannot easily follow the evaluation of the formula shown in Fig. 19.1, since the raw value of a solution (the numerator of the right-hand side) is meaningful only in relation to the raw values of all the other solutions. Therefore another form is also used in which the *a priori* probabilities of the solutions are updated with each new observation. This step-by-step method is also suitable for dialog control, in order to determine before each step which observation can contribute most to discerning the most probable solution. Finally, so-called Bayesian or probabilistic networks are also used, in which the probabilities propagate from the observations to the solutions via intermediate states.

The basic algorithm of the step-by-step solution method is as follows:
Input: set of observations
Output: most probable solution.

1. Set the current-probability of each solution to its "*a priori* value".
2. For each new observation $O$: multiply the current-probabilities of all solutions $S$ by a "probability quotient".
3. Select the solution with the greatest current-probability.

Two realizations of this algorithm are the use of *a priori* probabilities for the observations and the so-called odds-likelihood form. In the former case the *a priori* value of a solution is just its *a priori* probability $P(S)$, and the probability quotient is given by dividing the conditional probability $P(O/S)$ by the *a priori* probability $P(O)$ of the observation. If the frequency of an observation is increased by the solution, the quotient is greater than 1, so that the current-probability of the solution is increased. If the solution and the observation are independent of one another, the quotient is equal to 1 and leaves the current-probability of the solution unchanged. If the current-probabilities are converted to relative probabilities at the end of this algorithm, it gives the same results as the formula of Fig. 19.1.[2] A disadvantage of this method is that more probabilities must be known.

If instead of calculating relative probabilities the results are interpreted as absolute probabilities for each solution, a theoretical flaw must be accepted: The justification of the algorithm requires the assumptions that the observations are independent, i.e., $P(O2\&O2) = P(O1) \cdot P(O2)$, which is not correct, since $O1$ and $O2$ depend upon the same solution and are therefore not independent (see Sect. 8.1). In [Charniak 85, Sect. 8.2] this algorithm is presented without any discussion of the theoretical problems. In addition it is suggested that the problem of zero probabilities can be solved by putting the quotient $P(O/S)/P(O)$ equal to 1 when $P(O/S) = 0$, so that the probability $P(S)$ is independent of $M$ instead of being reduced by $M$. Although these changes appear to be slight, the

[2]The reason is that in calculating the relative probabilities the *a priori* probabilities $P(O)$ cancel in the formula of Fig. 19.1 when $P(O/S)$ is replaced by the ratio $P(O/S)/P(O)$.

resulting calculating scheme is not longer theoretically sound and must therefore be regarded as "heuristic".

Another variant of the step-by-step solution method is to take the "odds" $OD(S)$ of a solution, defined as $P(S)/P(\neg S)$, as its *a priori* value and to use the "likelihood ratio" $LR(O/S)$ as the probability quotient. The latter is defined as the ratio of the conditional probability of a hypothesis to that of its negation, i.e., $LR(O/S) = P(O/S)/P(O/\neg S)$. In order to exploit also the absence of observations, one can also use negative likelihood ratios, defined by $LR(\neg O/S) = P(\neg O/S)/P(\neg O/\neg S)$. The relationship between the odds and the corresponding probability of a solution is expressed by the formula $OD = P/(1 - P)$ or $P = OD/(1 + OD)$, and the likelihoods are related in the same way to the conditional probabilities of the observations. The interpretation is that a value of 1 means complete uncertainty, i.e., the presence of an observation or the correctness of a solution is just as likely as the opposite. Values greater than 1 are evidence for and values between 0 and 1 evidence against the observation or solution.

The algorithm is based on the following formula:

$$\begin{aligned} OD(S/O) &= \frac{P(S/O)}{P(\neg S/O)} = \frac{P(S)P(O/S)}{P(\neg S)P(O/\neg S)} \\ &= OD(S)LR(O/S). \end{aligned}$$

For several observations $O_1 \ldots O_n$ we have

$$OD(S/O_1 \ldots O_n) = OD(S)LR(O_1/S) \ldots LR(O_n/S).$$

The application of the formula for differentiating between more than two solutions (e.g., $S_1, S_2$ and $S_3$) is problematical, because the probability of $\neg S_1$ varies with the probabilities of the other two solutions $S_2$ and $S_3$ and is not constant as is actually required for the formula to be applicable (see [Szolovits 78]). Nevertheless, the so-called odds-likelihood form has been successfully used in practice for differentiating several solutions, e.g., in PROSPECTOR [Gasching 82]. However, it then belongs in the category of "heuristic" calculation schemes.

The odds-likelihood form is also the basis of the calculation of probabilities in Bayesian networks [Pearl 85,88 Cooper 87], in which the flat structure of Bayes' Theorem with direct reasoning from observations to solutions is abandoned, and solution hierarchies are also allowed. If *a priori* odds exist for all intermediate and final solutions, and positive and negative likelihood ratios for all relationships, the above algorithm can be applied iteratively by regarding the intermediate solutions as being at the same time solutions of one stage and observations of the next stage. Since the intermediate solutions are generally uncertain, and we have considered only certain observations so far, the conditional probabilities must be modified with the probabilities of the intermediate solutions. The consequence is that for each new observation the probability changes must now be transmitted through the whole network. Thus when an intermediate solution is given a new probability the change must be multiplied with all the conditional probabilities and passed on. The expert system PROSPECTOR mentioned above is a successful example for the application of Bayesian networks. [Pearl 88].

## 19.6 Example: FAKTA

An example of a case data base with the option of statistical evaluation according to Bayes' Theorem is provided by FAKTA [Mütter 89]. This system is implemented on the basis of the data base dBase III plus. The case descriptions are entered with a separate dialog program (MED2) and transferred to FAKTA through a coupling program. As an example of a data base (DABAS) we consider the task of selecting a suitable data base for specific user requirements. Suppose a user with an IBM-AT compatible computer is looking for a data base for administration and analysis problems and for the development of software, and that this data base is required to have a software interface to Word Perfect. The frequency distribution reproduced in Fig. 19.2 shows how often other users with the same requirements have used various data bases, where the present user is interested only in the data bases "dBase III", "dBase IV" and "ADIMENS". From this fre-

Table SYMDIAG1-WT1

Period 13.06.89 -- 19.06.89

Calculation of the absolute frequencies:

Others: OPEN-ACCESS-II, PARADOX, RAPID-FILE

| SYMDIAG1-WT1 | F(0) | DBASE-III | DBASE-IV | ADIMENS | OTHERS |
|---|---|---|---|---|---|
| ABS | 18 | 3 | 3 | 3 | 9 |
| DOMAIN = | | | | | |
| administration and analysis | 6 | 1 | 1 | 1 | 3 |
| COMPUTER = | | | | | |
| IBM-AT or compatible | 17 | 3 | 3 | 3 | 8 |
| USER-TOOLS = | | | | | |
| as appl. development | 9 | 3 | 3 | 1 | 2 |
| SOFTWARE-INTERFACE = | | | | | |
| ASCII/Word Perfect | 18 | 3 | 3 | 3 | 9 |

**Figure 19.2** Frequency distribution from FAKTA for the example of choosing a data base

quency distribution the *a priori* probabilities $P$(data-base) and the conditional probabilities $P$(data-base/user-requirement) shown in Fig. 19.3 are calculated. As the result of the application of Bayes Theorem in the form of Fig. 19.1 with the above-mentioned user requirements and the probabilities of Fig. 19.3 we obtain the relative probabilities shown in Fig. 19.4.

The problem of zero probabilities is treated in FAKTA by means of a "counter" which indicates how often the correlation $P(S/O) = 0$ occurred for a given solution. Its interpretation is left to the user. With the parameter "period" it is possible to specify which part of the case data bases goes into the statistics, since old cases may sometimes have to be left out on account of unavoidable knowledge fluctuations (new or modified techniques for collecting observations, new or modified solutions, different case pre-selection). The user can control the independence of the observations and the completeness and mutual exclusion of

Table SYMDIAG1-WT1

Period 13.06.89 -- 19.06.89

Calculation of P(O/S)

Others: OPEN-ACCESS-II, PARADOX, RAPID-FILE

| SYMDIAG1-WT1 | F(O) | P(O) | DBASE-III | DBASE-IV | ADIMENS | OTHERS |
|---|---|---|---|---|---|---|
| ABS | 18 | | 3 | 3 | 3 | 9 |
| P(S) | | | 0.1667 | 0.1667 | 0.1667 | 0.5000 |
| DOMAIN = administration and analysis | 6 | 0.3333 | 0.3333 | 0.3333 | 0.3333 | 0.3333 |
| COMPUTER = IBM-AT or compatible | 17 | 0.9444 | 1 | 1 | 1 | 0.8889 |
| USER-TOOLS = as appl. development | 9 | 0.5000 | 1 | 1 | 0.3333 | 0.2222 |
| SOFTWARE-INTERFACE = ASCII/Word Perfect | 18 | 1 | 1 | 1 | 1 | 1 |

**Figure 19.3** *A priori* and conditional probabilities from Fig. 19.2

Table SYMDIAG1-BAYES1

Period 13.06.89 -- 19.06.89

Others: OPEN-ACCESS-II, PARADOX, RAPID-FILE

| Symptom | Value | F(O) | P(O) |
|---|---|---|---|
| DOMAIN | administration and analysis | 6 | 0.3333 |
| COMPUTER | IBM-AT or compatible | 17 | 0.9444 |
| USER-TOOLS | as appl. development | 9 | 0.5000 |
| SOFTWARE-INTERFACE | ASCII/Word Perfect | 18 | 1 |

| Diagnosis | F(S) | P(S) | P(S/O) | Counter |
|---|---|---|---|---|
| DBASE-IIIPLUS | 3 | 0.1667 | 0.3420 | 0 |
| DBASE-IV | 3 | 0.1667 | 0.3420 | 0 |
| ADIMENS | 3 | 0.1667 | 0.1138 | 0 |
| OTHERS | 9 | 0.5000 | 0.2024 | 0 |

**Figure 19.4** Results of the application of Bayes' Theorem

the solutions by configuring the rows and columns of the tables in Figs. 19.2 and 19.3 himself in order to make a suitable pre-selection. This is possible because the case data base contains not only raw observations and solutions, but also a well-developed structure of data abstractions and intermediate solutions. FAKTA supports the configuration of the tables with menu-like input masks (see Fig. 19.5).

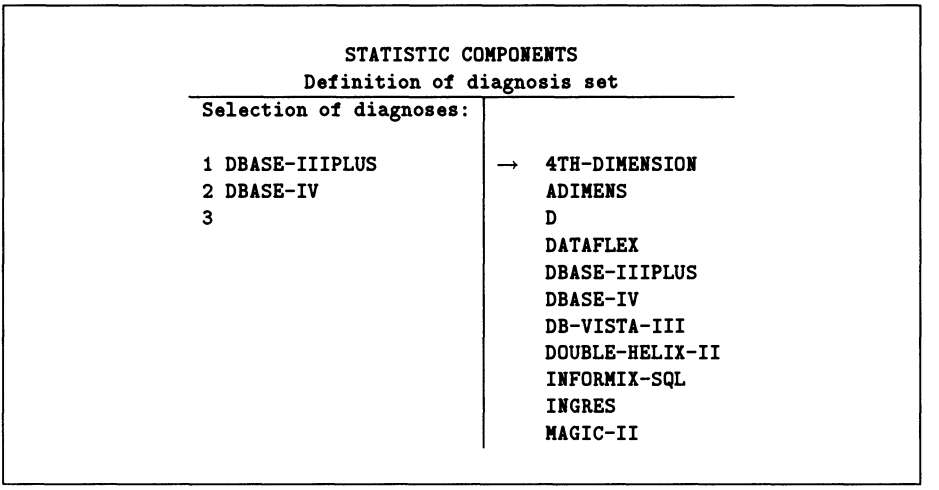



**Figure 19.5** Selection of diagnoses in FAKTA for configuring a table

# 20. Case-Comparing Classification

Case-comparing classification is suitable for classification problems for which there is a large collection of genuine or imagined cases with the correct solutions and additional knowledge, above all for the relative weighting of observations. A new case is solved by first finding the case in the data base whose features (observations) best agree with those of the new case. If the similarity is sufficient, the solution is used for the new case (see Fig. 20.1). In order to evaluate how well the

| | New case | Old case 1 | Old case 2 |
|---|---|---|---|
| Automobile type | make A | make B | make A |
| Mileage | 60,000 | 66,000 | 57,000 |
| Engine vibration | yes | yes | no |
| Does not start | usually | sometimes | always |
| Solution | ? | plugs worn out | battery flat |

**Figure 20.1** Typical situation in case comparing: Which of the old cases has the greater similarity with the new case? Is the similarity great enough to be able to use the solution of the old case?

cases agree a measure of similarity is required. For a given measure of similarity the quality of the solutions improves with each new case in the data base. Since this simplifies the knowledge acquisition, case-comparing classification is theoretically very attractive. However, hardly any practical experience has been gained so far, because the method has not been used for very long in expert systems, and suitable case collections are rare, despite the fact that they have to fulfill less stringent requirements than for statistical classification.

A case is represented by an observation vector, i.e., by a set of observations with their values. Case-comparing classification takes place in three steps:

1. Pre-selection of cases from the case data base.
2. Determination of the similarity between the selected cases and the new case.
3. Interpretation of the differences between the cases and decision whether to accept the solution.

The pre-selection is necessary for reasons of efficiency and can be done with a few key features of the new case. Abnormal observations of the new case are suitable

as key features. The result is a set of comparative cases which agree with the new case in at least some features.

In the second step the agreement with respect to all observations is determined and evaluated. The problem solution depends not only on the number and quality of the stored cases, but also on the measure of similarity used. Starting from a very simple measure of similarity, we show below how its power of discrimination can be increased with additional knowledge. A summary of the numerous aspects which are relevant to the measure of similarity is given in Sect. 20.1.

If all the features carry the same weight and have only the values "present" or "absent", a sufficient measure of similarity is simply the ratio of the number of agreements to the total number of features considered. These two conditions are rarely fulfilled, however.

If the observations have different weights, this should be given as additional knowledge (e.g., in the form of points) to each observation. Thus the observation "engine does not start" would carry a greater weight (e.g., 50 points) than "engine vibrates" (10 points), which in turn is normally more important than the automobile type (1 point). The measure of similarity is then the sum of the points of all agreeing observations divided by the total points for all observations.

So far we have considered only independent yes/no observations. In many applications there are also other types, in particular grouping and scaling. A scaled feature may have a numerical value range (e.g., mileage) or a symbolic one (e.g., "engine does not start": {always, mostly, sometimes, never}). The latter can, however, easily be transformed into a numerical range (always = 1, never = 4). In both cases we determine the similarity factor as a percentage of the weight of the observation. A simple way of calculating a percentage from scaled observations is to divide the smaller value by the larger one, e.g., for milage 60,000/66,000 = 91%. If that is not sufficient, a function $y = f(x)$ can be defined and the quotient $f(x_1)/f(x_2)$ used instead of $x_1/x_2$. The function $f(x)$ may be defined algebraically (e.g., $y = x + 1$) or as a series of linear segments between points given as a set of $x/y$ values (e.g., 0/0 10/1 100/2).

However, a consequence of all these methods is that a given difference leads to less similarity when the absolute values are smaller. Two milages of 6,000 and 12,000, for example, would have a similarity of only 50%, compared with the above value of 91%, although the absolute differences are equal. If this not desired, a constant $K$ can be given for calculating the similarity according to the formula $1 - |x_1 - x_2|/K$ or $1 - |f(x_1) - f(x_2)|/K$. With $K = 60{,}000$ this would give 90% for both the above milage differences.

A grouped observation is one for which some of the possible values are similar in meaning, e.g., automobile type $\{A, B, C, D\}$, where $A$, $B$ and $C$ are very similar to each other, and $C$ and $D$ have only a little similarity. In order to take into account such similarities, a grouping type, which groups together similar alternatives, is useful. If it is desired to distinguish between identity and belonging to the same group, a similarity factor can be given. A possible method of representing this for the above example is (((A,B,C) 0.9)((C,D) 0.5)), which has the following

meaning: if two cases have the same value (A,A), (D,D), etc., the similarity is 100%, for (A,C) it is 90%, for (C,D) 50% and for (A,D) 0%.

So far we have assumed that the different values of an observation all have the same weight. Generally one would like to give more weight to an agreement of rare values than frequent values. In fault finding rare values are the deviations from the normal values. Thus if two cases agree in many deviations they are more similar than when they agree in many normal values. This differentiation is important, in particular because it should make no difference in diagnostics whether we look only for the deviation from normal behavior in a particular case or perform systematically all possible tests. Since most observed values will be normal for a faulty system or a sick patient they should not be given much weight in case comparison in order to increase the discriminating power.

We therefore introduce the concept of abnormality in order to give an observation more weight, the more abnormal it is. In the case of yes/no observations this may mean that weighting points are awarded only when there is agreement about the *presence* of a symptom. If, on the other hand, two cases agree in that the symptom is *absent* in both (e.g., the engine does not vibrate), no points, or much fewer, are given. The concept of the weight of an observation introduced above is therefore not regarded as a constant, but is modified according to the abnormality, which depends upon the values. Abnormalities are of course not required for all observations – the automobile type, for example.

A further complication is introduced if the weight of an observation is influenced by external factors. This is typical, e.g., for precedence selection, where the user can determine the relative weights of the observations, which represent his wishes, thus over-writing any default weights. Another type of external weighting is required when knowledge about special features of the solutions is to be taken into account, so that certain observations may carry different weights for different solutions. For example, the sex of the patient is not significant for most medical diagnoses, but it can be a criterion for the exclusion of special diagnoses such as pregnancy. This can be taken into account by dynamic assignment of weights to the observations: if the case chosen for comparison has such a special diagnosis, the weights associated with it over-write the default values of the observations (e.g., in the case of suspected pregnancy the weight of the sex is changed from "0" to "−1000").

A more detailed interpretation of differences is possible with the aid of additional causal knowledge if one looks for a causal explanation of the differences. The significance of a difference is then determined according to the nature of the explanation. For this purpose it is not necessary to construct an entirely new model as in causal classification (Chaps. 17 and 18), but only to adapt a model which already exists for the old case to the new case. This aspect will not be treated further here (see, however, CASEY in Sect. 20.4).

So far we have regarded only the raw data of the cases as observations. The quality of case comparison can be considerably improved if pre-processed data abstractions (e.g., qualitative abstractions, calculated characteristic values, etc.) are compared instead. Their derivation corresponds to the data pre-processing

of heuristic classification (see Chap. 15). Care must be taken, however, that an observation (e.g., milage) is not used twice, e.g., both as a raw value (numerical mileage) and also as a qualitative abstraction (e.g., milage assessment: new engine, used engine, old engine, very old engine), in a case comparison. Each observation is therefore allotted one or more levels of abstraction. In a concrete case comparison the level of abstraction is first specified, and only observations on this level are considered. The higher the level of abstraction, the greater is the apparent similarity of the cases. High levels of abstraction are therefore suitable for pre-selection of cases. A slight disadvantage of abstraction is that the abstract concepts must first be derived from the raw data, for which further additional knowledge is required. This becomes particularly clear if one considers the most extreme form of abstraction, in which a case is compressed to a diagnosis, i.e., completely solved.

## 20.1 Knowledge Representation

For the simplest form of case comparison without additional knowledge it is sufficient to represent observations with the attributes "name" and current "value" (present, absent, unknown) and solutions with the attributes "name" and current "value" (right or wrong). In addition we require the object type case with the attributes "observations" (list of observations) and "solutions". For efficient pre-selection the cases should be stored in a data base or indexed according to important observations.

The additional knowledge for case comparison may be summarized under three headings:[1]

1. Level of abstraction: The level of abstraction on which the comparison is to be performed is specified externally, and only those observations are considered which belong to this level as specified in their attribute "level of abstraction".

2. Weights: For calculating the significance of an observation three attributes are required:

   - Weight: a number indicating the significance of the observation.
   - Case-specific-weight: a number, originating from the user or from the solution of the comparative case, which overwrites the weight. In the latter case all "case-specific-weights" must be recorded in an additional attribute "significance profile" of the solution.
   - Abnormality: a list of numbers which modify the weight as a function of the value of the observation.

[1]The measure of similarity is described in some detail here (unlike heuristic evidence calculi, for example), because we have so far found no comparable description in the literature (cf. Sect. 20.4), and we consider the differentiation presented here necessary for an acceptable expressive power.

3. Similarity data: the specification of a type and the contents for each observation, on the basis of which the relative similarity between observations may be calculated with a type-specific formula. The result is a number between 0 and 100%, or possibly a negative number for very great dissimilarity. The relative similarity is multiplied by the weight of the observation.

The methods introduced in the previous section for calculating the similarity are described in detail below according to the following scheme: The methods are classified according to the value range types of the observations (yes/no, numerical, one-choice, multiple-choice), and the possible similarity types. For each type the contents are described by giving the representation, a default value if applicable, the similarity formula and an example (the values from the new and old cases are denoted by $A$ and $B$, respectively, with $A \leq B$):

- Yes/no: no special representation required. *Similarity formula:* If $A = B$ 100%, otherwise 0%.
- Numerical: there are four subtypes:
  - Functional: *Representation:* a function $y = f(x)$. *Default:* if no function is given the identity $f(x) = x$ is assumed. *Similarity formula:* $f(A)/f(B)$. *Example:* With $f(x) = x + 1$, $A = 0$, $B = 1$ we obtain a similarity of $0.5 \mathrel{\widehat{=}} 50\%$.
  - Functional, constant denominator: *Representation:* a function $y = f(x)$ and a constant $K$. *Default:* if no function is given the identity $f(x) = x$ is assumed; if the constant $K$ is not given the difference between the upper and lower limits, if given, is assumed. *Similarity formula:* $1 - (f(B) - f(A))/K$. *Example:* With $f(x) = x$, $A = 15$, $B = 20$, $K = 100$ we obtain a similarity of $0.95 \mathrel{\widehat{=}} 95\%$.
  - Segmented: *Representation:* A set of $x/y$ value pairs which are interpreted as points on a function $y = g(x)$ which is taken to be linear between these points. *Similarity formula:* $g(A)/g(B)$. *Example:* with the point set (0/0 10/10 20/15 30/18 50/20 100/21), $A = 40$, $B = 100$ we obtain a similarity of $19/21 \approx 90\%$.
  - Segmented, constant denominator: *Representation:* A set of $x/y$ value pairs which are interpreted as points on a function $y = g(x)$ which is taken to be linear between these points, and a constant $K$. *Default:* If the constant $K$ is not given, the difference between the greatest and smallest $y$ values of the set of points is used. *Similarity formula:* $1 - (g(B) - g(A))/K$.
- One-choice (exactly one alternative from a set of given values): here there are four subtypes:
  - Individual: no special representation required. *Similarity formula:* If $A = B$ then 100%, otherwise 0%.

- Grouped: *Representation:* Set of value groups with similarity assessments. *Default:* If no assessments of similarity are given, the value $1 \mathrel{\widehat{=}} 100\%$ is assumed. *Similarity formula:* If $A = B$ then 100%, else if $A$ and $B$ are in the same group then similarity of this group, else 0%. *Example:* with the set $((X1, X2)((X3, X4)0.9))$, $A = X3$, $B = X4$ we obtain a similarity of $0.9 \mathrel{\widehat{=}} 90\%$.
- Scaled: *Representation:* list of numbers and a constant $K$. The length of the list must be equal to the number of possible values for the observation. *Default:* If no list of numbers is given, the list (1, 2, 3, ...) is used. If the constant $K$ is not given, the difference between the largest and smallest value in the list is used. *Similarity formula:* $1 - (\text{list-entry}(B) - \text{list-entry}(A))/K$. *Example:* With the value set {normal, high, too-high, much-too-high}, the list (0 10 60 80), $A =$ high, $B =$ too-high we obtain a similarity of $30/80 \approx 37\%$.
- Scaled, functional: *Representation:* List of numbers. The length of the list must be equal to the number of possible values for the observation. *Default:* If no list of numbers is given, the list (1, 2, 3, ...) is used. *Similarity formula:* $\text{list-entry}(B)/\text{list-entry}(A)$.

- Multiple-choice (any number of alternatives from a set of given values): This type of question corresponds approximately to a collection of many yes/no questions. There three subtypes:
  - Simple: no special representation required. *Similarity formula:* Agreements are first counted for each alternative answer as for yes/no questions, and then the quotient "number of agreements"/"number of comparisons" is formed. *Example:* Suppose we have $\{A, B\}$ in the test case and $\{B, D\}$ in the case used for comparison. First, three independent comparisons are made: $(A, -) = 0$, $(B, B) = 1$, $(-, D) = 0$. From this we obtain a relative similarity of $1/3 \approx 33\%$.
  - Grouped: *Representation:* set of disjunctive value groups with similarity assessments. *Default:* If no assessment of similarity is given, the default value $1 \mathrel{\widehat{=}} 100\%$ is assumed. *Similarity formula:* Agreements are first counted for each alternative answer, each group being regarded as a unit, and then the quotient "number of agreements"/"number of comparisons" is formed. *Example:* With the value groups $((A, B)0.9)((C, D)0.8)E)$ and the values $\{A, B, D, E\}$ in the test case and $\{B, C\}$ in the old case we have $((A, B), B) = 1$, $(D, C) = 0.8$, $(E, -) = 0$ and the hence similarity is $1.8/3 \mathrel{\widehat{=}} 60\%$.
  - Independent: *Representation:* set of disjunctive value groups with similarity assessments. *Default:* If no groups are specified, each alternative is a group. If no similarity assessment was given for a group, the default value $1 = 100\%$ is assumed. *Similarity formula:* Agreements are counted separately for each alternative, each group being regarded as a unit, but the counts are multiplied immediately

with the weights of the observations. *Example:* With the value groups $((A, B)0.9)((C, D)0.8)E)$ and the values $\{A, B, D, E\}$ for the test case and $\{B, C\}$ for the old case we have $((A, B), B) = 1$, $(D, C) = 0.8$, $(E, -) = 0$, and the total similarity is expressed through the three values 100%, 80% and 0%, as if three independent yes/no observations had been compared.

We finally obtain the following knowledge representation for case comparing classification, where the knowledge for deriving data abstractions is not given (it corresponds to that of heuristic classification, see Chap. 15):[2]

1. Observations

   - name
   - value-range-type (yes/no, numerical, single-choice or multiple-choice)
   - value-range
   - abstraction-level (number, or set of numbers, indicating the level of abstraction for the comparison)
   - weight (number)
   - abnormality (list of numbers, one for each alternative value, which can be used to modify the weight)
   - similarity-type (if value-range-type = numerical then {functional, functional-constant-denominator, segmented, segmented-constant-denominator}, if one-choice then {individual, grouped, scaled, scaled-functional}, if multiple-choice then {simple, grouped, independent}; interpretation see above)
   - similarity-representation (depends on similarity-type, see above)
   - value (d) (current value of observation in the new case)
   - comparison-value (d) (value from old case)
   - case-specific-weight (case-specific weight which can overwrite the normal weight)
   - maximum-similarity-points (d) (the maximum attainable weight in the comparison of this observation)
   - relative-similarity (d) (a number between 0 and 1 representing the similarity of the values of this observation for the new and old cases; for the similarity type "multiple-choice independent" there may be several numbers)
   - similarity-points (d) (actual number of points, given – except for the similarity type "multiple-choice independent" – as the product *maximum-similarity-points*× *relative-similarity*)

[2]Dynamic attributes are marked (d).

2. Solutions
   - name
   - significance profile (list of observations with weights)
3. Cases
   - name
   - set of observations with values
   - solution
   - similarity (d) (number between 0 and 1 representing the result of the similarity test. It is given by the quotient ($\sum$ *similarity-points*)/($\sum$ *maximum-similarity-points*), the sums being taken over all observations of the two cases.)

## 20.2 Knowledge Manipulation

In describing the problem-solving method we will assume that the cases are stored in a data base. If the stored cases contain only raw data and no data abstractions, the latter must be calculated for all stored cases if they are required for the comparison. The basic algorithm consists of the following steps:
Input: observations of a new case.
Output: solution of a case which is sufficiently similar in a relative and absolute sense.

1. Processing the raw data of the new case to data abstractions and global determination of the "abstraction level". Thereafter only those observations are considered which have a valid "abstraction level".
2. Extraction of cases from the data base which fulfill certain key criteria. Observations of the new case which have a high weight are suitable as key criteria.
3. Selection of an old case for comparison and entry of its observation values in the corresponding attributes "comparison-value". If a "significance-profile" exists for the solution of the old case, its values are taken into the "case-specific-weight" of the observations affected. The user can also manipulate the case-specific-weight of observations.
4. For each observation of the new case and the old case:
   4.1 Calculation of the maximum-similarity-points. They are taken from the "case-specific-weight", or if this is not present from the "weight", and as "raw weight" can be modified by the "abnormality".[3]

[3]The raw weight is the maximum possible weight and applies to the case of maximum abnormality. For smaller abnormalities it is reduced as follows: With the current data "value"

4.2 Calculation of the relative-similarity: The calculation of the relative similarity between "value" and "comparison-value" is performed with the similarity formula of the "similarity-type", using the "similarity-representation".

4.3 Calculation of the similarity points, usually given by the product: *relative-similarity*× *maximum-similarity-points*. The "similarity type" multiple-choice independent is a special case[4].

5. Calculate the similarity between the two cases as the sum of all the similarity points divided by the sum of all maximum-similarity-points for all the observations of both cases.

6. Repeat the steps 3 to 5 for all comparison cases extracted in the pre-selection stage, choose the old case with sufficient similarity (or the case with the maximum similarity) and take over the solutions.

The end result is a list of the solutions of the old cases with the highest similarity quotients, provided that they are greater than a certain threshold value. If no suitable cases are available in the case data base, it can be filled with some imaginary "typical" cases. In this way the typical cases at least can be solved even without a collection of genuine cases.

## 20.3 Knowledge Acquisition

The main part of the knowledge acquisition is the collection of cases (i.e., sets of observations) and their correct solutions. It may be useful to specify a set of typical cases, in order to assure a certain minimum performance. Case-comparing classification is more robust than statistical classification, because the result cannot be falsified by zero probabilities.

---

and "comparison-value" the corresponding number is selected from the list of abnormalities according to the value range and similarity representation. For example suppose the value is "too high" and the comparison-value "high" with the value range {normal, high, too high, much too high} and the abnormalities (0,10,60,80); this would give the numbers 60 and 10. Of these two abnormality numbers the larger is taken. (One could also take the number belonging to the "value", which would be more efficient, because it would not be necessary to calculate the abnormality anew for each comparison case, but this generally leads to less accurate results.) This number is divided by the largest number in "abnormality" (in the example 80) and the quotient is multiplied by the raw points to give the maximum-similarity-points. (If in the example the number of raw points is 20, then there would be $20 \cdot 60/80 = 15$ maximum-similarity-points in this case.) For numerical observations with segmented similarity type, the similarity can be calculated in a similar way.

[4]For the "similarity type" multiple-choice independent several products are formed. The "abnormality" list then corresponds to value range in such a way that each group of the latter has its own weight. Suppose, for example, that an observation has the value range {A,B,C,D,E,F}, the similarity representation ((A,B)(C,D)EF), the value (A,E) and the comparison value (B), giving the relative similarity (1,-,0,-), and let the abnormality list be (40,10,20,10). Then the maximum-similarity-points are $40 + 20 = 60$ and the actual similarity points 40.

However, additional knowledge is necessary for high efficiency. Since this knowledge is mainly of a heuristic nature, it must come from experts. The assignment of those attributes of observations which are typical for case-comparing classification, namely "weight", "abnormality", "similarity type", "similarity-representation" and "level of abstraction", can be carried out in a table. For entering the significance profiles of the solutions, tables with the observations as rows and the solutions as rows are suitable.

Finally the knowledge for pre-processing the observations must be specified. This input does not differ from that of heuristic classification.

The provision of the typical cases for the case data base leads to an incremental knowledge acquisition, starting with the typical appearance of the diagnoses and extending the knowledge of the possible variations of a diagnosis a small amount with each new case.

## 20.4 State of the Art

The simplest form of case-comparing classification is a normal data base inquiry directed to a case data base with selectors (observations) of which all or as many as possible must match. Such systems are in wide use, e.g., in libraries, where the author, the year and the key words are observations and the books the solutions, or in medical information systems such as the large rheumatology data base ARAMIS [Fries 72]. However, since only a very simple measure of similarity is used, the problem solution quality is very limited and corresponds to our preselection. The final test of similarity is left to the user.

In artificial intelligence case-comparing reasoning has become popular only recently. Some approaches are based on construction problems (see Chap. 27). In diagnostics there are as yet no known systems which have proved successful in routine practice. Presumably the reason is the lack of suitable, large case data bases, without which the power of case-comparing approaches cannot be demonstrated. First approaches to case-comparing classification can be found, e.g., in [Kolodner 85], [Koton 88] and [Porter 90]. In [Kolodner 85] the importance of the experience of previous cases for medical diagnostics is discussed in detail, but in the technical part the case comparison is performed on a relatively simple, abstract level, and diagnoses can also be used as descriptors (key observations), e.g.: "Look for cases with the diagnosis $X$ in which there were unexplained observations."

CASEY [Koton 88] is a case-comparing diagnosis system based on Long's causal diagnostic system (see Sect. 17.4). CASEY compares a new case systematically with all cases in the data base, using the significance of the observations of the stored cases (which corresponds to our "case-specific-weight"). This does not have to be estimated by experts, however, but is determined from the causal explanation for the solution of the old case: observations which play a role in the explanation are important, the rest unimportant. CASEY concentrates first on the comparison of abstract observations. Since in addition only yes/no observations are used, CASEY needs only a simple measure of similarity. If two

cases agree on an abstract level but not on a concrete level, CASEY tries to find an explanation for the difference by calling the causal diagnostic system. If this is successful, there is no reason for not adopting the solution for the new case, otherwise the difference is evidence against the similarity of the two cases.

In [Kolodner 85], [Koton 88] and [Porter 90] details of the measure of similarity used are unfortunately not given. There is also no mention of a differentiation between different types of observations and in particular the concept of abnormality.

## 20.5 Example: CcC

CcC (case-comparing component) [Hestermann 90][5] is a component for case comparison designed as an extension of the heuristic diagnostic shell MED2 (see Sect. 15.5) and the case data base FAKTA (see Sect. 19.6). As an example for the application of CcC we consider the comparison of workpieces in order to determine whether a similar workpiece has already been manufactured before drawing up the production plan for a new one (cf. ExAP, Sect. 25.5).

CcC uses the same knowledge representation as MED2 for observations with the different value range types (yes/no, one-choice, multiple-choice, numerical) and the mechanisms for data abstraction, and also the same dialog control. In addition it provides the specific case-comparing attributes (see Sect. 20.1) and the evaluation algorithm (see Sect. 20.2). The construction of a case library and its evaluation take place in CcC in the following steps:

1. Construction of a knowledge base for the domain:
   - 1.1 Definition of the observations and abstractions, and setting up the required attributes.
   - 1.2 Input of a set of cases with the correct solutions.
   - 1.3 Storing the cases in the case data base FAKTA.
2. Input of the symptoms of a new case by the user.
3. Pre-selection of comparative cases from FAKTA:
   - 3.1 Determination of the key observations. These are primarily observations concerning the number and type of the machining elements.
   - 3.2 Selection of a definite number of cases from FAKTA which agree with as many of the key observations of the new case as possible.
4. Comparison with each of the selected cases with the new case:
   - 4.1 Determination of the correspondence between the observations of the new case and the old case. (This step is necessary since in the case of

[5]The successor system CcC+ is described in [Puppe 91].

workpieces having several machining elements of the same type, e.g., several steps, it is not clear which elements fit one another).

4.2 The actual comparison.

5. Output of the most similar cases for the user.

The way CcC works is illustrated below with the example of the comparison of two workpieces which have only the machining element "step" (see Fig. 20.2).

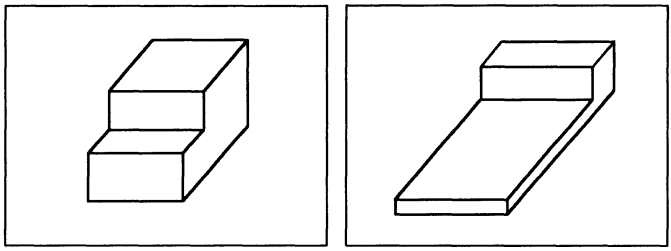


**Figure 20.2** Two workpieces with the machining element "step"

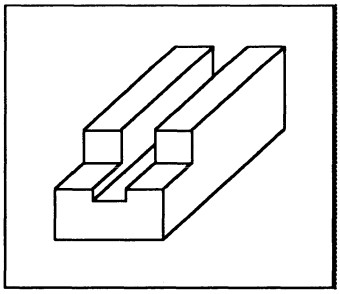


**Figure 20.3** Workpiece 3 with two machining elements "step" and "groove". It has a similarity of only 60% and 55%, respectively, to the two workpieces shown in Fig. 20.2.

Part of the workpiece knowledge base with the observation class "general workpiece data" and a class for the machining element type "step" is shown in Fig. 20.4. The values for the two workpieces shown in Fig. 20.2 are given together with the result of the comparison in Fig. 20.5.

The similarity of the two workpieces shown in Fig. 20.2 is 92%, but their similarities to the one depicted in Fig. 20.3 is much less on account of the additional groove, amounting only to 60% (Fig. 20.2 left to Fig. 20.3) and 55% (Fig. 20.2 right to Fig. 20.3).

| Name | Type | Weight | Abnormality | Similarity data |
|---|---|---|---|---|
| General workpiece data | | | | |
| WP-ident.-No. | str | 1 | — | — |
| WP-name | str | 1 | — | — |
| WP-length | num | 10 | — | — |
| WP-width | num | 10 | — | — |
| WP-height | num | 10 | — | — |
| WP-mach.-class | num | 50 | — | — |
| WP-material | str | 1 | — | — |
| WP-blank[1] | oc-i | 50 | — | — |
| WP-protection[2] | mc | 10 | — | — |
| WP-treatment[3] | mc | 10 | — | — |
| WP-tolerance[4] | oc | 10 | — | — |
| WP-surface[5] | oc | 10 | — | — |
| WP-form[6] | oc | 500 | (2 1 2 2 ) | (1 2 4 10) |
| WP-volume | num-s | 200 | (3 2 1 2 5) | $x$(0 1000 $10^4$ $5 \cdot 10^5$ $10^7$ $10^9$)<br>$y$(0 1000 $10^4$ $10^5$ $10^6$ $10^7$) |
| WP-mach.-degree | num | 50 | — | $y = x + 1$ |
| WP-No.-ME-total | num | 200 | — | $y = x + 1$ |
| WP-No.-ME-top | num | 100 | — | $y = x + 1$ |
| WP-No.-ME-front | num | 100 | — | $y = x + 1$ |
| WP-No.-ME-right | num | 100 | — | $y = x + 1$ |
| WP-No.-ME-left | num | 100 | — | $y = x + 1$ |
| WP-No.-ME-back | num | 100 | — | $y = x + 1$ |
| WP-No.-ME-bottom | num | 100 | — | $y = x + 1$ |
| Step | | | | |
| S-ME-Type | oc-i | 1 | — | — |
| S-ident.-No. | str | 1 | — | — |
| S-length | num | 10 | — | — |
| S-base-width | num | 10 | — | — |
| S-side-width | num | 10 | — | — |
| S-envelope-1[7] | oc-i | 100 | — | — |
| S-envelope-2[7] | oc-i | 100 | — | — |
| S-contains[8] | mc | 50 | — | — |
| S-lies-in[8] | mc | 50 | — | — |
| S-intersects[8] | mc | 50 | — | — |
| S-is-next-to[8] | mc | 50 | — | — |
| S-surface[5] | oc | 10 | — | — |
| S-form[6] | oc | 200 | (2 1 2 2) | (1 2 4 10) |
| S-volume | num-s | 50 | (3 2 1 2 5) | $x$(0 1000 $10^4$ $5 \cdot 10^5$ $10^7$ $10^9$)<br>$y$(0 1000 $10^4$ $10^5$ $10^6$ $10^7$) |

1 Forging, casting, rod material.
2 Paint, stove enamel, enamel, zinc.
3 Quenched, case hardened, normalized, annealed.
4 IT-06 – IT-10.
5 Untreated, roughed, smoothed, ground.
6 Plate, block, bar, angle.
7 Top, front, right, left, back, bottom.
8 A machining element, e.g., step, boring or groove.

**Figure 20.4** Knowledge base for the observation classes "general workpiece data" and "step". The value ranges of the observations are given in the footnotes. Abbreviations: see caption to Fig. 20.5. The value range type and the similarity type are given in the second column: num = numerical functional, num-s = numerical segmented, oc = one-choice scaled, oc-i = one-choice individual, mc = multiple-choice simple, str = string (yes/no). The level of abstraction is not given. In the last column "—" means the default for numerical-functional observations, i.e., the identity function $y = x$.

| Name | Value-1 | Value-2 | Similarity (%) | Actual points | Maximum points |
|---|---|---|---|---|---|
| Workpiece | | | | | |
| WP-ident.-No. | HAB-01 | HAB-02 | 0 | 0 | 1 |
| WP-name | step | shallow step | 0 | 0 | 1 |
| WP-length | 50 | 80 | 62 | 6.2 | 10 |
| WP-width | 30 | 30 | 100 | 10 | 10 |
| WP-height | 25 | 15 | 60 | 6 | 10 |
| WP-mach.-class | 2 | 5 | 40 | 20 | 50 |
| WP-material | MAT.2756 | MAT.2756 | 100 | 1 | 1 |
| WP-blank | casting | forging | 0 | 0 | 50 |
| WP-protection | (paint) | (paint) | 100 | 10 | 10 |
| WP-treatment | (case hardened) | (annealed) | 0 | 0 | 10 |
| WP-tolerance | IT-08 | IT-08 | 100 | 10 | 10 |
| WP-surface | roughed | roughed | 100 | 10 | 10 |
| WP-form | block | block | 100 | 250 | 250 |
| WP-volume | 37500 | 36000 | 98 | 39.2 | 40 |
| WP-mach.-degree | 86.4 | 41.6667 | 49 | 24.5 | 50 |
| WP-No.-ME-total | 1 | 1 | 100 | 200 | 200 |
| WP-No.-ME-top | 1 | 1 | 100 | 100 | 100 |
| WP-No.-ME-front | 1 | 1 | 100 | 100 | 100 |
| WP-No.-ME-right | 0 | 0 | 100 | 100 | 100 |
| WP-No.-ME-left | 0 | 0 | 100 | 100 | 100 |
| WP-No.-ME-back | 0 | 0 | 100 | 100 | 100 |
| WP-No.-ME-bottom | 0 | 0 | 100 | 100 | 100 |
| Sum | — | — | 90 | 1186.9 | 1313 |
| Step-01 | | | | | |
| S-ME-Type | 1 | 1 | 100 | 1 | 1 |
| S-ident.-No. | STEP-01 | STEP-01 | 100 | 1 | 1 |
| S-length | 30 | 30 | 100 | 10 | 10 |
| S-base-width | 17 | 70 | 24 | 2.4 | 10 |
| S-side-width | 10 | 10 | 100 | 10 | 10 |
| S-envelope-1 | top | top | 100 | 100 | 100 |
| S-envelope-2 | front | front | 100 | 100 | 100 |
| S-contains | FALSE | FALSE | 100 | 50 | 50 |
| S-lies-in | FALSE | FALSE | 100 | 50 | 50 |
| S-intersects | FALSE | FALSE | 100 | 50 | 50 |
| S-is-next-to | FALSE | FALSE | 100 | 50 | 50 |
| S-surface | roughed | roughed | 100 | 10 | 10 |
| S-form | block | block | 100 | 100 | 100 |
| S-volume | 5100 | 21000 | 42 | 4.2 | 10 |
| Sum | — | — | 97 | 538.6 | 552 |
| Sum overall | — | — | 92 | 1725.5 | 1865 |

**Figure 20.5** Description and comparison of the two workpieces from Fig. 20.2. Abbreviations: WP = workpiece, S = step, ME = machining element, mach. = machining

# Part V

# Construction

# 21. Review of the Problem-Solving Type Construction

In construction problems the solution cannot be selected as in classification, but must be put together from solution elements. In a wide sense of the word every problem may be regarded as a construction problem, classification being a simple special case. It is therefore not realistic to expect a more or less complete division into problem-solving classes with strong problem-solving methods. As for classification we will attempt a systematization according to three classification principles, although the knowledge type plays a less dominant role in this case:

1. Problem type: configuration, assignment, planning.
2. Knowledge type: heuristic, case-comparing, model-based.
3. Problem-solving method: (search strategies, skeletal construction, propose-and-revise, least commitment.

## 21.1 Domains

In order to explain the construction techniques let us consider some typical problems: generation of timetables, configuration of elevators or computers, planning experiments in molecular genetics, production scheduling and machine capacity utilization in manufacturing, planning in the "Blocks World" and therapy planning in the treatment of cancer. In each case we give the problem $P$ (i.e., the initial situation with the available elementary objects and the requirements placed on the solution), the type of solution sought $S$, the solution space $SS$ (i.e., the theoretically possible combinations of the elementary objects without regard to the requirements) and an example expert system $XPS$.

- Generation of timetables: $P$: classrooms, teachers, classes, subjects, times and the teaching duty plan laying down which teachers give which subjects to which classes and for how many periods, e.g., teacher Smith gives three periods of English to class 3a, two periods of English to class 4b, etc., and any special boundary conditions. $S$: the timetable showing when the lessons are held and in which rooms. $SS$: all conceivable timetable constellations. $XPS$: REST [Poeck 90].

- Configuring elevators: $P$: requirements on the elevator (e.g., height, width, depth, maximum load). $S$: determination of all the elevator parameters (e.g., number and strength of the cables, type of doors, materials, costs). $SS$: all possible combinations of the parameters. $XPS$: SALT/VT [Marcus 88].

- Configuring computers: $P$: die main components of the computer. $S$: a layout diagram showing the arrangement of the components in the housing and which auxiliary components (e.g., cables) are required in addition. $SS$: all constellations of the components. $XPS$: R1-XCON [McDermott 82, 84, Barker 89].

- Planning experiments in molecular genetics: $P$: the generally available substances and organisms, the possible actions and the aim of the experiment (e.g., insulin production). $S$: a sequence of actions to achieve the aim and the specification of the substances and organisms needed for the experiment. $SS$: all conceivable plans. $XPS$: Friedland's MOLGEN [Friedland 79, 85].

- Production planning of workpieces: $P$: the blank, the required shape and properties of the workpiece and the possible working actions (e.g., boring, grinding, painting, etc.). $S$: an economical sequence of manufacturing steps which transforms the blank into the required state. $SS$: all working plans. $XPS$: ExAP [Dyckhoff 89].

- Effective use of machine capacity: $P$: a set of blanks, for each blank a specification of the manufacturing steps and the information as to which machines are available for each step. $S$: an (optimal) machine occupancy plan. $SS$: all possible machine occupancy plans. $XPS$: ISIS [Fox 83, Smith 86].

- Blocks World: $P$: a set of building blocks in a given arrangement, a desired final arrangement and a robot with a set of possible actions. $S$: a sequence of actions for producing the desired arrangement of the blocks. $SS$: all conceivable sequences of actions. Example: NOAH [1] [Sacerdoti 77].

- Therapy plans for cancer treatment: $P$: the condition of the patient and the aims of the therapy. $S$: specification of the therapeutic measures (e.g., medicines, radiation treatment, operations) for achieving the aims. $SS$: all conceivable combinations of therapeutic measures. $XPS$: ONCOCIN/OPAL [2] [Hickam 85/Musen 87].

[1] Since planning programs for the Blocks World like NOAH do not contain knowledge from experts, they are not really expert systems.

[2] Actually ONCOCIN/OPAL is not a planning system; OPAL provides a graphical description language for therapy plans, and OPAL controls the execution of the therapy plans.

## 21.2 Problem Types

A division into problem types is more difficult for construction than for classification, since the spectrum of possible applications is very large, inhomogeneous and relatively poorly understood. From the user's point of view a preliminary classification gives the problem types configuration, assignment and planning. *Configuration* includes problems in which basic elements are selected from a given set, parametrized and assembled to form a solution object which fulfills the desired requirements. *Assignment* is the mapping of one set of objects on another, taking into account restrictions. A special case of assignment is *scheduling*, in which objects are mapped onto time intervals. The object of *planning* is to find a sequence of operators which transform a given initial state into a desired final state.

Assignment differs from configuration and planning in that its elementary objects are given and completely characterized in themselves, but their correspondence to other objects must be determined. In configuration the value ranges of the elementary objects is predominantly numerical or symbolic (e.g., in elevator configuration cable strength = 600 kg), whereas they consist of other objects in assignment (in timetable generation: a lesson is assigned a standard interval of time and a classroom, e.g., the lesson "1 period English by Mr. Smith in class 3b" might take place on "Monday in the first period" in room "R267").

Planning can be regarded as a special case of configuration, in which the operators, treated as elementary objects, must fulfill certain restrictions, i.e., they must be applicable in the particular situation and must collectively lead to the desired state. This point of view is reflected in common use of language: planning a journey, for example, is the same as configuring a travel plan. On the other hand the restrictions of planning problems are fairly complex in comparison with typical configuration problems. The applicability of the operators and the order in which they are applied play an important role in planning, whereas it is usually unnecessary – or very simple – to consider these factors in the arrangement of the elementary objects in configuration. We will therefore limit the concept of configuration to configuration without complex restrictions, and problems with such restrictions will be regarded as a special problem type like planning. Fig. 21.1 summarizes the elementary objects, the problems, the types of solutions, the solution spaces and the examples discussed above for planning, configuration and assignment.

## 21.3 Analysis of Problem Characteristics

On account of the generality of the problem-solving type construction, a universal method of solution would be a "general problem solver" and hence not particularly useful. Strong problem-solving methods are therefore to be expected only in special fields of application with limited complexity. Some possible limitations are described below. When a domain is characterized, the aim is to show that

| | Planning | Configuration | Assignment |
|---|---|---|---|
| Elementary objects | objects with attributes and (generic) operators with preconditions and actions | basic objects with attributes and relationships to one another | at least two disjunctive sets of objects |
| Problem | transformation of a given initial state into a desired final state (states are object constellations) | selection, parametrizing and aggregation of basic objects to the solution object such that the requirements are fulfilled | provision of an ordering plan which takes into consideration preferences, short resources and other restrictions |
| Solution | (optimal) sequence of applicable operators which transform the initial into the final state | (optimal) solution object, assembled from basic objects, which fulfills the requirements | (optimum) assignment plan, which maps the objects onto one another |
| Solution space | with $n$ = average length of the operator sequences and $m$ = average number of applicable operators in each intermediate state, there are about $m^n$ different plans | with $n$ = average number of object attributes required, each with $m$ values, there are $m^n$ different configurations | with two equal sets of $n$ objects there are $n!$ different assignment plans |
| Examples | planning experiments, production scheduling, moving building blocks, therapy planning | elevator configuration, computer configuration | timetable generation, utilization of machines |

**Figure 21.1** Comparative description of the problem types planning, configuration and assignment

empirical knowledge enables us to make simplifications, i.e., to do without the representation of states, the selection of operators or objects etc.

- *No representation of states:* The distinction between operators and objects with states which are changed by the operators is typical for planning. An operator is applicable if its preconditions are fulfilled and its action changes the state. Nevertheless, many planning problems can be solved without explicit representation of the states by selecting the operators not according to whether their preconditions are fulfilled by the current state, but purely heuristically. Examples: planning experiments (Friedland's MOLGEN), pro-

duction planning (ExAP).[3] Counterexample: planning in the Blocks World (NOAH).

- *No external change of states:* If the states are represented, are they affected only by the operators or is it necessary to take into account the possibility that external influences may change the states? In the latter case various alternatives must be considered in the plan. Example with no external change of states: planning in the Blocks World (NOAH). Counterexample: cancer therapy planning (ONCOCIN/OPAL[4]).

- *Given construction elements (no object/operator[5] selection):* Are the basic elements required for the solution already given or can they easily be derived from the formulation of the problem, so that it is not necessary to determine the objects or operators themselves, but only their properties? Examples: elevator configuration (SALT/VT), production planning (ExAP), timetable generation (REST). Counterexamples: planning experiments (Friedland's MOLGEN), planning in the Blocks World (NOAH).

- *No object/operator specification:* Are the construction elements more or less constants, with no properties to be specified, which only have to be selected and applied? This is characteristic of assignment problems. Examples: production planning (ExAP), timetable generation (REST). Counterexamples: configuration of elevators (SALT/VT) or computers (R1-XCON).

- *No assignment of operators to objects:* Is the arrangement of the construction elements of no importance or self-evident, so that it does not need to be considered in solving the problem? This is hardly the case for assignment or planning problems, but rather for some configuration problems. Examples: configuration of elevators (SALT/VT). Counterexamples: timetable generation (REST), production planning (ExAP), configuration of computers (R1-XCON), where the arrangement of the components must be taking into account for the wiring.

- *Division into phases:* Can the problem domain be divided into phases which can be carried out successively, independently of the concrete task at hand? Example: computer configuration (R1-XCON). Counterexamples: production planning (ExAP), planning in the Blocks World (NOAH).

---

[3] In production planning we assume that, on a suitable level of abstraction, it is possible to set up a clear correspondence between the required properties of the workpiece and the manufacturing actions, and that the main task is to find a correspondence between the manufacturing actions to the resources (machines, chucks, tools), taking into account restrictions in the order of the actions.

[4] ONCOCIN contains a set of pre-defined plans for the treatment of cancer patients, of which one is selected. On the basis of the selected plan ONCOCIN gives the doctor advice in the event of complications. OPAL facilitates the input of the treatment plans by providing a graphical programming language with procedures, branches, loops and domain-specific actions.

[5] "Object/operator" means the same as "construction element", since it refers to an operator in configuration and an object in assignment.

- *Hierarchies:* Is it possible and meaningful to first perform the construction with abstract objects and operators and refine these later along an object/operator hierarchy? And are reliable heuristics available for the refinement decisions? Examples: planning experiments (Friedland's MOLGEN), planning in the Blocks World (NOAH). Counterexample: timetable generation (REST).
- *Limitation to directional relationships:* For relationships which cannot be put into a hierarchy: Can the relationships between the objects/operators be represented as a directional graph, or is a non-directional graph required? To put it another way: Can the knowledge about the relationships be represented meaningfully by rules, or must constraints be used? Examples for directional relationships: configuration of elevators and computers (SALT/VT, R1-XCON). Counterexamples: production planning (ExAP).
- *No revision:* Is a problem-solving method possible in which the intermediate partial solution do not need to be revised, or can conflicts arise which require decisions to be withdrawn? Examples where no revision is required: configuration of computers (R1-XCON), planning experiments (Friedland's MOLGEN). Counterexamples: timetable generation (REST), configuration of elevators (SALT/VT).

## 21.4 Problem-Solving Methods

As in the case of classification we first consider different types of knowledge. In planning, one of the main criteria is whether or not we need an independent representation of the states which control the selection of the operators and are changed by the operators. If the states are represented, the simulation of a plan in which the application of the operator sequence to the initial state leads to the final state is always possible. Planning controlled by the states is certainly more basic than planning without states, which is why it will be called "model based" in analogy to the knowledge types in classification, while planning without explicit representation of the states is more "heuristic". Heuristic construction is also suitable for configuration and assignment, since in these cases the objects correspond to the operators of planning, and an independent representation of states is not necessary. Both types of knowledge can be supported by case-based construction, in which the objects/operators are taken over as far as possible from a similar solution from a case data base, and the necessary modifications are carried out with model-based or heuristic methods. We thus have the following coarse structure:

1. model-based construction (planning on the basis of states),
2. heuristic construction (heuristic selection of objects/operators),
3. case-based construction (selection of similar cases and heuristic or model-based modification of the old solution).

On account of the large search space of construction problems, search strategies [Nilsson 82, Chap. 2 and 3], [Barr 81, Part II] play an important role. In the simplest case the search space is traversed by a depth-first or breadth-first search. However, this usually fails on account of the exponential growth of the effort required. The search can often be considerably simplified by including heuristic knowledge (e.g., A*-algorithm, hill-climbing). According to our basic theory, however, it may be assumed that human experts do very little searching and use very knowledge-intensive search strategies. Since many search paths still have to be followed in parallel, procedures such as the A*-algorithm are less suitable for this purpose than local heuristics, such as the hill-climbing technique, which always look for the best local alternative. Possibilities for revision must be provided, since the possibility of running into a dead end cannot be excluded. Depending upon the nature of the knowledge, various forms of revision can be distinguished:

- Uninformed revision: cancelling the last, not yet tested alternative in the search tree, as in depth-first search.
- Dependency-controlled revision: cancelling an alternative in the search tree which led to the dead end, as in belief revision (TMS) techniques.
- Knowledge-based revision: cancelling an alternative in the search tree which is selected on the grounds of external knowledge.

Further important basic techniques for problem solving are abstraction and modularization, since they reduce the search space. In modularization a large search space is broken up into several small spaces (modules), which can be searched independently of one another except for specified interfaces. With the abstraction method the search is performed at the level of aggregated, non-elementary construction elements and later refined. In both cases it must of course be ensured that putting together the partial solutions does in fact give a solution to the whole problem, and this often causes new problems.

Finally, a further basic method is to convert the problem into a simpler one by deferring difficult decisions until they are relatively simple to resolve. The conversion may consist of a structural change or simply the restriction of the value range of problem variables as in the propagation of constraints. A typical example is Stefik's MOLGEN, in which difficult decisions remain open during the planning and are recorded as constraints, without having to interrupt the planning process. By this means the original planning problem is converted into a clear constraint problem and solved by simple constraint propagation.

From these basic ideas the following simple problem-solving methods can be derived, which, if sufficient knowledge is available, render the search space so manageable that only a few alternatives must be tried:

1. Skeletal construction (abstraction + modularization + local selection heuristics): The construction knowledge is structured hierarchically in a non-recursive AND/OR graph, the expansion of which is controlled by heuristic rules.

2. Propose and revise (hill climbing + knowledge-based revision): The best local alternative according to heuristic rules or procedures is proposed. If a dead end is reached, a suitable revision is proposed with the aid of heuristic rules or procedures.

3. Least commitment (problem conversion + constraint propagation): The problem is first re-structured, and the simpler problem can then often be solved with constraint propagation or basic strategies.

These basic methods appear in many variations and under different names in computer science. According to the considerations of Sect. 12.2 we limit our description to simple knowledge structures. We have therefore taken the names from papers whose authors have used the methods successfully in expert system applications with such simple knowledge structures: skeletal construction as a generalization of the concept of skeletal planning from Friedland's MOLGEN [Friedland 79, 85], propose and revise from SALT [Marcus 88] and least commitment from Stefik's MOLGEN [Stefik 81].

Of the categories introduced above (problem type: configuration, assignment, planning; knowledge type: model-based, heuristic, case-based; problem-solving methods: skeletal construction, propose and revise, least commitment) not all combinations are meaningful for the classification of construction problems. As mentioned above, model-based construction is relevant only for planning and not for configuration or assignment. Case-based construction has only a limited applicability to assignment on account of the general lack of possibilities for abstraction and the particularly large search space. For heuristic construction, on the other hand, there are no restrictions. Of the problem-solving methods, skeletal construction is rarely helpful for generally non-hierarchical assignment problems, for which the least-commitment strategy is also usually unsuitable on account of the lack of locally evaluable restrictions.

In the following description of problem-solving methods we concentrate on heuristic construction techniques. Model-based construction (planning) is too complex, as shown by the fact that there have been hardly any successful applications. We therefore give only a short review of this technique. Case-based construction also needs only a short description, because the case comparison corresponds to that of classification, and no specific case-comparing construction strategies are needed to modify the old solutions. The issue of learning from errors in the old solutions is not treated here.

If planning problems are solved heuristically there is no great difference between planning and configuration, since the complex additional conditions of planning are hidden in heuristic knowledge. On the other hand there is a considerable difference between configuration and assignment: the latter deals with many similar objects, while the individuality of the construction elements is often of prime importance in configuration. It is therefore meaningful to distinguish between two different versions of the propose-and-revise strategy for configuration and assignment. The assignment version is called "propose and exchange",

since the improvement of the solution of an assignment problem usually involves exchanging objects.

The strategies are first considered in their pure forms, e.g., propose and revise without hierarchies and skeletal construction without constraints. An approach to integration is considered in the last chapter of Part V.

The resulting structure of the problem-solving methods which are described in more detail in the following chapters is summarized in Fig. 21.2, and Fig. 21.3 shows the suitability of the different problem-solving methods for the problem types planning, configuration and assignment.

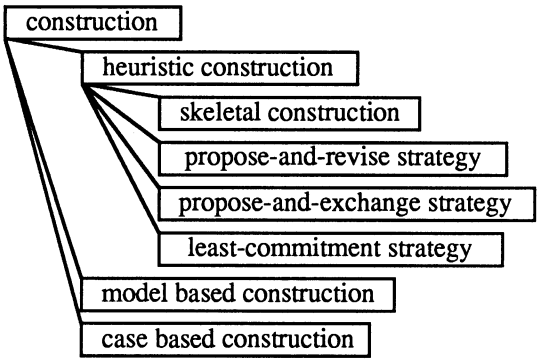



**Figure 21.2** Problem-solving methods for construction

| | Planning | Configuration | Assignment |
|---|---|---|---|
| Skeletal construction | × | × | |
| Propose and revise | × | × | |
| Propose and exchange | | | × |
| Least commitment | × | × | (×) |
| Model-based construction | × | | |
| Case-based construction | × | × | (×) |

**Figure 21.3** Suitability of the problem-solving methods for the different problem types. The first four methods operate with heuristic knowledge.

# 22. Skeletal Construction

Skeletal construction is suitable for simple planning and configuration problems in which the search space may be represented in a hierarchical, non-recursive AND/OR graph, and selection knowledge for the hierarchical expansion of the graph is available. The "AND" nodes in the graph are called "skeletal plans" and the "OR" nodes "construction steps" (Fig. 22.1).

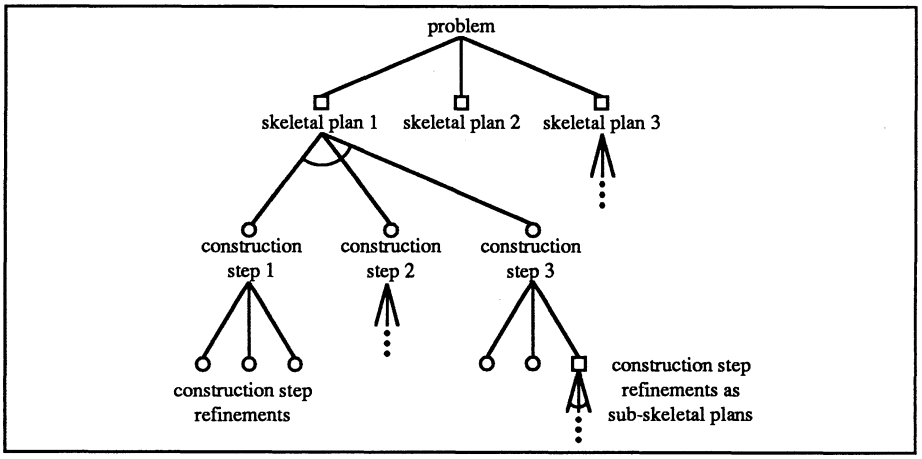



**Figure 22.1** Abstract structure of the knowledge in skeletal construction: a skeletal plan is selected and expanded into its construction steps, which are then refined, whereby sub-skeletal plans may be expanded, etc.

The decomposition of a construction problem into a set of partial problems (construction steps) by means of a skeletal plan and the representation of the search space of the partial problems in a hierarchy are not always predefined in the domain, but often based on the experience of experts, who are accustomed to reducing complex problems to a simple structure. The abstract construction steps of a skeletal plan then constitute a generalized representation of the solutions of many similar problems. When a new problem is encountered, a suitable skeletal plan is first selected, and all the abstract construction steps contained in it are then concretized to primitive construction steps with rules. The flexibility can be increased considerably by allowing subsidiary skeletal plans to be represented as construction steps. Parameters, i.e., properties of the operators or objects, do not play a dominant role in skeletal construction, since they are taken into account

implicitly in the hierarchical refinement. If required, they can be derived directly from the primitive construction steps. For this reason they are not included in the knowledge representation (a more complex representation of parameters would constitute a different problem-solving method). Simple examples of skeletal plans with construction steps for planning and configuration are given in Figs. 22.2 and 22.3.

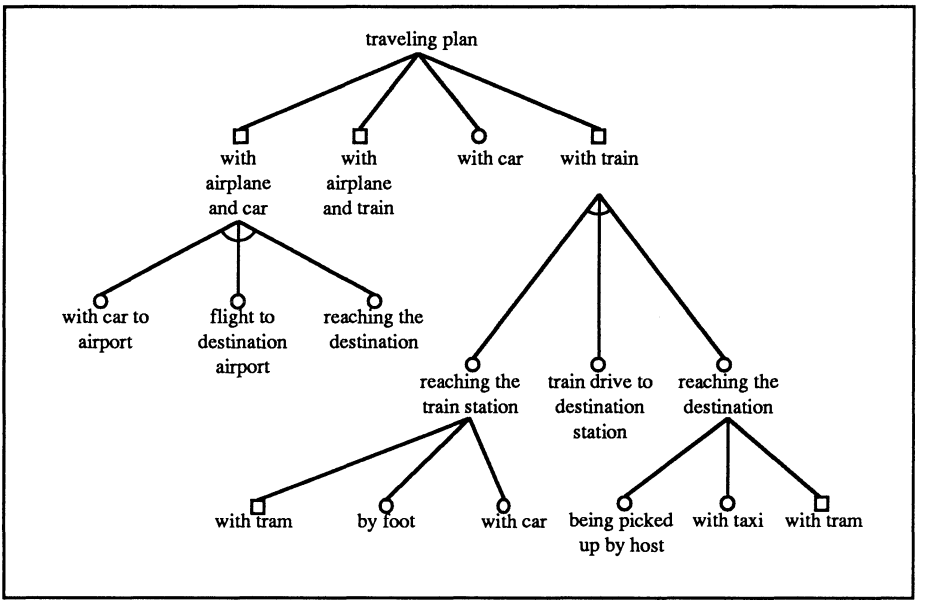



**Figure 22.2** Skeletal plans for travel planning

Skeletal construction is similar to multiple heuristic classification, since both the skeletal plan and the primitive construction steps are selected from a given set of alternatives. The uncertainties of heuristic classification have their counterpart in the priorities for choosing between various possible refinement steps. In addition, each construction step can be assigned a cost value, and the total costs of the construction can be found by summing over all steps. If the selection of a construction step is under-determined, and a random guess leads to a contradiction with respect to the selection of another construction step, it must be possible to revise the choice.

## 22.1 Knowledge Representation

The skeletal plans and the refinement hierarchies for the construction steps constitute the core of the knowledge representation. Problem features which control

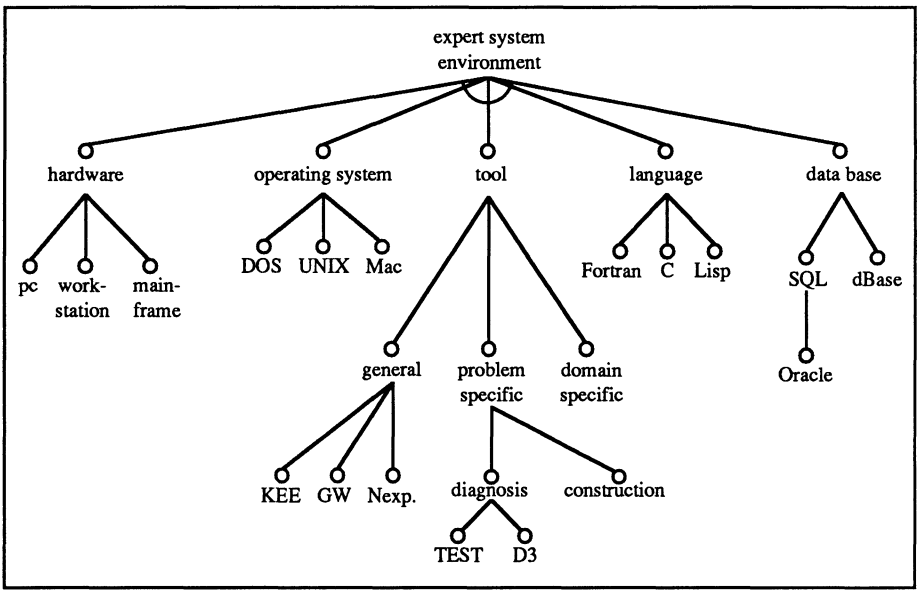



**Figure 22.3** Skeletal plan for the configuration of an expert system programming environment

the selection of the skeletal plans and construction steps by means of rules must also be represented. For these object types the following attributes are required:[1]

1. Skeletal plans
   - name
   - elements (abstract construction steps making up the skeletal plan)
   - derivation (s) (rules for the applicability of the skeletal plan)
   - value (d) (instantiated, not instantiated; derived by the rules).
2. Construction steps
   - name
   - refinement (more concrete construction steps or skeletal plans)
   - costs (global costs of the construction step)
   - predecessor (skeletal plan or general construction step)
   - derivation (s) (rules for deriving the construction step)
   - meaning (s) (rules for deriving the consequences)
   - assessment (d) (local assessment; derived by rules)
   - value (d) (instantiated, not instantiated).

[1]Semi-static attributes, which can be derived from the normal (static) attributes by the knowledge acquisition component, are marked (s), and dynamic attributes (d).

3. Problem features (user requirements)
   - name
   - value range
   - meaning (s) (rules for the selection of skeletal plans or construction steps)
   - value (d) (input from user).
4. Rules
   - context (construction step whose refinement is determined by the rule)
   - precondition (predicates concerning problem features and construction steps)
   - action (construction step or skeletal plan)
   - assessment (of the action)
   - value (triggered, not triggered)

On account of the similarity to heuristic classification, the knowledge representation can be supplemented by many of the structures mentioned in Chaps. 15 and 16, e.g., tests for dialog control and for data abstraction.

## 22.2 Knowledge Manipulation

We start from the following basic algorithm:
Input: Problem features
Output: set or sequence of construction steps constituting the solution.

1. Input of the problem features by the user.
2. Select a skeletal plan with the aid of the rules given in the "meaning" of the features.
3. Put the "elements" of the skeletal plan on a global data structure CURRENT-CONSTRUCTION.
4. *While* there is a construction step in CURRENT-CONSTRUCTION whose refinement is not empty, *repeat*
   4.1 Select a construction step from CURRENT-CONSTRUCTION whose refinement is not empty. With the aid of the "meaning" rules of the selected construction step $K$ select a successor $K'$. If other construction steps are needed for this purpose, postpone the refinement until they have been processed. Replace $K$ by $K'$ in CURRENT-CONSTRUCTION.

4.2 If $K'$ is a skeletal plan, replace $K'$ by the elements of $K'$ in CURRENT-CONSTRUCTION.

5. Output of the construction steps of CURRENT-CONSTRUCTION.

First the user specifies the features of his problem. The rules listed in the attribute "meaning" of the problem features choose a skeletal plan which is recorded in a global variable " CURRENT-CONSTRUCTION". Then the construction steps of the skeletal plan are instantiated and refined step by step to primitive construction steps. During refinement further problem features may be requested, or the process may be interrupted until other construction steps have been sufficiently refined. If several successors are allowed, the one with the best "assessment" is chosen. The same is true if there is not enough information so that a guess must be taken. In both cases the chosen construction step and the alternatives are recorded in a global variable "REVISION-LIST", which serves for handling any contradictions which may occur, i.e., if a later construction step cannot be refined. In this case the context and the preconditions of the rules of this step are investigated to see if they refer directly or indirectly to a construction step on the "REVISION-LIST", and if so alternatives are tried. The system terminates when all construction steps of the original skeletal plan and of any subsidiary plans have been resolved into primitive construction steps (with no "refinement"). The result is the selected skeletal plan with its refinements and costs. If desired, additional solutions can be generated by instantiating the alternatives of the "REVISION-LIST".

## 22.3 Knowledge Acquisition

The basic graphical elements for the knowledge acquisition are very similar to those of heuristic classification (Sect. 15.5):

1. For constructing a concept hierarchy with skeletal plans, construction steps and problem features, a graphical hierarchy with the following components (separable for the sake of clarity) can be provided: Each abstract construction step is the root of its own hierarchy of refinements. Abstract construction steps which belong together have their skeletal plan as predecessor. Since the selection of the skeletal plans and the construction steps is controlled by problem features, the latter should also be represented in the graphics, with links to the other objects.

2. For each object type specific forms are provided for entering the local knowledge.

3. The relational knowledge (i.e., the rules) can be specified either directly by mouse-clicking a link on the graph, whereupon a correspondingly prepared rule form appears, or by means of tables. Of the various types of table, that with a column for each rule (see Fig. 12.3) is preferable, if there is usually only one – possibly complex – rule for the selection of a refinement.

## 22.4 State of the Art

Searching in AND/OR graphs by the hill-climbing method is an established technique [Nilsson 82]. Its significance as a strong problem-solving method for planning problems was demonstrated above all by Friedland in his thesis on his MOLGEN [Friedland 79] and in the successor system SPEX [Friedland 85], both of which plan experiments in molecular genetics. According to Friedland's studies, experienced experts in molecular genetics adapt suitable skeletal plans to the experiment instead of planning experiments completely from basic principles. Friedland's MOLGEN works essentially according to the knowledge representation and problem-solving structure described above, whereas SPEX contains two extensions: Preconditions and actions are given for the individual construction steps, so that the finished plan can be simulated, and the order in which the planning steps are executed (e.g., expansion of a skeletal plan, refinement of a construction step) is controlled by an agenda, in order to be able to try out different planning strategies (so-called meta-planning).

A large, very attractive domain for skeletal construction is system configuration when drawing up offers for customers, e.g., specifying computer systems with hardware and software for special applications. The various main alternatives (e.g., the types of application) correspond to the skeletal plans, and the elements of these alternatives (e.g., type of computer, operating system, data base, etc.) correspond to the construction steps, which must be concretized (cf. Fig. 22.3).

An example of a problem-specific expert system tool for skeletal construction is PLAKON, which also includes other construction strategies, however (see Chap. 28).

## 22.5 Variants of Skeletal Construction: Generate and Test

An essential property of skeletal construction is that all skeletal plans and construction steps are named as separate objects. This allows the knowledge to be represented clearly in an AND/OR hierarchy with uncomplicated rules. In the general generate-and-test strategy the objects are not named explicitly, but an algorithm is specified for determining the successor of the current construction step. In principle this algorithm can generate all possible solutions (i.e., the whole search space) and would therefore be too inefficient without test functions. The tests correspond to the selection rules in skeletal construction and determine which of the successors generated are relevant for the solution and are therefore added to the current construction step.

An example for the application of the generate-and-test strategy is DENDRAL [Lindsay 80], which determines the structure formula of chemical compounds from their sum formula and the interpretation of mass spectrograms. The generating algorithm can produce all structure formulas consistent with the sum formula, and is controlled by the mass spectrograms and possible additional

knowledge about the properties of the chemical substance to such an extent that unlikely partial structures are recognized immediately and excluded.

## 22.6 Heuristic Classification and Skeletal Construction

The gradual transition from the selection to the assembly of solutions becomes particularly clear when heuristic classification is compared with skeletal construction. In the former case the knowledge consists of an OR-hierarchy of coarse and fine diagnoses, and in the latter case of an AND/OR hierarchy of skeletal plans and refinements. Since multiple diagnoses are also possible in classification due to explicit decisions between competing, independent alternatives, this additional mechanism only needs to be further extended in order to include skeletal construction. This may be illustrated by the example of the application of the diagnostics shell MED2 (see Sect. 15.5) to the therapy planning for diabetes patients: A diabetes therapy consists of several partial therapies for diet, oral antidiabetics, insulin, resorption inhibitors, appetite reducers, physical exercise, monitoring and patient instruction. The refinement of the various partial plans is essentially a classification task. A pure diagnostics shell like MED2 supplies the solution as a set of individual suggestions, where there is no clear distinction between partial plans which belong together (e.g., diet-$X$ and insulin-$Y$) and competing alternatives within a plan (e.g., insulin-$Y$ or insulin-$Z$). Without having to change the shell, the problem can be solved by reciprocal exclusion of competing diagnoses (using certain knowledge or the attribute "competitors", see Sect. 15.1) and a suitable representation (in print) of the results. This approach was also used in applications of MED2 to the heuristic construction of spray-cast tools [Nedeß 88, Friedewald 90] and to the selection of tool machines. On the conceptional level this procedure can be supported by a new object type "skeletal plan", which is added to the diagnostics shell and contains a set of construction steps (i.e., "diagnoses"), in order to represent the grouping of elements correctly.

# 23. Propose and Revise

The propose-and-revise strategy is suitable for deriving values for a set of parameters. It makes possible an efficient, sequential determination of the parameter values, even when the search space is locally or globally under- or over-determined or when cyclic relationships exist. Its flexibility results from the fact that the parameter values first proposed can be corrected at any time with "revision knowledge" if this turns out to be necessary. It is suitable for cyclic parameter relationships because loops can be broken by taking a cyclic parameter as a starting parameter and estimating its value. If a contradiction occurs in the subsequent calculations on account of the cyclic relations, the parameter value is raised or lowered (Fig. 23.1).

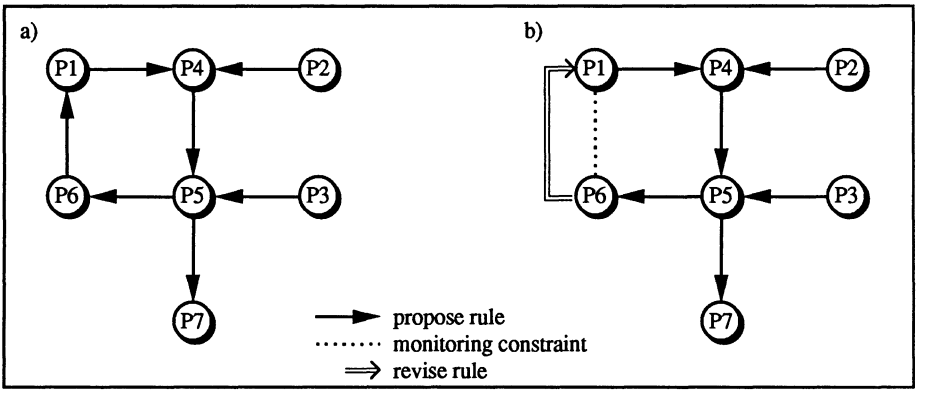



**Figure 23.1** Cyclic (a) and non-cyclic (b) calculation. To solve (a) optimizing methods for equations with variables are needed, but for (b) a simple rule interpreter with a revision technique is sufficient. However, the method is very inefficient if no good heuristics are available for choosing the initial value of P1 or for choosing a possible revision on the grounds of the value of P6.

The first step in the procedure is to determine proposed values for the parameters with the aid of rules. As each new value is determined, its consistency with the other parameters is checked with constraints. If a contradiction is found, a value must be corrected, and this is done by a special revision rule assigned to the constraint. Then the relationship network is updated taking into account the new value. If further contradictions occur, they are dealt with in the same way. If two constraints change a value in opposite directions, they should both be taken

into account in determining the new value. If no set of parameter values is found which fulfills all constraints, then the locally available alternative which violates the constraints of lowest priority is chosen.

The quality of the proposal and revision rules determines how close the solution comes to the optimum and how many corrections are needed. The order in which the parameters are determined may also play a role, since the process is not commutative. The proposals and the revisions contain the knowledge of the application expert. The structural knowledge about the boundary conditions which the solution must fulfill is represented in the constraints.

## 23.1 Knowledge Representation

The procedure consists of two components: an assignment system which derives values for the parameters and a control system which discovers and corrects any contradictions among the values. A forward-chained rule interpreter is suitable as the assignment system. The revision of values is simplified if the rules are provided with pointers to all their objects, as will be assumed in the following discussion. Like the rules, the constraints of the control system have a precondition, which describes a contradiction; the action is a pointer to a revision rule which removes the contradiction by re-calculating a value. A further attribute of the constraints is a list of interacting constraints, the validity of which should also be checked when calculating a new value. The parameters are divided into given parameter, which represent the user requirements, and derived parameters. We thus have the five object types user requirements, parameters, proposal rules, constraints and revision rules with the following attributes:[1]

1. User requirement (question)
    - name
    - value range type
    - value range
    - meaning (s) (rules or constraints which derive consequences from the user requirement)
    - value (d) (answer given by the user to the question).
2. Parameter
    - name
    - value range type
    - value range
    - meaning (s) (rules or constraints which derive consequences from the parameter)

[1]Semi-static attributes, which can be derived from the normal (static) attributes by the knowledge acquisition component, are marked (s), and dynamic attributes (d).

- derivation (s) (rules for deriving the parameter)
- value (d) (derived)
- reason (d) (rule with which the value was derived).

3. Proposal rule
   - precondition
   - action (procedure for calculating parameter values)
   - target (parameters affected by the action)
   - value (triggered, not triggered)
4. Constraint
   - condition (logical restriction concerning user requirements and parameters which must be fulfilled)
   - reason (comment)
   - revision (pointer to revision rule)
   - interactions (pointers to interacting constraints)
   - priority (natural number, where 1 represents the lowest priority)
   - value (fulfilled, violated, inactive).
5. Revision rule
   - context (s) (the constraint which activates the revision rule)
   - precondition
   - action (procedure for calculating values)
   - target (parameters affected by the action)
   - value (triggered, not triggered).

## 23.2 Knowledge Manipulation

First the user enters his requirements. The proposal rules of these objects are activated by the attribute "meaning" and determine derived parameter values. Through their "meaning" the latter activate mainly constraints to detect possible contradictions. Then the revision rules given under the attribute "target" of the constraints are tried out one after the other to see if their actions resolve the direct contradiction. If "interactions" exist, their constraints are also taken into account. If a new value for a parameter has been found, it is passed on (recursively) to all the relationships under "meaning", in order to effect the revision of all consequences. In addition the new "reason" for the value is entered. The result is a complete set of parameter values, which may come very close to the optimal

solution on account of the heuristics. However, no objective criterion can be given for this. The basic procedure may be summarized as follows:
Input: partial initialization of parameters as user requirements
Output: complete, consistent set of parameter values.

1. Evaluate the proposal rules on the basis of the user requirements. This can be done with a forward-chaining rule interpreter or by an agenda control, which takes note of the parameters to be determined and processes them in a favorable order.
   *If* there are no more active proposal rules or values have been determined for all parameters and there are no unresolved contradictions,
   *then* output parameter values.

2. If a control rule discovers a contradiction, interrupt step 1 and:

   2.1 With the revision rule of the constraint calculate a new parameter value (which resolves the contradiction).

   2.2 Calculate all the consequences of the new parameter value, and in particular revise all affected parameter values which have already been calculated.

   2.3 *If* after completing the revision with one constraint other contradictions are discovered,
   *then*

   *if* a certain number of revisions has not been exceeded,
   *then* repeat step 2 for the new contradiction, retaining the revisions tried so far,
   *else* repeat step 2 from scratch, annulling all previous revision attempts, but ignore all constraints with priorities one more than the old priority level (initialized to 0);

   *else* continue with step 1 at the interruption.

## 23.3 Knowledge Acquisition

Graphs representing the relationship network (see Fig. 23.1b) serve as the basis for knowledge acquisition. The nodes of the graphs represent the parameters, and the links proposal rules, revision rules or constraints. For the sake of clarity the network may be divided into smaller sub-graphs. The knowledge acquisition takes place in two steps:

1. Drawing the parameter nodes and the links with specification of the direction and type.

2. Filling in the type-specific forms for the parameters and links.

## 23.4 State of the Art

The propose-and-revise strategy is a combination of three well-established, basic problem-solving methods: (1) forward-chaining of rule systems, (2) monitoring with constraints and (3) dependency-controlled revision with a specialized form of a JTMS (justification-based truth maintenance system), in which, however, the corrections for resolving contradictions are carried out with domain-specific knowledge. Unlike optimizing algorithms for systems of linear equations, the propose-and-revise strategy cannot guarantee to supply the optimum values for the parameters, but it has the advantage of being applicable even if the optimizing criteria are complex or difficult to specify, since it does not require a complete mathematical understanding of the domain. Like skeletal construction it is a relatively strong problem-solving method, since the procedure is efficient and knowledge-intensive, the object-attribute structure is fixed and the use of variables is restricted to container variables, so that a good graphical support of the knowledge acquisition is possible. In the form described above, however, the strategy is suitable only for non-hierarchical parameter configuration. An extension for assignment problems is discussed in the next chapter.

In spite of its simplicity and usefulness, the propose-and-revise strategy was implemented in explicit form only recently in the problem-specific expert system tool SALT [Marcus 88a,89] on the basis of the rule language OPS5 and tested with the expert system VT [Marcus 88b] for the configuration of elevators. SALT functions by processing a directional network of (mainly numerical) parameters and assigning a value to each by calculation or by access to a data base. If a contradiction is discovered by a constraint, one or more alternative values are calculated. The alternatives are weighted with numbers from 1 (causes no problems) to 12 (impairs the system performance), the best is selected, and the new value is passed on. SALT can also compare the global effects of several alternatives by listing the resulting values of all parameters.

## 23.5 Variant for Therapy Planning by Parameter Adjustment

In many cases therapy planning consists of a parameter adjustment in order to favorably influence recognized problem variables and to minimize the usually unavoidable side effects. The knowledge consists of rules defining how the parameters affect the problem and side-effect variables. The relationships can be specified in a linearized table form (parameter influences problem variable by the factor $x$) or as complex functions, possibly involving many variables.

The starting point is the occurrence of unfavorable values for some of the problem variables. Parameters which affect the problem variables favorably are selected and activated. The side effects are monitored, and if the value of a side-effect variable shows a large increase a parameter can be selected which counteracts the increase. If this parameter causes other side effects, they must also

be corrected. If necessary, side effects up to a certain level (and with as low a priority as possible) must be accepted.

This process corresponds to the propose-and-revise strategy if we regard the primary parameter selection as step 1 in the algorithm of Sect. 23.2, the monitoring of the side effects as step 2 and the correction of the side effects as the steps 2.1 to 2.3. The knowledge representation of Sect. 23.1 can be simplified, since the condition of a constraint is simply the comparison of a side-effect variable with a threshold value, and the proposal and revision rules all point directly from parameters to problem or side-effect variables with no intermediate elements, so that no TMS is required to maintain consistency during revision. An example for the above method of parameter adjustment which has been successful in practice is PAREX-CO [Hartinger 90] for the configuration of the parameters of a production planning and control system.

## 23.6 Heuristic Classification and Propose and Revise

The data pre-processing of heuristic classification (Chap. 15) with the additional mechanisms for plausibility checking (Sect. 16.3) and belief revision (Sect. 16.5) provide a good basis for the realization of a simple form of the propose-and-revise strategy.[2]

The most important extension required is to connect the monitoring rules to the plausibility check with corrective measures whose evaluation is taken over by the revision algorithm. This gives a new type of rule with the basic form: *If <monitoring condition violated> then {raise, lower} a parameter Y by the amount x, but not more than n times, or set the value of the parameter to z.* The action part of the rule should also be able to correct several parameter values at the same time and to supply several correction alternatives.

The parameters can be represented as data abstractions in the heuristic diagnostic shell. In addition, their status with regard to the configuration must be specified, e.g., whether they are part of the final result. For this purpose new attributes are required.

[2]Since the integration of the problem-solving strategies is discussed later (see Part VII) we emphasize mainly the cross-connections here. We take MED2 (see Sect. 15.5) as a model of the classification shell.

# 24. Propose and Exchange

The propose-and-exchange strategy is suitable for the heuristic solution of complex assignment problems for which exact optimizing algorithms are not applicable. The basic idea is that of the propose-and-revise strategy (Chap. 23). The task of assignment is not to determine object properties as in configuration (e.g., the thickness of a cable in elevator configuration) but to assign given sets of objects to one another according to certain boundary conditions (see Fig. 24.1).

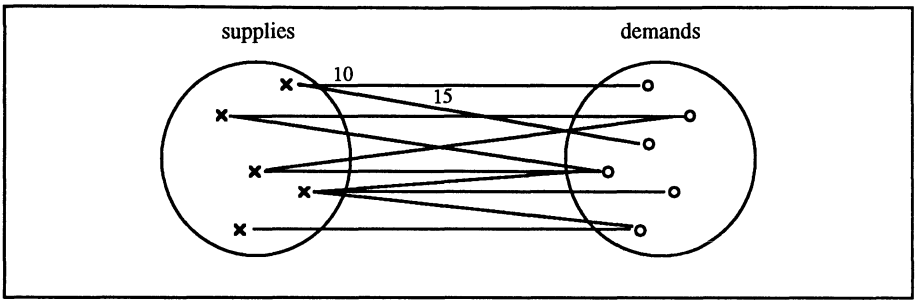



**Figure 24.1** Basic structure of an assignment problem. The task is to find a mapping, often bijective, between at least two sets of objects ("supply" and "demand") which fulfills certain boundary conditions, e.g., the availability of resources or the "affinity" between the objects, which is indicated by the lines and may be weighted numerically or qualitatively. In the latter case we have an optimizing problem. Examples: jobs/jobseekers, homes/tenants, university places/study wishes, production capacity/orders, school periods/lessons, storage room/products, airport park positions/aircraft, chessboard squares/eight queens, pieces of cake/tea-party guests, etc. If necessary the sizes of the sets can be adjusted by means of "dummy objects".

The distinguishing feature of assignment problems is their extraordinarily large search space, which usually makes systematic testing impossible, since for $n$ elements $n!$ assignments would have to be tried, even for the simple structure shown in Fig. 24.1. Although there are some efficient, special optimizing algorithms for simply structured assignment problems, they can usually consider only certain types of boundary conditions. For example, static weighting of the affinities of pairs of objects in Fig. 24.1 would not cause problems, but dynamic weighting, which may change according to the assignments already made, would. Examples of assignment problems with dynamic weighting are timetable generation, if gaps in the students' timetables (i.e., free periods) are to be avoided, or

machine occupancy planning, if the time needed to reconfigure a machine depends upon the order in which the tasks are carried out.

The propose-and-revise strategy introduced in the last chapter cannot guarantee an optimal solution, but it can easily be adapted to all kinds of boundary conditions and can also use well-tried heuristics for proposals or revisions. On account of the differences in the knowledge representation of assignment and configuration problems, the propose-and-exchange strategy for assignment problems is described here as an independent problem-solving method, although the basic structure, with proposal, monitoring and revision knowledge, is the same.

The special features of the knowledge representation for assignment problems arise from the fact that the usual division into (a) object types given by the problem-solving method, (b) objects defined by experts as instances of the object types and (c) particular values of the objects input by the user or derived during the problem-solving process is less clearly defined. In assignment problems the object types are application-specific (e.g., for the timetable problem object types such as teacher, classrooms, classes, lessons and times), so they must be determined by the expert. The object instances, i.e., the actual teachers, lessons, etc., must be specified by the user, and the "values", i.e., object assignments such as lessons to times, are derived by the problem-solving mechanism. The problem-solving method can only provide aids for specifying the object types and instances, so that more demands are placed on the expert and the user than in most other problem-solving methods. It may be that this phenomenon is typical of complex construction strategies (see also Chaps. 25 and 28).

Since the abstract level of the problem-solving method is difficult to understand, it will be illustrated with the simple assignment problem depicted in Fig. 24.1. A more ambitious application, the generation of timetables, is described in Sect. 24.5 as an example.

## 24.1 Knowledge Representation

An assignment problem has at least two object types, each with an attribute for entering the assigned object, e.g., object type "supply" with attribute "demand" and object type "demand" with attribute "supply". An attribute entry in one object triggers an "inverse" entry in the assigned object. Depending upon the application, other object types and attributes can of course be used to represent resources, preferences or object properties which are directly or indirectly relevant for the assignment. Since these cannot be pre-defined, the expert must specify the names of the object types and their attribute structure. However, help can be provided by offering the user a selection of attribute types, so that the attribute definition needs only to be chosen from a set of options. An attribute is characterized by the type of value range and the type of meaning, which may have to be specified by giving the type of object or the name of the attribute. If the meaning type is "derived", the proposals and constraints for the derivation must also be given as attached procedures of the attribute.

Some important types of value range are:

V1: "string",

V2: "number",

V3: "element of a set of strings",

V4: "an element of a set of objects of a certain type",

V5: "several elements of a set of objects of a certain type",

V6: "several elements of a set of objects of a certain type, with preferences."

Some important meaning types of an attribute are:

M1: "value is entered by user",

M2: "value is determined from another, inverse attribute entered by the user",

M3: "value is derived, but does not initiate any subsequent actions",

M4: "value is derived and causes an inverse entry in the assigned object, which then does not need to be derived directly",

M5: "value is derived and causes similar entries in other objects which are listed in certain attributes of the object."

The meaning type M5 is important for the management of resources, e.g., if a certain period is entered for a lesson, then the resources required for the lesson (teacher, class and classroom) must be allotted the same period.

Since the derivation procedures for all attributes of a given object type are similar, it is not necessary to give each individual its own set of rules or constraints as in the propose-and-revise strategy for configuration, but one can define general procedures for the attributes of the object type. These procedures are more complicated than normal rules, since they can refer to concrete objects only via reference variables.

The procedures serve to evaluate the assignment boundary conditions specified in the attributes. For the propose-and-exchange strategy three types of procedures are needed: proposals, constraints and corrections. They usually have the following structures:

- Proposal procedures: *Contents:* "Propose all objects of a certain type which fulfill certain predicates. If preferences exist, the proposals should be given in the corresponding order." *Attachment:* to at least one object-type attribute. *Call:* whenever a value is to be determined.

- Constraints: *Contents:* predicates concerning the values of the object-type attributes. *Attachment:* possibly to several object-type attributes. *Call:* after entry of a value in one of the attributes.

- Corrections: *Contents:* "Make a new suggestion for the entry in an object-type attribute which contributed to the violation of the constraint. If necessary try to make room for the new value by shifting the entries in other object-type attributes in one or more stages." *Attachment:* possibly to several constraints. *Call:* by the constraints.

Although the exact procedure code cannot be laid down at the level of the problem-solving method, since it depends upon the still unknown object types, the expert can be provided with a language which includes functions for determining the basic set, the predicates about object attributes and the relationships. However, this language does not lend itself to graphical input support.

When defining the proposal procedures, the expert must find an application-dependent optimum between two extremes: either simple generation of proposals with possibly complex corrections or complex generation in order to avoid corrections as far as possible. A typical "trick" is to consider weak preferences up to a certain level only in the proposals, and not to monitor them with constraints, in order to avoid endless corrections.

A feature of the propose-and-exchange strategy is that the objects can usually be assigned very well at first, but the last objects may be left with poorly suited partners. This can be compensated only with difficulty, and even then only partially, by corrections. For this reason the order in which the objects are processed is of great importance. The objects with the greatest restrictions and the most preferences should be assigned first, whereby the priorities may change dynamically. Therefore a central control is required, e.g., an agenda with all objects which still have no partner, where the objects may be sorted according to a priority function. Since re-sorting the agenda after every assignment would usually be too time-consuming, a compromise must again be found between the sorting effort and the quality of the result. One possibility is to pre-sort the agenda and then check during the problem-solving process with the aid of a new type of attached procedure whether the number of assignment possibilities for any object has become so small that it should be moved to the top of the agenda. In the timetable problem, for example, if there are only a few periods left for a class, a teacher or a classroom, the remaining lessons for this object should be assigned with high priority.

A complete knowledge representation for the simple example of Fig. 24.1 with object types supplies and demands could therefore have the following form (the types of the value ranges and meanings are given in brackets, see above):

- Supplies:
  - name (V1, M1)
  - incompatibilities (V5, M1) (list of incompatible demands)
  - preferences (V6, M1) (list of demands with preferences, e.g., {D1 5, D2 1, D3 -3}]
  - demand (V4, M4) (the object of type demand which was assigned to the supply object in the course of the problem-solving process)

- Demands:
  - name (V1, M1)
  - incompatibilities (V5, M2) (list of incompatible supplies, symmetrical to the supply incompatibilities)
  - preferences (V6, M1) (list of supplies with preferences, e.g., {S1 4, S2 1, S3 -2}, which are not necessarily symmetrical to the supply preferences, however)
  - supply (V4, M4) (the object of type supply which was assigned to the demand object in the course of the problem-solving process)
- Proposals:
  - Seek-supply(DEMAND): "Propose all objects X of type 'supply' which are not listed under the attribute 'incompatibilities' of DEMAND and for which X.demand has not been assigned. Sort the proposals in the order of the values given in DEMAND.preferences. Enter the best X in DEMAND.supply and DEMAND in X.demand."
  - Seek-demand(SUPPLY): "Propose all objects Y of type 'demand' which are not listed under the attribute 'incompatibilities' of SUPPLY and for which Y.supply has not been assigned. Sort the proposals in the order of the values given in SUPPLY.preferences. Enter the best Y in SUPPLY.demand and SUPPLY in Y.supply."
- Constraints [with weighting]:
  - assigned(SUPPLY)[3]
  - assigned(DEMAND)[3]
  - incompatible(SUPPLY,DEMAND)[2]
  - negative-preference(SUPPLY,DEMAND)[1] "The predicate is true if one of the two preferences has a negative value."
- Corrections:
  - Single-step-exchange(SUPPLY,DEMAND): "For the object X under SUPPLY.demand or DEMAND.supply, seek a position which has a high preference, is not assigned and is not incompatible. Put X there, and put DEMAND or SUPPLY, respectively, in the old position of X. If the best correction possibility has a lower preference than the constraint originally violated, then no correction is made."
  - Two-step-exchange(SUPPLY,DEMAND): "As single-step-exchanged, but additional combination with two exchanges are tried."

- Agenda control:
  - sort-agenda: "Sort all objects according to the length of the attribute 'incompatibilities' and if equal according to the length of 'preferences'."

Each proposal procedure is attached to one of the two object types, all constraints are attached to both object types, and the correction procedures to all constraints. The agenda procedure is global, so that the object types do not require an attached agenda procedure.

## 24.2 Knowledge Manipulation

The basic algorithm is largely the same as that of the propose-and-revise strategy described in Sect. 23.2, except that the forward-chained rule interpreter in step 1 has been replaced by an explicit agenda control (steps 1 to 2.2):
Input: specification of the supplies and demands with properties
Output: Assignment between supplies and demands, taking into account their properties.

1. Place the objects on an AGENDA according to an agenda sorting function.
2. *While* AGENDA not empty *repeat*:
   - 2.1 Select an object on AGENDA and determine a partner object with its proposal procedure.
   - 2.2 Register the assignment and execute direct consequences, in particular record inverse assignments and execute agenda manipulations.
   - 2.3 Check the constraints, and if any are violated: execute of the attached corrections as in the steps 2.1 to 2.3 of the propose-and-revise algorithm of Sect. 23.2.
3. Output the assignments for all objects.

The procedure needs to be changed only slightly even if several coupled assignment problems are to be solved, e.g., the assignment of lessons to rooms and to periods in the timetable problem, or the simultaneous search for jobs and homes by people. Only a few additional object types, attributes and/or more complex procedures are needed.

## 24.3 Knowledge Acquisition

As mentioned above, the knowledge acquisition cannot be supported graphically as easily as in the previous problem-solving methods, since the expert must formulate object types and attached procedures rather than concrete objects and rules

referring to them. Without special aids the definition of the procedures in particular requires a knowledge of a normal programming language, which imposes considerable restrictions on the use of a corresponding shell by specialist experts. An important requirement is therefore the definition of a special language with basic functions and predicates for the procedure definitions. In addition, attempts should be made to provide a graphical shell interface or at least a comfortable syntax-controlled editor. For this purpose, however, more experience in different domains is needed (see Sect. 24.4).

However, graphical aids can be of great assistance to the user who wishes to enter a concrete problem from his domain, since almost all the input data consist of relationships between concrete objects (e.g., the concrete incompatibilities and preferences in the example of Sect. 24.1). Tables are particularly suitable. In addition it is usually important for the user to be able to manipulate the results interactively, which can be integrated without difficulty into the propose-and-exchange strategy. For this purpose also, the results can be presented in the form of tables. However, the expert now has the additional task of configuring the interface for the user. In this he can be assisted with some primitive graphical objects, but it is not possible to pre-define a standard interface.

## 24.4 State of the Art

To our knowledge the first assignment shell to use the propose-and-exchange strategy was COKE (see Sect. 24.5). We have found few references to comparable shells in the expert system literature. Even in the case of SALT [Marcus 88], where the propose-and-revise strategy is used for simple configuration problems (see Chap. 23), no indications are given concerning possible applications to assignment problems.

For many assignment problems, however, there are special algorithms which, when applicable, usually give much better results than the propose-and-exchange strategy. The simple assignment problem with preferences illustrated in Fig. 24.1 can be solved with variants of general operations research algorithms, e.g., linear programming procedures such as the SIMPLEX method, adapted to integer problems, or the Busacker-Gowen algorithm for minimum-cost maximum-flow systems. For the latter method the assignment problem of Fig. 24.1 is converted into a flow system by adding a "source" with links to all supplies and a "sink" with links to all demands; the costs of the links are derived from the preferences and the capacity of all links between supplies and demands is always 1. Reviews are given in textbooks on operations research.

A practical example of the application of the exact technique of dynamic programming[1] to the assignment problem of test planning is given in [Ernst 90]. The object of test planning is to assign the time periods of a test rig for complex

[1]The idea of dynamic programming is to store important intermediate results in order to avoid having to calculate them again later. In this case different paths leading to the same node in a search tree are recognized and only the best one is followed further.

tests in such a way that the re-equipment time between the tests is a minimum, taking into consideration time preferences of the groups carrying out the tests (e.g., one group may require two periods, but at least two days apart, in order to have enough time to evaluated the results of the first test). Since the tests are planned for a week in advance and the day is divided into three periods of four hours, we have an assignment problem with 15 supplies and 15 demands. It would take too long to try all 15! assignment plans, but dynamic programming reduces the complexity to such an extent that only $2^{15}$ different states have to be investigated, which was found to give an acceptable response time and still guaranteed the optimum solution. (Since the generation of the search tree took into account the time preferences, considered to be non-relaxable, even fewer states had to be tested). However, the problem with exact time-assignment techniques is that they can guarantee the optimum solution only for a single test rig, whereas in practice many different test rigs are used in parallel. They are largely independent, but not completely so, because test equipment can be exchanged between them. (A test rig consists of many pieces of test equipment which are needed in different arrangements for the tests). An attempt to find the optimum solution for all test rigs fails because of the exponential growth of the search space. The problem was avoided by generating the test plans locally for the rigs in the order of their priority, but the propose-and-exchange strategy could possibly be of use for the global problem.

Finally, it should be mentioned that many programs are commercially available for special assignment applications such as timetable generation. However, since these programs have a fixed object structure the special features of a particular school can often be represented only very imperfectly, so that an adjustable knowledge representation, as provided by a general tool for the propose-and-exchange strategy, has clear advantages.

## 24.5 Example: COKE and REST

COKE is a problem-specific expert system tool for the propose-and-exchange strategy with a form-orientated knowledge acquisition component [Poeck 90]. Compared to the knowledge representation and manipulation described above, COKE contains some extensions. For example, the proposals can be set tentatively, and the best proposal is taken as the one for which the weighted sum of violated constraints is a minimum. In addition, the above-mentioned Busacker-Gowen algorithm, in the specialized version for assignment problems, is implemented in COKE. As an exact method for problems with boundary conditions which are not more complex than preferences, it is usually superior to the propose-and-exchange strategy.[2] A knowledge base in COKE is constructed as follows:

[2]Not, however, for assignment problems in which it is better to have as many individual assignments with a high preference as possible, rather than a technically optimum solution with many medium preferences (i.e., it may be pragmatically better to have a mixture of good and bad offers than many mediocre ones).

1. definition of the object types in a graphical hierarchy,
2. specification of the attributes for each object type with a special form,
3. specification of the value range for each attribute in a special form,
4. assignment of inverse attribute pairs with tables,
5. definition of the procedures in procedure-specific forms,
6. assignment of the procedures to the attributes which call them with tables.

Various applications have been constructed with COKE, the most complex being REST for the generation of timetables. Since it has the tool COKE in the background, REST can easily be adapted to take into account school-specific boundary conditions. In its standard form REST has ten object types (periods, lesson group, lesson, block group, block, teacher, class, classroom, subject, day and hour), which have between 5 and 10 attributes each, and numerous predefined procedures. It provides mechanisms for processing the following main types of boundary conditions:

- No multiple assignment of lessons to any one teacher, class or room in the same period.
- Priorities for the available periods (e.g., as few lessons as possible in the afternoon).
- Information about the availability of teachers and classrooms, with priorities (e.g., teacher X cannot teach in the first or second period on Monday, or would like a free day).
- Indication of blocks, i.e., lessons which must be held parallel to one another (above all when a class has to be divided, e.g., for religion).
- Avoidance of free periods for the classes and, with lower priority, for the teachers.
- A class should not have more than one lesson in a given subject each day, except for intended double periods. Double periods may begin only at certain times.
- Suitability of rooms for subjects.

The input for REST is a duty list with entries of the form "teacher X gives $n$ lessons of subject Y to class Z" (see Fig. 24.2) and the boundary conditions. For the input of the duty list the user first defines his object sets as instances of the given object types. All attributes which express relationships to other objects are entered in tables.

The generation of the timetable is divided into three sequential, partial assignment problems by the agenda control:

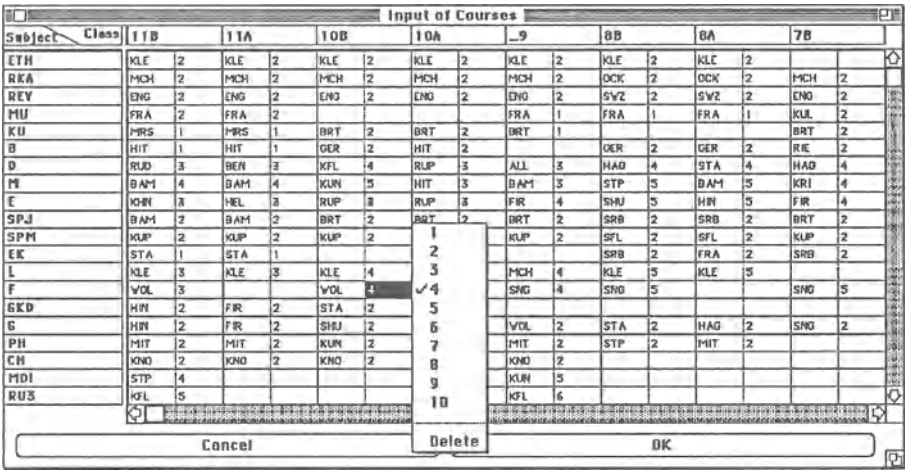



**Figure 24.2** A table for entering the courses in REST. Each pair of entries in the table corresponds to a course, consisting of a *class,* a *teacher,* a *subject* and a *duration.* For example the selected entry in the figure means that the teacher Wol[lengerger] is to give 4 lessons of F[rench] in class 10b. Each course leads to several *units,* depending on the duration (e.g. 4 units of French in class 10b in the above example).

1. Assignment of classes to classrooms. One classroom is assigned to each class.

2. Assignment of lessons to rooms. Each lesson which cannot take place in the classroom is assigned to another suitable room, multiple assignments being avoided by constraints.

3. Assignment of lessons to periods. Each lesson is assigned to a period, which means that the teacher, the class and the classroom concerned are automatically assigned to the same period. The result is a set of timetables for all teachers, classes and classrooms (see Fig. 24.3)

REST also provides a form for the interactive generation or subsequent modification of a timetable.

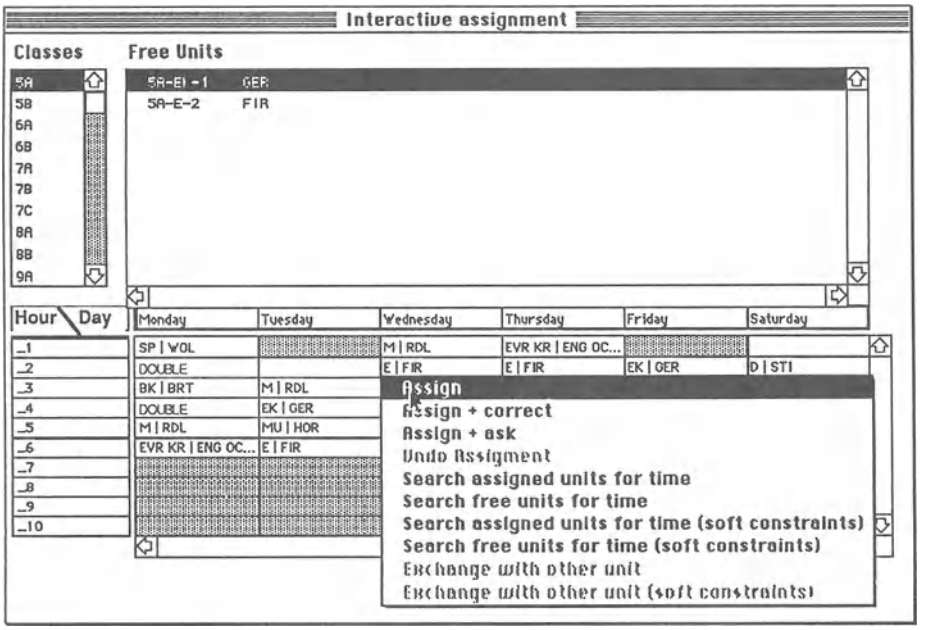



**Figure 24.3** Resulting timetable for the class 5a constructed by REST. It may be manually corrected by the user as indicated by the options in the pop-up menu. The table entry "$X \mid Y$" means that subject $X$ is taught by teacher $Y$. "DOUBLE" means a double period of the subject and teacher above. More than one entry in a box means splitting of the class and usually merging with split parts of other classes, where the corresponding courses must be taught in parallel.

# 25. Least-Commitment Strategy

Like the propose-and-revise strategy, the least-commitment strategy is suitable for deriving the values of a set of parameters, but it avoids a rapid, temporary, heuristic determination of parameter values. In particular, there are no loops to be broken as is Fig. 23.1. The knowledge about the relationships between the parameters is represented in the form of undirected constraints instead of directed rules. The least-commitment strategy is therefore useful only when reasonably efficient constraint programming is possible. The most important constraint propagating techniques [see Chap. 7] are:

- Propagation of fixed values. It is generally assumed that $(n-1)$ of the $n$ variables of a constraint are known.
- Propagation of sets of values (example see Sect. 25.1). This is useful above all for small, symbolic parameter value ranges.
- Propagation with case distinction. Case distinctions are meaningful only for limited, symbolic parameter value ranges.
- Symbolic Propagation. This requires powerful algebraic techniques for transforming symbolic expressions and is therefore used only rarely and in simplified form in expert systems.

The above list shows that efficient constraint programming is dependent upon strict boundary conditions, which are not immediately satisfied in many domains. However, in many cases it is possible to re-formulate the problem in such a way that it can be solved by the above constraint propagation techniques. The expert knowledge then consists to a large extent of the generation of suitable constraints from the original formulation of the problem. The combination of the generation of a relatively simple constraint network and its evaluation by constraint propagation is the idea behind the least-commitment strategy.

The following is an example, from production planning, of a constraint-generating rule which has statements about problem features as its precondition and the generation of a simple constraint as its action: *If the relative tolerance of two surfaces of two different machining elements is less than a threshold value, then both elements must be machined without removing the workpiece from the chuck. (Priority: high).* These rules differ from our usual rules (see Sect. 12.2) in that they work with "for-all" quantified variables and are instantiated several times

with different values for the variables. They therefore require an efficient rule interpreter with pattern matching which generates as the result a set of constraints with instantiated reference parameters, so that no "for-all" quantified variables need be considered in the constraint propagation. Since the rule language for the generation of constraints depends upon special features of the domain, the knowledge representation cannot be specified exactly. This applies also to the constraint propagation phase if the standard techniques fail and have to be replaced by other methods (e.g., special algorithms or even the propose-and-revise or propose-and-exchange strategy). The least-commitment strategy is therefore a relatively weak problem-solving method which is not on the same level as the problem-solving methods discussed so far. The following description of the knowledge representation and manipulation is intended to give only the basic ideas and may need to be extended considerably for particular cases.

## 25.1 Knowledge Representation

The knowledge representation consists of (a) parameters, (b) rules for generating constraints on the basis of given parameter values and (c) the constraints which serve to determine the values of the remaining parameters. In analogy to the distinction between constraint-generating rules and instantiated constraints, it is also useful to make a distinction between parameter classes and instantiated parameters. The instances of the parameters inherit the general properties and are given a specific value by the user. The constraint-generating rules require a rule language with "for-all" quantified variables which refer to the parameter classes. The constraints can be represented as sets of (simple) rules, as tables with compatibility relationships (e.g., $X$ is compatible with $A$ or $B$ or $C$) or as domain-specific functions. It is important that they must be able to propagate not only fixed values but also value sets (e.g., if the current value sets $A = \{x, y, z\}$ and $B = \{v, w, x, y\}$ are known, the constraint $A = B$ gives $A = B = \{x, y\}$) and intervals (e.g., the function $X = Y + Z$ can process the partial information $Y > 5$ and $3 < Z < 6$ to give $X > 8$). In addition, the constraints, and possibly the parameters if they represent user requirements, have priorities. We thus have the following knowledge representation:[1]

1. Parameter class
    - name
    - value range
    - type (requested from user, derived or both are possible)
    - instances (d) (all instantiated parameters of the parameter class)
2. Parameter

[1]Dynamic attributes are marked (d).

- name
- parameter class (d) (pointer to class to which the parameter belongs)
- constraints (d) (affected by the parameter)
- current-value-range (d)
- pre-setting (d) (value or value range pre-set by the user or the system)
- priority (d) (of the pre-setting)
- alternatives (d) (other possible pre-settings with priorities)
- tested-pre-settings (d) (already tested and revised pre-settings with priorities)

3. Constraint-generating rules
   - precondition
   - action
4. Constraints
   - name
   - pointer (to the generating rule)
   - parameters (to which the constraint refers)
   - priority (of the constraint)
   - type (table, rules or functions)
   - contents (pointer to table, rules or functions)
   - status (d) (active, [inactive], removed)

## 25.2 Knowledge Manipulation

The user (or another program) first specifies the number and the types of the parameter classes, whereupon the corresponding parameters are instantiated and some values are requested from the user. With this data base a forward-chained rule interpreter is called, which generates as many constraints as possible. The evaluation of the constraints is carried out first with a constraint-propagating process. If the solution space is over-determined, which is shown by the fact that a constraint cannot be fulfilled by the current parameter values, the contradiction must be resolved either by removing the constraint or changing at least one of the parameter values. A parameter value may be given by the user or derived by case distinction or from other constraints. If the different alternatives are weighted, the parameter with the lowest priority of those which can resolve the contradiction should be revised. With the aid of the pointers "constraints" of the parameters and the pointers "parameters" of the constraints, the basic parameters involved in the contradiction (whose attributes "pre-setting" and "priority" have been

assigned) and the revisable constraints (whose attribute "priority" is set) can easily be found and arranged in order of priority.

The constraint propagation terminates either when there are no more applicable constraints or when no constraint causes any change in the value range of any parameter. The result is the current value set of the parameters. If only one value is left in the attribute "current-value-range" of each parameter, these values are output as the result. If there are still untested alternatives of case distinctions, more solutions may be generated by processing them. If some constraints or user pre-settings have to be revised during problem solving, these are also output with their priorities. However, if there are more values left in the "current-value-range" of the parameter than can be reasonably treated with case distinctions, the solution space is under-determined, and a solution must be generated in a second step with a separate, possibly domain-specific algorithm. The second step can take advantage of the value range restrictions of the constraint propagation. The procedure may be summarized as follows:
Input: number and types of parameter classes, pre-settings of some parameter values.
Output: unique values of all parameters.

1. Input of the number and types of parameter classes.
2. Instantiating the parameters, requesting some pre-settings.
3. Forward-chained processing of the constraint-generating rules to produce the constraints.
4. Evaluation of the constraints by constraint propagation. If the current-value-range of a parameter is empty, the contradiction is resolved either by revising pre-set parameter values or by removing constraints, taking priorities into account.
5. If no unique solution is found, i.e., the value ranges of some parameters have more than one value, a special algorithm is used to generate the unique solution.

## 25.3 Knowledge Acquisition

No graphical aids for the knowledge acquisition will be given, because the least-commitment strategy, in the general form described here, is too weak a problem-solving method to offer assistance in knowledge acquisition.

## 25.4 State of the Art

The least-commitment strategy, being a weak problem-solving method, has a wide range of possible applications, but it is often not particularly helpful if no

constraints suitable for evaluation by the usual constraint propagating technique are available or can be generated. This is often the case, for example, in model-based, non-linear planning systems (see Sect. 26.3).

An early example of an expert system which works successfully using the least-commitment strategy with a constraint-generating phase and a constraint-evaluation phase is Stefik's MOLGEN[2] [Stefik 81]. It is used for planning experiments in molecular genetics, a rough plan of operators being generated first by means-ends analysis (see Sect. 26.2). In the hierarchical refinement of the operators, the objects concerned are not instantiated immediately, but constraints are generated which usually express compatibilities, e.g., that a particular bacterium is compatible with a particular vector, the vector fits a certain gene, etc. The knowledge about basically possible compatibilities between the objects appearing in the knowledge base is stored in tables. When all constraints have been generated they are evaluated to determine the identities of the objects. For the constraint propagation in Stefik's MOLGEN, a simple algorithm is sufficient, which propagates only values and value sets and can perform case distinctions. The removal of constraints or user requirements is not needed.

GARI [Descotte 85] is a more complex expert system which generates production plans with the least-commitment strategy. Starting from workshop and workpiece data, constraint-generating rules with priorities are used to produce weighted constraints which are then evaluated to give the production plan. Unlike Stefik's MOLGEN, it may happen that the application of a constraint leads to a contradiction. Since the user requirements cannot be changed, a constraint must be removed to resolve the contradiction. For this purpose GARI always chooses the constraint with the lowest priority. If several values remain for parameters at the end, a set of possible plans are represented. GARI guarantees that these plans are all optimal in the sense that there are no other plans which violate only constraints with a lower priority.

## 25.5 Example: ExAP

ExAP [Dyckhoff 89] is a partially implemented expert system for production planning. The input is a description of the workpiece, and the output is a production plan with a sequence of operations which requires as few changes in the clamping of the workpiece on the machine as possible. The workpiece is described largely on an abstract level with machining elements, which can, however, be derived theoretically from the CAD base files. ExAP recognizes the machining elements surface, groove, pocket, breakthrough, step, recess, corner, boring, thread, through boring, through thread and depression. The machining elements and the workpiece as a whole are represented as object classes which have different attributes. A step, for example, has an orientation with respect to the different

[2]Friedland's MOLGEN (see Sect. 22.5) has the same field of application and also largely the same knowledge base as Stefik's MOLGEN, but uses skeletal planning as the problem-solving strategy. Unlike Stefik's MOLGEN it was developed further.

surfaces and certain geometrical dimensions. Each machining element is assigned to an operation which produces it. The production plan must specify the order of these operations and the machine configuration (above all the clamping).

First the object classes are instantiated on the basis of the workpiece data. For example, if the workpiece has five steps, five objects of the class steps are created. Some of the attribute and parameter values are assigned directly, others are calculated from the already assigned values by means of rules. Then the operations are created for each machining element, which is little more than a re-naming process. For each operation the machine, the workpiece clamping and the tools must be specified. In ExAP only one type of machine is considered (since one machine is generally sufficient for producing a workpiece) and the process is optimized with respect to changing the clamping of the workpiece (it is assumed that the change of tools causes less trouble on the machine used). The number of possible machine clampings is known from the start. On the basis of the attributes of the machining elements constraints are then generated for the operations. The following constraint classes are used:

- (possible C1 {OP1,...,OPn}): The operations {OP1,...,OPn} can in principle be carried with the clamping C1.
- (before OP1 OP2): The operation OP1 must be carried out before OP2.
- (same-phase OP1 OP2): Both operations must be carried out in the same phase, i.e., in the same clamping.
- (dependent OP1 {OP2,...,OPn}): If possible, OP1 must be carried out in the same clamping as one of the operations {OP2,...,OPn}.
- (blocks OP1 {C1,...,Cn}): When the operation OP1 is carried out, the clampings {C1,...,Cn} are impossible.

When the constraints are evaluated in ExAP it is found that the solution space is not over-determined as in the case of GARI, but under-determined, so that additional assumptions are required. ExAP therefore first finds the smallest set of clampings with which all operations can be carried out, regardless of their order, and then tries to determine the precise assignment of the operations to the clampings by propagation of all constraints. If a contradiction is found, such that an operation cannot be performed with the clampings so far available, a new clamping is added to the set.

The way ExAP works is illustrated below on the example of a production plan for the workpiece shown in Fig. 25.1 [Dyckhoff 89, Sect. 5.4].

Figure 25.2 shows the assignment of the machining elements to the operations and the constraints governing the order, which were instantiated from the exact properties of the machining elements by forward-chained evaluation of the constraint-generating rules and the data base.

The operations which must be carried out in the same clamping can be joined together to reduce the complexity. For this purpose four new operations are introduced: OP-26 (replaces OP-6 and OP-11), OP-27 (replaces OP-8 and OP-23),

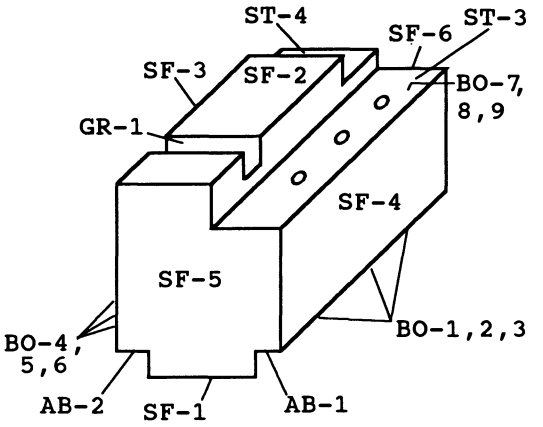



**Figure 25.1** Example of a workpiece with 20 machining elements: 6 surfaces SF-1,...,SF-6, 4 steps ST-1,...,ST-4, 1 groove GR-1 and 9 borings BO-1,...,BO-9

OP-28 (replaces OP-9 and OP-24) and OP-29 (replaces OP-22 and OP-25). The list of operation/clamping constraints generated is shown in Fig. 25.3.

The result of the constraint evaluation produced by ExAP is shown in Fig. 25.4. From this we obtain the following order for the machining of the elements shown in Fig. 25.1:

Clamping C-26: SF-3, SF-5
Clamping C-13: SF-1, SF-4, ST-1, BO-1,2,3, ST-2, BO-4,5,6
Clamping C-5: SF-2, GR-1, ST-3 (coarse), ST-3 (fine), BO-7,8,9
Clamping C-36: SF-6, ST-4.

| ME | OP-i | (before OP-i OP-j) | (sp OP-i OP-j) | (dep OP-i OP-j) |
|---|---|---|---|---|
| SF-1 | OP-1 | (OP-8,9) | – | (OP-8,9) |
| SF-2 | OP-2 | (OP-7,11,21,22) | – | (OP-7,11,21,21) |
| SF-3 | OP-3 | (OP-9,21,22,24) | – | (OP-9,22,24) |
| SF-4 | OP-4 | (OP-8,23) | – | (OP-8,23) |
| SF-5 | OP-5 | (OP-7,23,24,25) | – | (OP-7,23,24,25) |
| SF-6 | OP-6 | (OP-11) | (OP-11) | – |
| GR-1 | OP-7 | – | – | – |
| ST-1 | OP-8 | (OP-23) | (OP-23) | – |
| ST-2 | OP-9 | (OP-24) | (OP-24) | – |
| ST-3 | OP-21 | (OP-7,11,22) | – | – |
| | OP-22 | (OP-25) | (OP-25) | – |
| ST-4 | OP-11 | – | (OP-6) | – |
| BO-1,2,3 | OP-23 | – | (OP-8) | – |
| BO-4,5,6 | OP-24 | – | (OP-9) | – |
| BO-7,8,9 | OP-25 | – | (OP-22) | – |

**Figure 25.2** List of order constraints generated for the operations (OP) assigned to the machining elements (ME) (sp = same-phase, dep = dependent, see above)

| Clampings | | Possible | Clampings | | Possible |
|---|---|---|---|---|---|
| No. | Descr. | Operations | No. | Descr. | Operations |
| C-1 | SF-1/3/5 | OP-2,7,21,26 | C-25 | SF-4/1/5 | OP-3 |
| C-2 | SF-1/3/6 | OP-2,5,7,21 | C-26 | SF-4/1/6 | OP-3,5 |
| C-3 | SF-1/4/5 | OP-2,7,26 | C-27 | SF-4/2/5 | OP-3 |
| C-4 | SF-1/4/6 | OP-2,5,7 | C-28 | SF-4/2/6 | OP-3,5 |
| C-5 | SF-1/5/3 | OP-2,4,7,21,29 | C-29 | SF-4/5/1 | OP-3,2,7 |
| C-6 | SF-1/5/4 | OP-2,3,7,21 | C-30 | SF-4/5/2 | OP-3,1,28 |
| C-7 | SF-1/6/3 | OP-2,4,21 | C-31 | SF-4/6/1 | OP-3,2 |
| C-8 | SF-1/6/4 | OP-2,3,21 | C-32 | SF-4/6/2 | OP-3,1 |
| C-9 | SF-2/3/5 | OP-1,28 | C-33 | SF-5/1/3 | OP-4 |
| C-10 | SF-2/3/6 | OP-1,5,28 | C-34 | SF-5/1/4 | OP-3 |
| C-11 | SF-2/4/5 | OP-1,27 | C-35 | SF-5/2/3 | OP-4,26 |
| C-12 | SF-2/4/6 | OP-1,5,27 | C-36 | SF-5/2/4 | OP-3,26 |
| C-13 | SF-2/5/3 | OP-1,4,27,28 | C-37 | SF-5/3/1 | OP-2,7,26 |
| C-14 | SF-2/5/4 | OP-1,3,27,28 | C-38 | SF-5/3/2 | OP-1,7,26 |
| C-15 | SF-2/6/3 | OP-1,4 | C-39 | SF-5/4/1 | OP-2,7,26 |
| C-16 | SF-2/6/4 | OP-1,3 | C-40 | SF-5/4/2 | OP-1,7,26 |
| C-17 | SF-3/1/5 | OP-4 | C-41 | SF-6/1/3 | OP-5,4 |
| C-18 | SF-3/1/6 | OP-4,5 | C-42 | SF-6/1/4 | OP-5,3 |
| C-19 | SF-3/2/5 | OP-4,21 | C-43 | SF-6/2/3 | OP-5,4,7 |
| C-20 | SF-3/2/6 | OP-4,5,21 | C-44 | SF-6/2/4 | OP-5,3,7 |
| C-21 | SF-3/5/1 | OP-4,2,21,29 | C-45 | SF-6/3/1 | OP-5,2,7 |
| C-22 | SF-3/5/2 | OP-4,1,27 | C-46 | SF-6/3/2 | OP-5,1,7 |
| C-23 | SF-3/6/1 | OP-4,2,21 | C-47 | SF-6/4/1 | OP-5,2 |
| C-24 | SF-3/6/2 | OP-4,1 | C-48 | SF-6/4/2 | OP-5,1 |

**Figure 25.3** List of the allowed clamping/operation combinations for a certain machine (the exact definition of the clamping is not given)

1st phase:

- clamping: C-26
- operations: OP-3, OP-5

2nd phase:

- clamping: C-13
- operations: OP-1, OP-4, OP-27, OP-28

3rd phase

- clamping C-5
- operations OP-2, OP-7, OP-21, OP-29

4th phase

- clamping C-36
- operations: OP-26

**Figure 25.4** Finished production plan with three clamping changes

# 26. Model-Based Planning

Heuristic planning is restricted to the critical aspects of the planning process and requires empirical knowledge about which aspects are the critical ones and how to represent them, but no such reductions are performed in model-based planning. Instead, the objects of the field of application are described with sufficient accuracy, so that the operators may be completely characterized by their effects on the objects. A feature of model-based planning is the inherent possibility of a simulation of the finished plan by showing how the execution of the operators transforms the starting situation step by step into the target situation. This possibility is lacking in heuristic planning, because the effects of the operators on the situations are usually not known with sufficient detail.

The price for the generality of model-based planning is an extremely large search space, consisting of all possible sequences of operators. The most important techniques for reducing the search space are modularization and abstraction. Modularization is the division of a planning problem into a number of intermediate problems which can be solved largely independently of each other. In the abstraction technique a rough plan is first set up with abstract, simplified operators, and its parts are then concretized.

Unfortunately, these techniques bring new problems with them: in modularization interactions may mean that the solutions of the intermediate problems do not add up to a solution of the overall problem, and in abstraction it may turn out that a coarse plan which has been developed cannot be refined.

An analogy which is often helpful and at least shows the complexity of the planning problem is the comparison with programming. The objects of the field of application correspond to the objects of the programming language, and the operators to the programming instructions. A general problem-solving method for planning would correspond to an automatic program generator, which is inconceivable at present.[1] In fact, only relatively simple example problems can be solved with present planning methods. These problems often come from the Blocks World, which has become established as a standard field of application of planning.

Since the complexity of problems which can be solved by general planning techniques is usually too low for practical expert system applications, although

[1] However, many planning problems seem to be somewhat simpler than programming problems in that a plan is usually conceived for a concrete problem, rather than for a class of problems defined by the possible input data, thus requiring less parametrizing.

the techniques can become very complicated on account of their generality, this chapter will be restricted to a general overview. NOAH [Sacerdoti 77] and SIPE [Wilkins 84] are fully developed classical planning systems. Good introductions to planning can be found in [Nilsson 82, Chaps. 7 and 8], [Cohen 82, Chap. XV] and [Hertzberg 86]. Amongst many current research activities in the field of planning we may mention [Ginsberg 88] and [Wilkins 88]. A review of the literature and a comprehensive, detailed description of general planning strategies are given in [Hertzberg 89].

## 26.1 Non-Hierarchical Planning

A simple formulation of planning problems is the representation as a control system. The description of the states of the objects corresponds to the data base, and the operators correspond to the rules. The target situation can be derived by backward chaining of rules (e.g., in PROLOG), the sequence of applied rules being the required plan. The precondition of a rule describes the situation in which the operator is applicable, and the action gives the positive and negative effects (see Fig. 26.1).

Unfortunately, this strategy works only if different intermediate plans, which are often given by the fulfillment of several preconditions for one operator or for the overall target, do not interact. An example of such interactions is given by the Sussman anomaly shown in Fig. 26.1, where the two resolved intermediate plans for "stack C on B" and "stack B on A" do not represent a solution of the overall problem in this order, because the effects of the first step must be undone in order to carry out the second step.

Interactions must be recognized and resolved. For recognition purposes "protections" may be used, which are attached to situation features which may not be changed, such as those required in the target situation or as the precondition of an operator from the production until its execution (e.g., in order to protect the result of the first intermediate step: C is stacked on B).

The most obvious way of resolving interactions is to change the order of the operators. However, since there are $n!$ permutations of $n$ operators, not all can be tried. Two efficient, though not complete methods are: 1) Only the order of the intermediate plans is changed. 2) The problematical intermediate plan is displaced within the overall plan so that it precedes the generation of the violated protection; this is more flexible than the first method. If these methods fail, the violated intermediate plan can be generated a second time. Since endless loops may result (e.g., in the case of two intermediate plans which exclude each other), they must be detected (e.g., as a periodically recurring sequence of operators).

Apart from the fact that the operator sequence generated may not be the optimum, non-hierarchical planning suffers from the concentration on details, like discussing the first steps of a long journey. In practice, therefore, the techniques described in the following two sections – hierarchical and non-linear planning – are used almost exclusively.

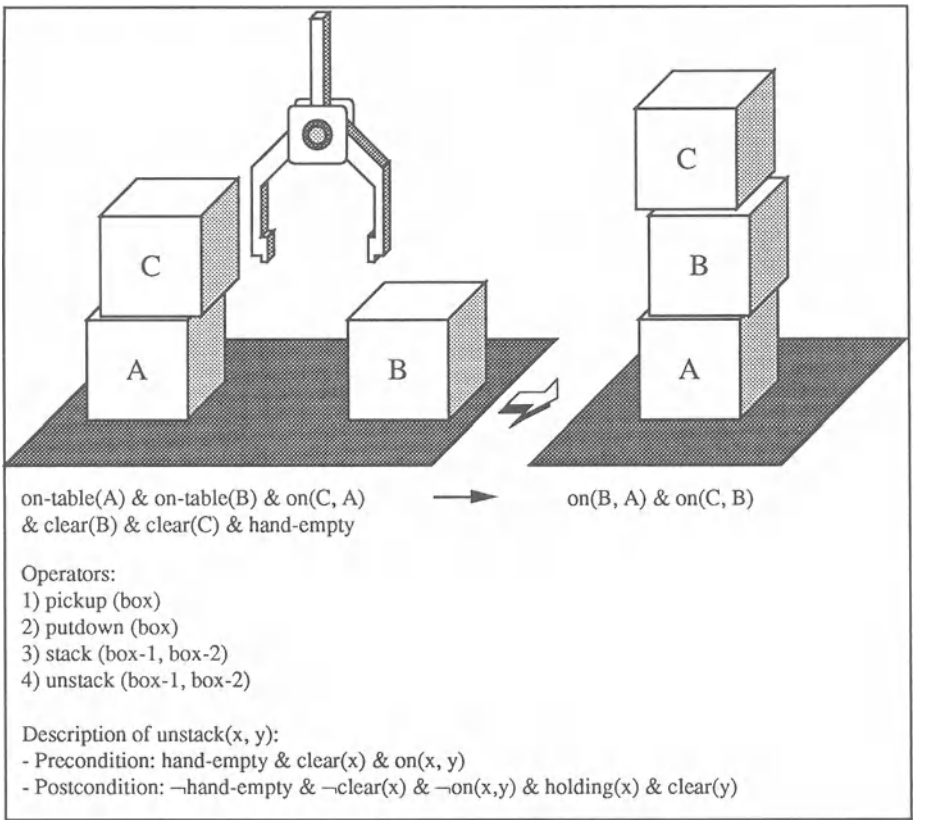



**Figure 26.1** A robot planning problem from the Blocks World

## 26.2 Hierarchical Planning

Hierarchical planning requires empirical knowledge about good abstractions, in order to provide the basic possibility of refinement. There are two possibilities for abstraction: (a) planning with abstract situation features, whereby the features are weighted and at first only the most important ones are derived by operators, and (b) planning with abstract operators, which are not directly executable but must be resolved into simpler operators. Since operator abstraction usually implies situation abstraction, because abstract operators generally refer only to important features, both forms are described below together.

A plan is first generated on the most abstract level. For each abstract operator a subsidiary plan is then generated, having the precondition of the abstract operator as starting situation and its postcondition as target situation. This procedure is iterated until only primitive operators remain. An important advantage of hierarchical planning is that the definition of "primitive operators" can easily be changed in order to adjust the degree of detail of the plan to the previous knowledge of the user.

If interactions are detected, one may try to resolve them with the same methods as in non-hierarchical planning, i.e., by changing the order of the operators first on a high abstraction level or displacing an intermediate plan to precede the protected feature which was violated independently of the level of abstraction.

Instead of the backward-directed search it is sometimes more advantageous first to seek an abstract operator for the situation feature which is most difficult to fulfill, and then to expand the plan in both directions, i.e., from the starting situation to the precondition of the abstract operator and from its postcondition to the target situation. This "means-ends analysis" is based on the attachment of weighted situation features to operators. If the situation features are not given directly in the formulation of the problem, they can be generated from a comparison of the starting and target situations.

The performance of a hierarchical planner depends upon the quality of the abstractions. Since the description of the situation has to be greatly simplified for the abstract operators, there is a continuous transition to heuristic planning. The main difference is in the treatment of interactions.

## 26.3 Non-Linear Planning

In the techniques described above, interactions are resolved by subsequent rearrangement of an ordered plan. In non-linear planning, the determination of an order is avoided as long as there is no compulsive reason for it. In deriving several intermediate plans for an overall plan, or several preconditions for an operator, they are treated as a set with no internal order and are synchronized only as a prerequisite for the higher-ranking target. After each extension of the plan it is checked whether there are interactions between the new operators or between them and already existing operators. If so, a sequence restriction, i.e., an additional synchronization, is determined for the affected operators. Thus the two phases – plan analysis and plan extension – are repeated in an alternating sequence. In the plan analysis phase it is also possible to attempt optimization; for example, if two identical preconditions are used in different intermediate plans, one can be eliminated. If operators remain unordered in the overall plan, they can be processed in an arbitrary order or in parallel. Non-linear planning can therefore be regarded as a variant of the least-commitment strategy in which constraints (i.e., sequence restrictions) are generated during the actual planning phase and subsequently evaluated by constraint propagation. This idea can be generalized by keeping not only the order but also the attributes of the operators variable as long as possible in the plan, i.e., the interactions with other objects are also formulated as constraints (cf. Stefik's MOLGEN, Sect. 25.4).

# 27. Case-Comparing Construction

In case-comparing construction a case library is searched in an attempt to find a case which is as similar as possible to the task at hand, so that its solution can be adopted. Unlike the problem type classification, the solutions of two different problems seldom agree in construction, so that the old case must be modified according to the special requirements of the new one. For this purpose an additional knowledge representation is required. Case-comparing construction should not therefore be regarded as an independent problem-solving method, but rather as an extension of the construction method whose knowledge representation it uses. Therefore in this chapter only the special case-comparing features are described. Case-comparing construction has two phases: the selection of the old case and the adoption or modification of its solution.

If desired, the old case can be given by the user. Otherwise it is selected by a similarity comparison with reference to the current task, corresponding to the case selection in classification (see Chap. 20), which will also not be repeated here.

There are two fundamental possibilities for modifying the basic technique to deal with the adoption or modification of the solution:

- The old case serves as an aid for decision-making in the basic technique. First a construction step is tried as in the old case. If this does not work, the construction step is carried out according to the knowledge of the basic method. In skeletal construction the refinements in the AND/OR hierarchy are taken over from the old case, as long as they do not contradict existing rules. In the propose-and-revise strategy the proposals come from the old case instead of from the proposal rules. With the least-commitment strategy, however, there is not much room for adjustment to the old case, since the constraints should be evaluated in any case, and the comparison can only be used to restrict any remaining freedom of decision.

- The other alternative is to adopt the solution of the old case completely and then to correct it according to the differences between the requirements of the two cases. For this purpose the propose-and-revise strategy is particularly suitable, where the monitoring constraints detect contradictions and the correction rules effect specific improvements to the adopted solution. With the other methods, which do not have correction rules, there remains only the possibility of adjusting the solutions with a general TMS, which

makes changes to remove contradictions but does not optimize them in the same way as specific correction rules.

In the approaches discussed so far the cases themselves are not subjected to any further analysis. This requires that there must be enough cases to be able to find one which has enough resemblance to the new problem, and also that the stored solutions do not contain errors. The preparation of tenders for simple system configurations and the production planning of workpieces are domains where these conditions are likely to be fulfilled, since the number of cases is usually large, and the solutions are subsequently checked. In many other domains, however, and in particular in everyday planning, these conditions are rarely fulfilled. Possible learning methods are the generation of skeletal plans from one or several cases with the technique of explanation-based learning or the analysis of faulty plans, in order to learn from the errors.

Case-comparing construction is a still young, but rapidly expanding field or research. Amongst the pioneers are Janet Kolodner, in whose research group the systems MEDIATOR [Simpson 85, Kolodner 87b], for the case-based analysis from differences of opinion in discussions, and JULIA [Kolodner 87a,b], for the case-based planning of menus for guests with different dietary wishes, and Kristian Hammond, who developed the system CHEF for analysing and correcting errors in cooking recipes [Hammond 86, 87, 89].

# 28. Partial Integration of Construction Methods

The construction methods discussed above are each based on a different central idea:

- *Skeletal construction:* top-down refinement along an AND/OR hierarchy.
- *Propose and revise, propose and exchange:* direct, heuristic determination of parameter values or assignments with the possibility of knowledge-based correction, respectively.
- *Least commitment:* the use of constraint generation to avoid premature fixing of parameter values, and subsequent evaluation by constraint propagation,
- *Case-comparing construction:* adoption of as many construction decisions as possible from a similar case.
- *Model-based construction:* construction on the basis of a model which is manipulated explicitly with operators.

Further basic ideas are:

- *Bottom-up aggregation.* As the reverse of the top-down refinement used in skeletal construction, one can also imagine a bottom-up aggregation of given construction elements. Intermediate forms such as the means-ends analysis (see Sect. 26.2) are also conceivable.
- *Interactive construction.* The user makes the decision in all critical steps.

This list shows that the ideas of the construction methods complement one another and do not overlap as much as the classification methods (see Chap. 34). While the classification methods represent different ways of achieving the same aim, from which the best is selected for the concrete case, the construction methods tend rather to have different aims, i.e., they are suited to different tasks. The aim of the integration is therefore not so much to exploit competitive effects, but rather to extend the limited fields of application of the individual methods.

Thus hierarchical skeletal construction and the propose-and-revise strategy are ideally complementary. However, the combination of object refinement along

AND/OR hierarchies with the determination of property parameters of specialized or newly generated objects may result in a considerable increase in complexity if the set of objects to be handled varies dynamically, since new rules and constraints then have to be generated. On the other hand, with the generation of constraints one of the most important conditions for the integration of the least-commitment strategy is fulfilled. The latter complements the propose-and-revise strategy in that a maximum restriction of the value ranges of all parameters by constraint propagation creates a good starting situation for propose and revise. Case-comparing and model-based construction also offer attractive additions, since the former provides practically a solution pattern, if the similarity is sufficient, which only needs to be modified, and the latter allows the solutions of planning problems to be tested by simulation.

On account of the great complexity of a complete integration the dynamic generation of objects will not be considered here, and the following description of a partial integration is limited to the construction methods with the most empirical knowledge. These include skeletal construction, the propose-and-revise strategy and case-comparing construction. For the least-commitment strategy we will consider only simple constraint propagating techniques (the propagation of fixed values and possibly sets of values). We will not treat the model-based planning techniques because they are too complex; they turn up again in a rudimentary form in the overall integration (Sect. 35.1) for the simulation of completed plans. Instead we will include interactive construction as an option and a restricted form of bottom-up construction.

The construction procedure may be summarized roughly as follows: the objects are first refined along the concept hierarchies, whereby the rule and constraint network of the parameter relationships is built up and hence the parameter values determined. Any decision to be made during this process which cannot be settled unequivocally may be derived heuristically with default values or proposal rules, adopted from a similar case from a case library or determined by the user.

This "top-down strategy " must be extended if it is required to take into account available knowledge about concrete objects which are to appear in the configuration. For this purpose the concrete objects may be abstracted in the reverse direction along the hierarchy, or the selection rules for the hierarchy refinement must be changed in such a way that the concrete objects are not excluded. The latter method is provided automatically in case-comparing construction if a case is chosen which contains the desired concrete objects.

The more difficult the decisions to be made in the construction process are, the more important it is to be able to make corrections when contradictions occur, either with a general TMS or – if available – with specific correction rules as in the propose-and-revise strategy. If insufficient knowledge is available about the decisions and their corrections, an alternative is to use interactive construction, where the user makes the critical decisions.

In order to make the problem-solving method stronger, the structure of the concept hierarchies may be simplified by excluding the possibility of specifying a variable number of partial objects in decomposing hierarchies and by not mixing

specifying and decomposing hierarchies, i.e., giving both parts and refinements of an object. This does not in principle reduce the expressive power, since the case distinctions involved can also be represented by additional objects in the specializing hierarchy.

Below we outline only the basic structure of an integration which shows above all the diversity of the relational knowledge (several types of hierarchies, rules and constraints).

## 28.1 Knowledge Representation

The basic elements of knowledge representation are the concept hierarchy consisting of objects, the parameters and their relationships. An object has only parts or only refinements as its successors. A variable number of parts is allowed, but may be used only if the number of parts is of no great importance, so that a case distinction with specializing objects is not required (in configuring a conference room, for example, there is no need to have an object for every possible number of chairs). The number is represented as the parameter "number" of the part. This has the consequence that all parts must be refined in the same way, since they are represented as a single object.

The refinements of an object have the same parameters as the predecessor, but the parts of an object usually have a different parameter structure. The specializing hierarchy therefore corresponds to the usual frame structure with inheritance, attached procedures and default values. However, as in the knowledge representations of the other problem-solving methods, the parameters will be represented as independent objects rather than as attributes, the objects having pointers to their parameters, so that we require only a triple representation (object-attribute-value + parameter-attribute-value) instead of the quadruple representation (frame-attribute-facet-value). A parameter belongs potentially to several objects of a specializing hierarchy. For each refinement of an object the current value range of a parameter may be further restricted. We thus have the following attribute structure of the objects, where user requirements have been added as an additional object type. For the sake of simplicity the classification in user requirements and derived parameters is assumed to be static.[1]

1. Objects
    - name
    - specializations (successor objects which specialize the object)
    - abstractions (inverse relation to specialization)
    - has-parts (successor objects which represent parts of the object)
    - is-part-of (inverse relation to has-parts)

[1]Semi-static attributes, which can be derived from the normal (static) attributes by the knowledge acquisition component, are marked (s), and dynamic attributes (d).

- parameters (list of the parameters of the object)
- selection-rules (s) (list of rules governing specialization)
- decomposition-rules (s) (rules which set the parameters of the parts on decomposition)
- restriction-rules (s) (rules which restrict the parameters on specialization)
- value (d) (instantiated, not instantiated)
- selection-justification (d) (rule which justifies the instantiation of the object)

2. parameters

- name
- objects (list of objects to which the parameter belongs)
- value-range
- default (value, or procedure for calculating the default value)
- object-rules (s) (rules of objects which restrict the value range)
- constraints (s) (list of constraints which connect the parameter)
- proposal-rules (s) (proposal rules for deriving the parameter value)
- correction-rules (s) (rules for correcting the parameter value in the event of contradictions)
- current-value-range (d)
- rule-justifications (d) (restrictive, triggered object rules)
- constraint-justifications (d) (restrictive, triggered constraints)
- value (d)

3. User-requirements

- name
- value range
- weight (number; for case comparison, see Sect. 20.1)
- similarity-type (for case comparison, see Sect. 20.1)
- similarity-representation (for case comparison, see Sect. 20.1)
- meaning (s) (rules or constraints with conclusions)
- value (d)

4. Old-case (complete representation of a solution of a comparative case with all intermediate construction steps)

5. Rules
    - type (selection, decomposition, restriction, proposal, correction, constraint)
    - context (object)
    - precondition
    - target-object (object or parameter affected by the action)
    - action (formula or assignment)
    - value (d) (triggered, not triggered)
6. Constraint
    - type (monitoring or restricting)
    - parameters (concerned with the constraint)
    - contents (pointer to a set of rules or table look-up)
    - correction (the correction rules for monitoring constraints)

The current state of the configuration is recorded in a global variable "configuration", which is initialized with the root node of the concept hierarchy (which is itself stored in a global variable) and extended with every decomposition and specialization through the attributes "has-parts" and "specializations".

## 28.2 Knowledge Manipulation

Starting from the requirements of the user, who specifies objects, parameters or properties of the configuration, the completion of the configuration is carried out step by step. The most important basic strategies are:

1. Top-down configuration. Starting from the root node the objects are first refined and decomposed into their parts. Then the parameters of the objects are instantiated, their value ranges are restricted as far as possible, and the corresponding constraint network is constructed and evaluated.

2. Bottom-up configuration. If the problem specification contains details of concrete objects and parameter values, these are instantiated first and all possible definite conclusions are drawn. The process is then continued with the top-down method, with the additional boundary condition that objects which have already been instantiated must be taken into account.

3. Case-comparing configuration. First a comparative case is selected which fulfills the user requirements as well as possible, or such a case is specified by the user. In all configuration decisions those of the old case are tried first.

In order to implement these three strategies or mixed forms in a system, an agenda for controlling the heuristic decisions is required. Non-heuristic actions, on the other hand, are carried out immediately. The agenda may contain the following types of action:

1. Refinement of an object with the attribute "specialization".
2. Decomposition of an object with the attribute "has-parts".
   In both these steps the value ranges of all parameters of the new objects are restricted according to the "restriction-rules".
3. Abstraction of a concrete object with the attribute "is-part-of".
4. Abstraction of a concrete object with the attribute "abstractions".
   for these two steps additional heuristics in the knowledge representation of the objects are required.
5. Heuristic determination or restriction of a parameter (the non-heuristic restrictions of parameters are always carried out automatically in connection with the actions causing them and do not therefore need to be controlled by the agenda).

There are also various possibilities for the execution of an action:

1. Adoption of an action from the old case.
2. Selection according to heuristic default rules, or adoption of default values.
3. Request of relevant requirements from the user.
4. Direct control by the user.

The more heuristic decisions have to be made, the greater is the probability that contradictions will arise. If there are correction rules these are used first in an attempt to remove the contradictions as in the propose-and-revise strategy. Otherwise the justification of each step in the problem-solving process must be stored. In the event of a contradiction the justifications are searched recursively for heuristic decisions, since these are the candidates for a revision. For this purpose it is helpful if the heuristics are weighted, since the one with the lowest value can be revised first (otherwise the most recent heuristic decision could be revised, for example). For the action concerned an alternative is then selected and executed, and the consequences are propagated through the network. This process must be repeated until a consistent state is achieved. To avoid repetitions, inconsistent combinations should be recorded in a global variable.

## 28.3 Knowledge Acquisition

The starting point for knowledge acquisition is the construction of the concept hierarchy. For this purpose a strictly hierarchical graph is suitable, with two types of links for specialization and decomposition. To improve the readability the specializing hierarchies can each be represented in a separate graph, where each decomposition leads to a corresponding number of new specializing hierarchies. Each link in the graph may be clicked with the mouse, whereupon a rule form appears, in which the "selection-rules", and possibly also the "decomposition-rules" and the "restriction-rules" may be specified. With the aid of another form local information can be given to each object in the graph; this will usually consist of the parameters and their attributes. If the same parameter is used in several places in a specializing hierarchy, it is sufficient to specify it only once for the highest-placed object in the hierarchy.

The second phase of knowledge acquisition is the determination of the relationships between the parameters in a network of rules and constraints. There are three ways of restricting the value range of a parameter: by means of "restriction-rules" on specialization, through "decomposition-rules" when instantiating parts or by parameter-parameter relationships. In all cases only a small group of parameters is normally affected, since only a few objects are concerned. This is also true for the parameter-parameter relationships, since they normally refer only to parameters which all belong to the successor objects of a decomposition. This group of parameters can be listed in a window, so that the objects can easily be referred to by clicking with the mouse when filling in rule or constraint forms.

## 28.4 State of the Art

In an integrated construction shell the variabilities of the individual methods combine in a multiplicative way. Such a shell may become very complicated, especially if the parameters and their relationships are not given explicitly in the knowledge base, but are generated dynamically during the construction. The above knowledge representation therefore assumed the simpler alternative. However, in the system which is probably the most highly developed at present, the integrated construction shell PLAKON [Cunis 89,91], the dynamic generation of parameters and constraints is possible. PLAKON integrates an AND/OR graph with top-down refinement and bottom-up aggregation, parameter calculation through default values, attached procedures and constraint propagation, case-comparing construction by adoption of construction steps from given comparative cases and also allows for interactive construction. For the revision of construction step it offers four different options: no revision, chronological revision, a JTMS algorithm and an ATMS algorithm. A specific correction corresponding to the propose-and-revise strategy is not provided, however. On account of the complexity of the knowledge representation, a graphical support of the knowledge acquisition is possible only to a limited extent.

# Part VI

# Simulation

# 29. Review of the Problem-Solving Type Simulation

In classification and construction, solutions are selected or assembled, respectively. Simulation, however, serves "only" for predicting the effects of certain assumptions on the system. A prerequisite is a model of the system containing parameters and relationships between them. The simulation consists in deriving the values of certain (output) parameters from given values of other (input) parameters.

## 29.1 Domains

Applications for simulation may be found in all fields concerned with systems. In this connection a system is regarded as a unit whose behavior can be derived from the behavior of its components. The distinction between systems and components is a matter of the viewpoint: an engine can be regarded as a system or as a component of an automobile, for example, and an automobile as a system or as a component of a traffic system. Other examples are organ systems, such as the blood circulation system in the organism, the organism as a whole, ecological systems, the cooperation of men and machines in a factory, electrical circuits, mains supply systems, etc.

With the simulation technique one can generate predictions as in weather forecasting or carry out virtual experiments as in a flight simulator. For this purpose, however, very precise, complex simulation models are generally required. Applications in expert systems have so far been restricted to relatively simple uses, e.g., to check whether solutions derived by classification or construction methods do in fact produce the required effects, to get a "feeling" for the domain (as a human problem-solver) or to perform a weak-link analysis of a system. The limitations of the simulation are determined by the model, which must be specified as a precondition for the simulation, i.e., only a limited, pre-selected set of parameters and components can be considered. This specification requires empirical knowledge, which is "important".

## 29.2 Classification According to Problem Types

Since a simulation is necessarily limited to the essential features, the problem types can be classified according to what is the essential feature: the *time*, the *parameter values* or the *parameter trends.*

When doing weak-link analysis of a factory, an administration or a traffic system, the main points of interest are how long the work to be performed is held up at each station and to what extent the capacities of the stations and the operating materials are utilized. If the utilization is too low on average, but one station or one material is utilized to 100% capacity, the bottleneck has been found. One can then start a new simulation with an altered model, to test means of overcoming the bottleneck. The actual working processes, e.g., the operations performed on a workpiece by each machine, are not simulated. Instead, only the processing times and any transport or waiting times are represented.

In other situations, e.g., the simulation of the blood circulation or an electrical circuit, one is interested in how the system reacts to external perturbations. Such influences, e.g., taking a medicine or switching on an electrical device, bring about changes in the values of certain parameters, and the reaction of the system is a change of other parameters and the adjustment to a new equilibrium state. If this takes place more or less immediately, one is not interested in the time dependence, but only in the values of certain critical parameters in the new equilibrium state.

There are still other applications where the equilibrium state does not exist, or is reached only very slowly, so that the time dependence of the parameters is important. This applies to simple periodic processes such as the oscillations of a pendulum or an electric bell, and also to some complex processes such as weather forecasting or economic trends. The typical feature is that the parameter values change with time, and depending upon these values new processes may be activated and others de-activated. Since it is hardly possible to simulate a continuous change, the parameter values are calculated for certain discrete times, selected more or less at random. The result is a sequence of states characterized by their parameter values.

## 29.3 Analysis of Problem Characteristics

The problems of simulation increase with the complexity of the system model. In the simplest model there is a component for every parameter and a simple formula for calculating the parameter from the input parameters. The following features of the simulation model contribute to increasing the complexity:

- *Combination of parameters.* A parameter is influenced by different components, the effects of which have to be combined.
- *Component states.* The components may have different possible states, in which they show different behavior. The states may be unknown at the start of the simulation, or they may even change during the simulation.

- *Uncertain knowledge about the components.* The effect of a component cannot be predicted with certainty, because the basic knowledge is lacking or the model would become too complicated. In this case it is necessary to consider various alternatives, which may be weighted with probabilities.
- *Inexact knowledge about the input parameters.* If only rough values are available for the input parameters, a simulation should nevertheless be possible, the predictions of which should reflect the uncertainties of the input.
- *Feedback loops.* Feedback loops, in which a change in a parameter causes an amplification or attenuation of the change in the output via one or more intermediate steps, complicate both the model and problem-solving method considerably.
- *Different levels of abstraction and multiple models.* It is often desirable to keep the precision of a simulation variable. For this purpose it is helpful to have the model represented on different levels of abstraction. In many cases there are different, incompatible models which lead to different results.
- *Temporal behavior.* In many cases the time required for an effect to become established is not negligible. Different time representations are required according to the precision of the data and the knowledge.

## 29.4 Problem-Solving Methods

The basic problem-solving method is forward-chained reasoning within the model in order to derive the output parameters from the input parameters. While a single determination of the output parameters is sufficient in many domains, there are other applications where one is interested in the temporal behavior of the parameters. The former type is called *single-phase simulation* and the latter *multiple-phase simulation.* According to the type of the value range of the parameters we may distinguish further between numerical, analytical and qualitative simulation. In *numerical simulation* the calculations are performed with concrete numbers, which requires exact knowledge about the relationships and the input data. The numerical values of the output parameters must then be interpreted as a result. If the user is interested in the dependence of certain output parameters on certain input values, it is sometimes better to express the dependence in the form of an equation. Such *analytical simulation* usually requires complex algebraic transformations, which severely limits the applicability. *Qualitative simulation* is a simplified form of numerical simulation, where the aim is to derive the most important conclusions despite the lack of exact numerical data and equations. For this purpose the value range is reduced to qualitative "landmark values" having a particular significance for the parameter, e.g., for the temperature of water these could be the freezing and boiling points together with the corresponding intervals.

Of the six resulting types of simulation ({single-phase, multiple-phase} × {numerical, analytical, qualitative}), numerical and qualitative, single-phase simulation will be grouped together, because their structures are relatively similar in contrast to the different phase transitions in multiple-phase simulation. In addition, analytical simulation will be omitted, because expert system techniques cannot contribute much to it, and the equations can become very complex. We thus obtain the problem-solving structure shown in Fig. 29.1.

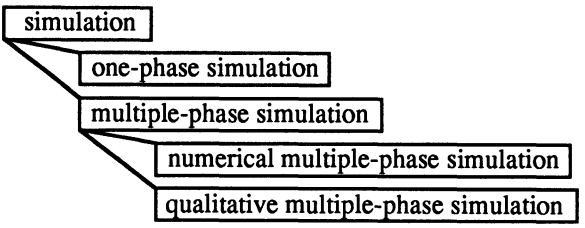



**Figure 29.1** Problem-solving methods for simulation

Single-phase simulation is especially suitable for predicting the new equilibrium state resulting from external perturbations of a system. A "bottleneck" analysis in a factory or a test of proposed changes can be done well with numerical multiple-phase simulation. The third problem type, the prediction of the temporal variations of parameter values, can be performed with both types of multiple-phase simulation; the numerical method is more exact, but the qualitative method is on the whole much simpler (with regard to the exactness of the required input data and the required knowledge).

These simulation methods are typical for expert system applications but by no means cover the wide field of simulation. For reviews and further literature on simulation reference may be made to [Round 89], [Griffin 86] and [Bratley 87].

# 30. Single-Phase Simulation

Although single-phase simulation is relatively simple on account of the lack of an explicit temporal representation, it is sufficient for many applications, since temporal relationships are often implicit. In Fig. 30.1, for example, it is obvious without an explicit time representation that the arrows on the materials represent a strict before/after relationship.

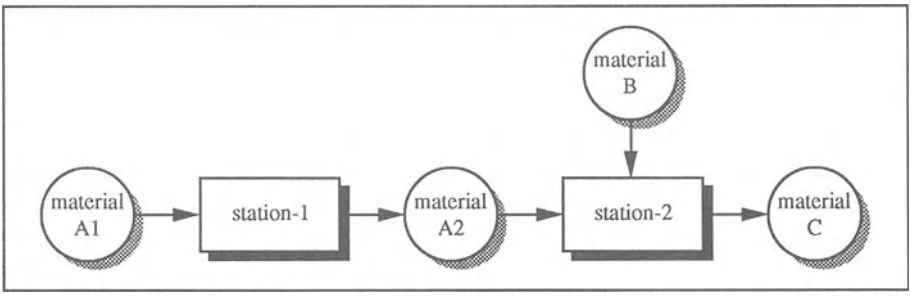



**Figure 30.1** Basic structure of single-phase simulation. The materials are manipulated by the components in the direction of the arrows (e.g., material A1 = coated cardboard, material A2 = containers, material B = milk and material C = milk in containers).

Single-phase simulation is suitable for deriving parameter values in simulation models, e.g., in the simulation of production plants for filling liquids or soldering circuit boards, in the simulation of computers or machines or in the simulation of biological experiments such as the analysis of DNA.

The basic structures of the knowledge representation, manipulation and acquisition are those of the functional models described in Chap. 18 which will not be repeated here. These structures are not only suitable for checking the solutions of classification problems, however, but also for the simulation of construction solutions (see Chap. 35). The main differences between the various applications lie in the value ranges of the properties (parameters) of the materials (cf. Chap. 18):

- For classification with functional models the typical value range was given as deviations from the normal value (from −3, i.e., "much too low", through 0, i.e., "normal", to +3, i.e., "much too high") since only the discrepancies between the actual and desired values are relevant for the result of the simulation.

- For a simulation in its own right, however, the absolute values of the parameters are of greater interest. According to the precision requirements, we may distinguish between qualitative and quantitative value ranges. While the problem-solving method is not affected, absolute values usually require a more complex model. If a system has different operating modes (e.g., driving in first, second, third or fourth gear), different rules may be needed for deriving the absolute values of the mode-dependent parameters, whereas the conditions for deviations from the normal value are the same in all modes and thus need to be specified only once.

- In action planning there are often no scaled value ranges, but only those of ordinal type. In a step of an experiment, for example, a property of a DNA may be set to a fixed value for which there is no "higher" or "lower". This also has no influence on the problem-solving method.

Apart from the systems mentioned in Chaps. 17 and 18, in which simulation is used for checking diagnoses, an important independent field of application of single-phase simulation is the analysis of circuits with resistances, diodes, transistors, etc. as component types and voltage and current as material types. However, the temporal behavior of oscillator circuits or the effect of delay components cannot be modelled satisfactorily. Well-known examples are EL [Stallman 77] and SOPHIE I, II and III [Brown 82], where the simulation model was used for tutorial purposes.

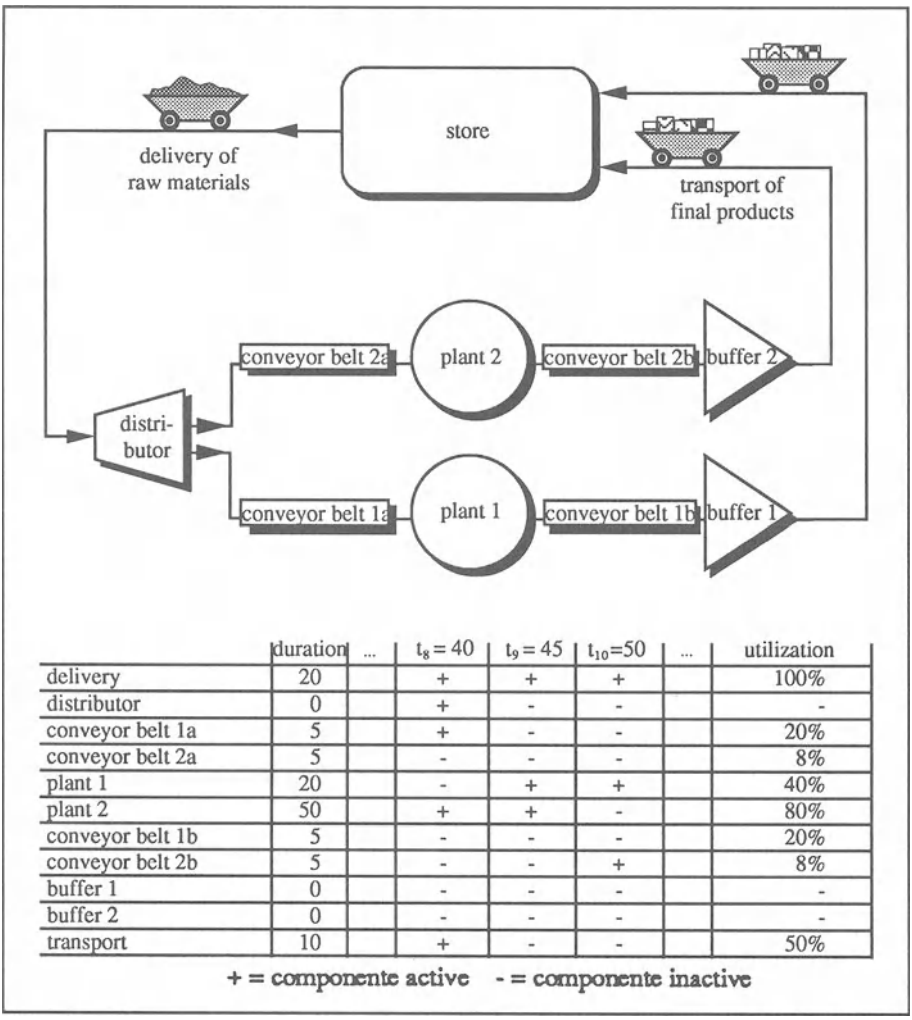
# 31. Numerical Multiple-Phase Simulation

In multiple-phase simulation the components may change their behavior during the simulation, the changes being governed by the input data or by a random number generator representing failure frequency. The different behaviors are represented by different simulation phases. In numerical simulation the phase transitions and the behavior of the components can be calculated exactly, whereas qualitative simulation does not use exact data. Numerical multi-phase simulation is a well-established, widely used aid in science and technology. We describe here only a simple basic structure, which must be extended according to the complexity of the domain.

An important application of numerical multiple-phase simulation is the simulation of production systems for bottleneck analysis. In this case we are not concerned primarily with the actual operations performed on the materials during the manufacturing process, as in the examples of single-phase simulation given in the previous chapter, but rather with the throughput times of the materials and the utilization of the machines when different processes are being carried out in parallel (see Fig. 31.1)

A central aid in multiple-phase simulation is the global clock, which advances the simulation process in discrete, numerical intervals of time. The length of the time interval is application-dependent and may be constant, e.g., 5 minutes or 24 hours, or variable. For each point on the time scale the new values of the parameters are calculated from the old values by means of arithmetical formulas.

Multiple-phase simulation alternates between the calculation and the transformation of states. Starting from the initial state with partial assignment of parameter values, the values of the remaining parameters are calculated with time-independent formulas, just as in single-phase simulation. The change of state is initiated by the advancement of the clock and brings about a change in the values of certain parameters according to time-dependent formulas (e.g., the time a material remains in a particular component). The alternation between the calculation and the transformation of states is continued until it is stopped by the user or an equilibrium state is reached in which no more time-dependent formulas are applicable. The result may be presented, e.g., as an animation or in the form of statistics of the utilization of machines or the production times.




| | duration | ... | $t_8 = 40$ | $t_9 = 45$ | $t_{10}=50$ | ... | utilization |
|---|---|---|---|---|---|---|---|
| delivery | 20 | | + | + | + | | 100% |
| distributor | 0 | | + | - | - | | - |
| conveyor belt 1a | 5 | | + | - | - | | 20% |
| conveyor belt 2a | 5 | | - | - | - | | 8% |
| plant 1 | 20 | | - | + | + | | 40% |
| plant 2 | 50 | | + | + | - | | 80% |
| conveyor belt 1b | 5 | | - | - | - | | 20% |
| conveyor belt 2b | 5 | | - | - | + | | 8% |
| buffer 1 | 0 | | - | - | - | | - |
| buffer 2 | 0 | | - | - | - | | - |
| transport | 10 | | + | - | - | | 50% |

+ = componente active - = componente inactive

**Figure 31.1** An example of a simulation model for a manufacturing process, in which the blanks are taken from a stock and fed into two plants, and the products are transported back to the store. The main weak link with insufficient capacity is the delivery.

## 31.1 Knowledge Representation

As in all simulation types the most important elements are components and materials. The materials are of specified types and have attributes which indicate where and/or in what condition they are. The components change the properties of the materials according to numerical formulas specifying e.g., how long the materials remain in the components or how their physical properties are changed by the components. The formulas may be time-dependent (e.g., material flow or differential equations) or time-independent (e.g., information flow or normal equations). In addition, the components have states (e.g., active, waiting or defective), which determine the attributes of the output materials from those of the input materials by means of rules. If some components require resources which may become scarce, the latter should also be represented. Since there are often many different components or materials of the same type, it is worthwhile defining corresponding object types. For production the most important component types would include the following:

- Fixed or movable means of transport (e.g., chutes, conveyor belts or fork-lift trucks), whose behavior in the transport of materials may be characterized by their capacity and the duration of transport. If movable means of transport are to be used for different purposes according to the utilization of their capacity, they should be represented not only as dedicated components but also as resources.
- Plants (machines), which involve a certain duration of stay of the materials. The change in the material properties need not necessarily be represented.
- Storage facilities (buffers and depots) with their capacities.
- Branches or joints in the material flow, with distribution or mixing of the materials, respectively, according to control rules.

For states, materials and parameters we may thus introduce generic objects describing their general properties, which are then inherited by the instances of the objects. However, since this does not change the structure of the knowledge representation, the following description is restricted to the attributes of the instances:[1]

1. Components
    - name
    - icon (graphical representation of the component in the simulation model)
    - type (of the component)

[1]Semi-static attributes, which can be derived from the normal (static) attributes by the knowledge acquisition component, are marked (s), and dynamic attributes (d).

- input materials
- output materials
- states (list of possible states)
- current state (d)
- utilization (d) (sequences of occupied states with durations in the course of the simulation)

2. States

   - name
   - components (pointer to components concerned)
   - icon (graphical representation as addition to or replacement of the component icon)
   - time-dependent-rules
   - time-independent-rules

3. Materials

   - name
   - icon (graphical representation of the material)
   - type (of the material)
   - parameters (for material attributes such as position, amount, working state; if there is only one attribute per material, the parameter attributes may be stored directly with the material)

4. Parameters

   - name
   - material (back-pointer to the material concerned)
   - type
   - value-range (usually numerical)
   - derivation (s) (rules for deriving the parameter)
   - meaning (s) (rules which draw conclusions from the parameter value)
   - value (d)
   - activity-procedure (to make changes of value visible)

5. Rules

   - context (states concerned)
   - temporal-type (time-dependent or time-independent)
   - precondition
   - action (usually contains numerical formulas)

6. Clock

   - time-unit (d) (second, minute, hour, ...)
   - interval (d) (between moves of the clock)
   - current-time (d)

The behavior of the components is described by the rules of their states, which may often be summarized in the form of a small decision table. An important aim of the simulation may be to observe the reaction of the system to abnormal states of the components. For this purpose it is advisable to use random perturbations (with return to the normal state after a certain time to simulate a repair) in addition to specific changes of state by the user.

In order to make the simulation model more flexible, important characteristic values, such as the states of components or constants in numerical formulas, should be represented as variables which may be adjusted by the user. In the case of a large model, the clarity of the representation can be considerably improved if parts can be treated as a subunit. This does not change the simulation, which always takes place on the concrete level, i.e., in the expanded state of the model.

## 31.2 Knowledge Manipulation

The main advantage of numerical simulation in the form described here is that it gives unambiguous results, since the parameters can be assigned unequivocal values by means of the formulas. A simulation begins with the assignment of the simulation variables, the specification of the clock interval and the input of the values of the input parameters. Then the directly effective relationships are calculated first with the time-independent rules, and for this current state the relevant parameters are displayed to the user. Then the clock is advanced, and the relevant time-dependent rules are activated. The new parameter values which result are treated in the same way as the input parameters, and this sequence is iterated until no more time-dependent rules are triggered or the user interrupts the simulation. We thus have the following algorithm:
Condition: adjusted simulation model.
Input: initial values of the parameters.
Output: display of the change of values of the parameters with time, statistics.

1. Assignment of the initial values of the parameters.

2. Calculation of all dependent parameters with time-independent rules.

3. During and after step 2 the "activity procedures" cause their parameter values to be displayed on the screen.

4. *While* the user has not interrupted and there are still time-dependent rules which can be triggered, *repeat*

4.1 Advance the clock by one interval.

4.2 Apply the time-dependent rules with the new time.

4.3 For the new parameter values the time-independent rules are activated as in step 2 to calculate further parameter values.

4.4 During and after steps 4.2 and 4.3 the activity procedures cause their parameter values to be displayed on the screen.

5. Output of the statistics of the parameter values as in the lower part of Fig. 31.1.

The rules which are applicable at each stage are determined from the changed parameters through their attribute "meaning". The time-independent rules can be evaluated immediately, while the time-dependent rules and the components to which they refer are first recorded in a global variable and activated only after the clock has been advanced. If the value of some parameter is determined through several different rules and it is desired to display the parameter only after all the rules have fired, an additional global variable is needed in which all changed parameters are recorded, and their activity procedures are not activated until all the time-dependent rules have been evaluated. In this case the steps 3 and 4.4 of the above algorithm cannot be carried out until the preceding step has been completed.

The attribute "utilization" of the components serves as a basis for calculating the statistics, which form the actual output in addition to the screen animation.

## 31.3 Knowledge Acquisition

The knowledge acquisition begins with the drawing of a network, in which graphical symbols (icons) are defined for the different types of components and materials, arranged on the screen and joined to one another with arrows. Sub-models (e.g., the operation of a depot in a factory) are set up first and provided as icons for building larger models. When components or materials are clicked with the mouse, an object-specific form is displayed in which their local attributes can be entered. The definition of the rules can be simplified by means of a local decision table for each component. A formula editor should be provided for entering numerical formulas for the rule actions.

## 31.4 State of the Art

Numerical simulation is one of the standard techniques of computer science. A thorough review with example systems and references to further literature can be found in [Round 89]. Two examples of graphical, commercial development tools with which models for numerical multiple-phase simulation can be constructed, are SIMKIT, which is based on KEE, and SIMPLE [Becker 90]. The

latter tool, which was specially designed for the simulation of manufacturing systems, is an example of a problem-specific expert system tool with which experts can construct simulation models themselves. Unlike the knowledge representation described above, the component and material types are largely pre-defined in SIMPLE (classified as movable or fixed, active or passive and material or information flow). Their attributes are entered in object-specific forms, and the rules are defined in simple decision tables. As far as possible, the behavior of components in the event of a congestion due to a defective component is pre-defined.

# 32. Qualitative Multiple-Phase Simulation

Qualitative multiple-phase simulation is a simplified form of numerical multiple-phase simulation. Unlike the latter it requires no global clock and cannot supply exact utilization statistics, but gives a state transition diagram as its result. A state is characterized by a set of simultaneous parameter values. The value range of the parameters is qualitative and is chosen so that each value has a special significance for the simulation, which is why they are often called "landmark values". Any change of a value leads to a new state. Since value changes are no longer given by the advancing of a clock and numerical formulas, another concept must be introduced. It is based on the fact that in addition to the qualitative value of each parameter the trend, with the value range "increasing", "constant" or "decreasing", is also represented. A change of state occurs by replacing the value of a parameter with the trend "increasing" or "decreasing" by the next highest or lowest value, respectively. However, since there is no clock the time required for a change of state cannot be determined. On the other hand, state transition diagrams are relatively compact, and cyclic sequences of states can be recognized easily, which would be much more difficult with numerical multiple-phase simulation. Figure 32.1 shows the result of a qualitative multiple-phase simulation for a thermostat-controlled central heating system with a cyclic sequence of states (episodes 1 to 4).

In a numerical multiple-phase simulation of the same system the sequence of states would be determined by the intervals of the global clock, e.g., a new state every 10 minutes, the inside temperature would not be measured in relation to the landmark values (upper and lower thermostat threshold values, outside temperature) but absolutely (in degrees Celsius), and on account of the large amount of data a second program would be needed to interpret the results. The aim of qualitative multiple-phase simulation is to avoid the complexity of the numerical model and supply directly an interpretation.

The central problem of qualitative simulation is the uncertainty which arises when different effects with opposite trends overlap, e.g., in Fig. 32.1 the influences of the heating and the outside temperature on the inside temperature; without numerical values it is not possible to decide whether the inside temperature has reached the thermostat switch-off temperature (e.g., transition from episode 4 to episode 1) or not (transition to episode 5). To remove such uncertainties additional knowledge is required, e.g., special evaluation of numerical knowledge or data concerning the orders of magnitude of various effects ("order of mag-

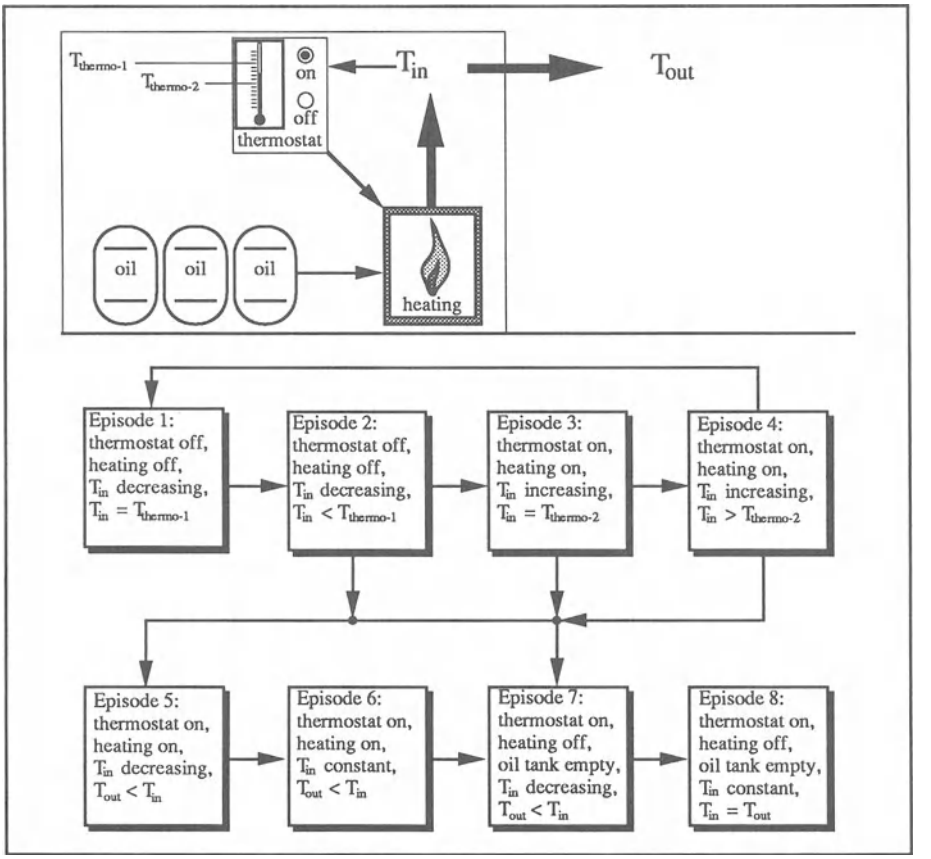



**Figure 32.1** Model of a thermostat-controlled central heating system and the result of a qualitative multiple-phase simulation. The boxes represent states with the most important parameter values, and the arrows indicate possible transitions between the states. The landmark values for the inside temperature ($T_{in}$) are the upper thermostat threshold $T_{thermo-1}$, the lower thermostat threshold $T_{thermo-2}$ and the outside temperature $T_{out}$, which satisfy the strict relationship $T_{thermo-1} > T_{thermo-2} > T_{out}$.

nitude reasoning"). In Fig. 32.1, for example, we can state that under "normal circumstances" the influence of the heating is greater than that of the outside temperature. The basic structure of the knowledge representation and manipulation is described below.

## 32.1 Knowledge Representation

The knowledge representation is similar to that of numerical multiple-phase simulation. The differences are the lack of a clock, the additional parameter attribute "trend", which together with domain-unspecific state transition rules

makes domain-specific, time-dependent rules unnecessary, and the replacement of numerical value ranges and formulas with their qualitative equivalents. Further, the rules can be replaced by constraints if it is required that the direction of derivation should not be restricted. If the orders of magnitude of influences are to be taken into account, the parameters can be given an attribute which indicates the relative strengths of the different factors influencing the parameter. We thus obtain the following structure:[1]

1. Components
    - name
    - type (of the component)
    - input materials
    - output materials
    - states (list of the possible states)
    - current-state (d)
    - effect (list of relationships)
2. Materials
    - name
    - type
    - parameters (for material attributes; if there is only one attribute per material, the parameter attributes may be stored directly with the material)
3. Parameters
    - name
    - material (back-pointer to material concerned)
    - type
    - value-range (defined qualitatively by landmarks)
    - relationships (s) (constraints in which the parameter value is involved)
    - influences (s) (constraints for calculating the trend of a parameter)
    - orders-of-magnitude (assignment of orders of magnitude to the influences)
    - value (d)
    - trend (d) (increasing, constant or decreasing)

[1]Semi-static attributes, which can be derived from the normal (static) attributes by the knowledge acquisition component, are marked (s), and dynamic attributes (d).

4. Constraints

   - name
   - type (pointer to constraint type, e.g., proportionality, additivity, derivative, etc.)
   - parameters (involved in the constraint)

   Constraint-type

   - name
   - definition (set of rules or tables)

The constraint types are qualitative formulas, e.g., a value is proportional to another value or is given by a qualitative addition or multiplication of two other values (which in the case of the value range {+, 0, -} is equivalent to addition or multiplication of signs) or its trend is proportional to another value. The exact behavioral definition of a constraint type may be given by a set of rules or tables. An example is given in [Kuipers 84, Appendix B].

## 32.2 Knowledge Manipulation

As in numerical multiple-phase simulation the problem-solving method alternates mainly between an intrastate analysis, in which a partial set of parameter values is completed, and an interstate analysis, in which the transition to another state is determined. The calculation of time-dependent formulas in numerical simulation is replaced here by the projection of trends. The main differences are the occurrence of uncertainties in qualitative simulation and the possibility and necessity of a global analysis. Uncertainties occur for example when a parameter is affected by two opposing influences, so that its trend is indeterminate, and may be handled without additional knowledge by branching into all possible succeeding states. The global analysis controls the branching, checks possible unifications of different branches and detects repeated states and equilibrium states. From this we obtain the following algorithm:

Condition: qualitative simulation model.
Input: qualitative initial values of parameters.
Output: state transition diagram.

1. Assignment of the initial values to the parameters.

2. Intrastate analysis. The rules are forward-chained with the given parameter values in order to obtain values and trends for all parameters. In the event of uncertainties all alternatives are investigated. Each different set of parameter values is treated as a different state.

3. *Repeat until* no more active states are found in the global analysis:

   3.1 Interstate analysis. Parameters whose trend is "increasing" or "decreasing" take the next higher or lower qualitative value, respectively. If different, independent parameters change, a new uncertainty occurs, and new branching points are added, which contain information about the order in which their value changes take effect. The number of branching points can be limited by general state transition rules, e.g., by requiring that a change leading away from a landmark should take place before a change towards a landmark, or by using the "orders-of-magnitude" of the parameters to eliminate possible branches.

   3.2 Intrastate analysis. The changed parameters initiate another intrastate analysis, in which the changes are propagated through the simulation model.

   3.3 Global analysis. The newly calculated states are investigated to see if they are active, i.e., have not already been derived in the simulation, in which case they would represent loops, and are not equilibrium states. In addition, if identical states occur in different branches they are henceforth treated as one state.

4. Output of the state transition diagram for all states, where each state is described by its parameter values (see lower part of Fig. 32.1).

## 32.3 Knowledge Acquisition

As in numerical multiple-phase simulation the knowledge acquisition begins with the drawing of a network in which graphical symbols (icons) for the parameters and the different constraint types (which must of course have been previously defined by the user or provided as basic functions by the system) are arranged and connected by lines. When a node is clicked with the mouse an object-specific form is displayed, in which the local attributes can be entered. The determination of the landmark values in the value ranges of the parameters is particularly important. Materials and components serve to structure the network by grouping the parameters. To reduce the amount of branching orders of magnitude should be given for the influences in the event of a parameter being affected by several influences.

## 32.4 State of the Art

Qualitative multiple-phase simulation has been the subject of intensive research activity for about 10 years. An up-to-date review is given in [Iwasaki 89]. Pioneering work is represented by the models of de Kleer [84] Forbus [84] and Kuipers [84], which were given considerable attention in [AI-Journal 84,91] and have often

been compared and combined [Bredeweg 88], [Struß 89a]. Further developments are reported, e.g., in [Voss 86] for including hierarchical modeling and explicitly definable temporal relationships, [Kuipers 87] for treating exploding numbers of branches, [Davis 87] for order of magnitude reasoning and [Sacks 88], who analyzed a number of approximation methods with different complexity and precision. A formal framework for qualitative reasoning can be found in [Struß 89b] and [Sacks 90]. In the latter paper qualitative simulation is regarded as a variant of dynamic system theory.

# Part VII

# Integration of Problem-Solving Methods

# 33. Basic Ideas for the Integration of Problem-Solving Methods

The dilemma of expert system tools is to find the right compromise between time-saving in the development of an expert system and the associated restriction of its applicability (cf. Fig. 2.1). The history of expert systems shows a trend to specialization, which constitutes an essential difference with respect to universal programming environments. On the other hand, with specialized tools it is usually possible to solve only selected parts of the whole problem field in an optimum way, while other problem-solving methods and tools are required for other parts. The price for the simplicity resulting from the use of specialized tools is usually the difficulty of coupling different partial solutions. There is therefore considerable pressure from users of expert system for the development of support for the coupling process.

The minimum requirement, which is now fulfilled by many tools, is an open architecture in which the expert system tool provides interfaces to other programming environments. In addition to this weak form of support, in which the coupling has to be implemented by the user, some tools also provide ready-made couplings between important program packets, e.g., to certain data bases. The most advanced approach is the integration of different programming concepts in a single tool, where two programs are not simply added, but united to a single program. Examples are the hybrid general expert system tools which integrate different basic knowledge representation systems, such as rules and frames, and have meanwhile become more widespread than the simple tools with only one basic formalism.

In problem-specific expert system tools the problem of integration becomes considerably more acute. On the one hand, the need for integration is greater on account of the additional specialization, but on the other hand greater demands are placed on the integration, since the aim is to make the development of expert systems possible for experts without knowledge of programming. Interfaces and the coupling of programming environments still require a great deal of concrete programming work and cannot be more than a makeshift measure for problem-specific tools. An attempt is often made to re-formulate all partial problems in such a way that they can be treated with one tool, but such adjustments of the knowledge representation to fit a shell are of very limited use. Particularly if the expert system is regarded as a knowledge medium and not just a problem solver, all representations which appear unnatural to human beings should be avoided.

The development of hybrid, problem-specific tools is therefore destined to become one of the main objects of expert system research. On account of their greater specialization and the good development support for the expert, the question of integration must be treated in much greater detail than in the case of the hybrid general tools. Below we discuss the integration of those problem-solving methods which are at present best understood, i.e., all methods of diagnostics, the heuristic and case-comparing construction methods and simple construction. We first introduce three complex domains as a motivation, then we present the different levels of integration and give a review of pairs of problem-solving methods which lend themselves to integration.

## 33.1 Domains

The necessity for the integration of problem-solving methods is illustrated briefly below for three classical domains of expert systems. A more detailed description of the integration is given in Chap. 35.

1. Decision-making in medicine. This includes diagnostics (a classification problem with heuristic, case-comparing, statistical and set-covering knowledge), therapy planning, which may be a classification or construction problem, depending upon the complexity, and possibly a simulation to check and compare the influence of the therapy and side effects.

2. Tenders, configuration and maintenance. The sale of a technical system involves the specification and price estimate based on the customer requirements (a construction problem), the complete configuration of the system on the basis of the specification (also a construction problem) and the subsequent maintenance, which in turn consists of monitoring (a classification problem), trouble-shooting (a classification problem) and the planning of repairs (a classification or construction problem according to the complexity). Initially only functional models from the construction are available for trouble-shooting, but with increasing experience knowledge becomes available also for all other classification methods.

3. Work planning, production planning and quality control. In a manufacturing process a working plan must first be set up (a construction problem), then a machine utilization plan for the optimum execution of all production orders according to the working plans must be generated and adapted in the event of break-downs or additional requirements (an assignment problem). If faults are discovered in the quality control (a classification problem), the causes in the production process must be found (as in the maintenance problem described above).

## 33.2 Levels of Integration

An integration of problem-specific expert system tools involves first of all collecting together their four components (problem-solving, knowledge-acquisition, explanation and interview components). The advantage of the intended integrated expert system in the form of better solutions and explanations should more than compensate for the additional work required for constructing the knowledge base and for data acquisition. In order to be able to make use of interactions, a common terminology is required. This is manifested in the object references of the knowledge representation. Finally, the integration may also simplify the programming of the individual tools, since some modules may be multiply used. We thus have the following levels of integration, explained in more detail below:

- Object references in the knowledge representation.
- Combination of the problem-solving methods.
- Reciprocal exchange of knowledge during knowledge acquisition.
- Offer of explanations and tutorial strategies in different types of knowledge.
- Minimization of data acquisition by data exchange.
- Reduction of the programming work for the shell development.

## 33.3 References to Objects in the Knowledge Representation

Integration requires that equivalent objects from knowledge bases designed for different problem-solving methods may be related to one another. Thus a certain object, i.e., a name, may represent a diagnosis in heuristic classification, a skeletal plan in skeletal construction or a component state in qualitative, single-phase simulation. The meaning of the object in the different problem-solving methods is expressed by corresponding attributes or attached procedures. The relationship between two objects can be recorded in an assignment table; a better method is to combine the two sets of attributes to a single object.

## 33.4 Problem Solving

In combining the problem-solving methods there is a competitive and a cooperative approach. In the competitive approach a problem may be solved with different problem-solving methods, and criteria are needed for deciding which methods are activated and which solutions are to be preferred. This is the situation above all for different problem-solving methods in the same main problem class (classification, construction or simulation). In the cooperative approach

the problem is resolved into a number of partial problems for which different problem-solving methods may be applicable. This approach may be used for different problem-solving phases, either within a main problem class (e.g., in classification the hypothesis generation may be heuristic and the hypothesis testing model-based), but especially between the main classes (e.g., fault-finding with classification methods, repair planning with construction methods, plan testing with simulation methods). The results of one phase are usually the starting data for the next one.

## 33.5 Knowledge Acquisition

Since the individual knowledge bases are added in integrated expert systems, it is important to be able to handle redundant knowledge. With competing problem-solving methods redundances are unavoidable, but even with cooperative methods similar knowledge may be used, e.g., the result of a construction may be a functional model which contains the basic knowledge for both simulation and model-based diagnostics. Unless special measures are taken, redundances mean extra work for the expert if the same knowledge has to be entered more than once in different syntactical forms. In addition, inconsistencies may arise between the different types of knowledge, so that different problem-solving methods lead to different results. There are three ways of dealing with redundancies:

1. The shell is able to transform knowledge automatically from one formalism to another. This is another way of describing learning. The quality of the acquired knowledge usually improves with the amount of additional knowledge available to the learning system. When learning rules and rule evaluations from case knowledge, for example, causal knowledge is helpful in distinguishing between important and unimportant features and in correctly interpreting correlations. In our opinion successful learning would require so much additional knowledge, including general knowledge, that its acquisition and management is unrealistic at present.[1] Since the lack or insufficiency of background and general knowledge is very detrimental to the quality of the learning process, and especially because there is no guard against serious errors, the knowledge acquired in this way must be checked and if necessary corrected by an expert.

2. The expert enters the knowledge redundantly in different formalisms, and the shell checks its consistency. There are several techniques for consistency checking. The simplest way is to solve exemplary cases with the different problem-solving methods, compare the results and report an inconsistency if they differ. A more complicated method is to compare the knowledge in the different formalisms directly with compatibility rules. However, the

[1] An attempt to make general knowledge available in a knowledge base is being undertaken in the CYC project of the MCC, which has been running since 1984, so far without convincing success.

experience with MORE (see Sect. 17.4) has shown that even within one type of knowledge many small inaccuracies may be found in the knowledge base, but obvious inconsistencies are very rare on account of the use of vague knowledge.

3. The middle course between the above alternatives is to have the expert constructs the knowledge bases in the different formalisms himself, but provide him with support in the form of transformation schemes which help in adopting as far as possible knowledge which has already been formulated in one form when constructing a knowledge base in another formalism, and in keeping it consistent in the event of subsequent changes. This means that the expert keeps control over the knowledge base, but needs to enter hardly any redundant knowledge. The development of the transformation schemes requires an exact analysis of the relationships between the knowledge types on which the problem-solving methods are based and forms one of the central parts of the integration method.

Another problem with integrated knowledge acquisition components is their complexity, which makes it difficult for the experts to learn to use them. This is a serious problem, because the main motivation for the development of problem-specific tools is the ease of operation by the expert. In simple cases the integrated shell should therefore present itself to the expert like a corresponding simple shell. This can be achieved by configuring the shell in such a way that all options which are not needed are hidden. The configuration task is then comparable to the task of selecting a suitable simple shell. One advantage of using a configured, integrated shell is that errors in the analysis of the domain are easier to correct by re-configuration than by replacing the unsuitable single shell, so that the rapid development of a prototype, which is important for expert systems, is better supported.

## 33.6 Explainability

The basis of an integrated explanation component is the explainability of each of the individual methods. In addition, the explanation component should be able to indicate when the result of one problem-solving method is adopted as the starting point of another in the cooperative technique. With competitive problem-solving methods the meta-knowledge about the evaluation differences in the results and about the relationships between the knowledge types should also be explainable.

## 33.7 Data Collection

As in the knowledge acquisition, the input of redundant data should also be avoided when collecting data about the problem features. In the case of competitive problem-solving methods, which solve the same problem with the same

problem data, these data only need to be entered once, so a common interview component is sufficient. The data input can also be considerably simplified in the case of cooperating problem-solving methods when the results of one method are used as input for the next one. At all stages of the problem-solving process, and particularly between two phases, the user should have the possibility of entering or overwriting intermediate results.

## 33.8 Programming

Finally, simplification through multiple use of modules is also possible in the programming of the integrated expert system tool. At the level of the basic knowledge representations this potential for saving has already been partially realized by the development of general expert system tools, which can be used for the development of different, problem-specific tools. Graphical tools which provide the graphical primitives (tables, hierarchies etc., see Sect. 12.5) required for knowledge acquisition are also useful. The same applies to tools for designing dialogs and providing graphic primitives and connections to other programs (e.g., to data bases and to process computers for accepting data). In addition, one can take advantage of specific similarities in the problem-solving methods. In skeletal planning, for example, the refinement of planning steps may be regarded as heuristic classification, and in classification with functional models a simulation method may be used for testing the hypotheses, so that the corresponding program units may be used several times.

## 33.9 Survey of the Integration of Problem-Solving Methods

The task of the integration outlined above is to investigate the relationships between diagnostic methods (heuristic, statistical, case-comparing, set-covering and functional), construction methods (skeletal construction, propose and revise, least commitment and case-comparing construction) and single-phase simulation (for testing selected or constructed solutions) with regard to the level of integration (object references, problem solutions, explanations, knowledge acquisition, data collection and programming techniques). With 10 methods and 6 levels a complete investigation would involve $45 \times 6 = 270$ unidirectional or 540 bidirectional interactions. We will therefore concentrate on the most important ones. With competitive problem-solving methods the interactions are more intensive than in the case of cooperating methods. Within a main problem class the methods tend to be competitive, while those from different problem classes rather cooperate. We will therefore first look at the interactions within a main problem class and then at those between problem classes. Our main interest will be focussed on the transformation of knowledge for knowledge acquisition, the extent to which the

methods complement each other in the solution and the multiple use of program modules.

In diagnostics, especially in the case of new technical systems, knowledge is transformed, as experience is gained, from functional models via fault models into heuristic knowledge, and via case data banks into statistical and case-comparing knowledge.

In construction the application spectra of hierarchical skeletal construction and the propose-and-revise strategy are complementary. Case-comparing construction requires a modification of the solution when the comparative case has been selected; a suitable technique for this modification is the propose-and-revise strategy, in which the chosen case is regarded as the proposal to be revised.

We are interested in simulation only as a means of testing diagnoses or constructions. The integration of simulation is therefore of secondary importance, since with appropriate simplifications a single-phase simulation is sufficient in all cases.

There is a strong interaction between diagnostics and simulation in the case of functional models, since almost the same model may be used for both methods.

The main interaction between diagnostics and construction is the exchange of program modules. In case-comparing diagnostics and case-comparing construction, the comparison process is essentially the same, the main difference being that in the latter method the selected case has to be modified. There are also many similarities between heuristic classification with the establish-refine strategy and skeletal construction. The main difference here is that in skeletal construction we have AND nodes in the hierarchy in addition to the diagnosis-like OR nodes. There are also relationships between heuristic classification and the propose-and-revise strategy if the concept of exception rules in the former method is extended to include proposals for improvement.

Heuristic construction and simulation can be related by transfer of knowledge, preferably from the predominantly component-orientated, heuristic constructions to the more functionally orientated simulation models.

A summary of the integration possibilities is shown in Fig. 33.1.

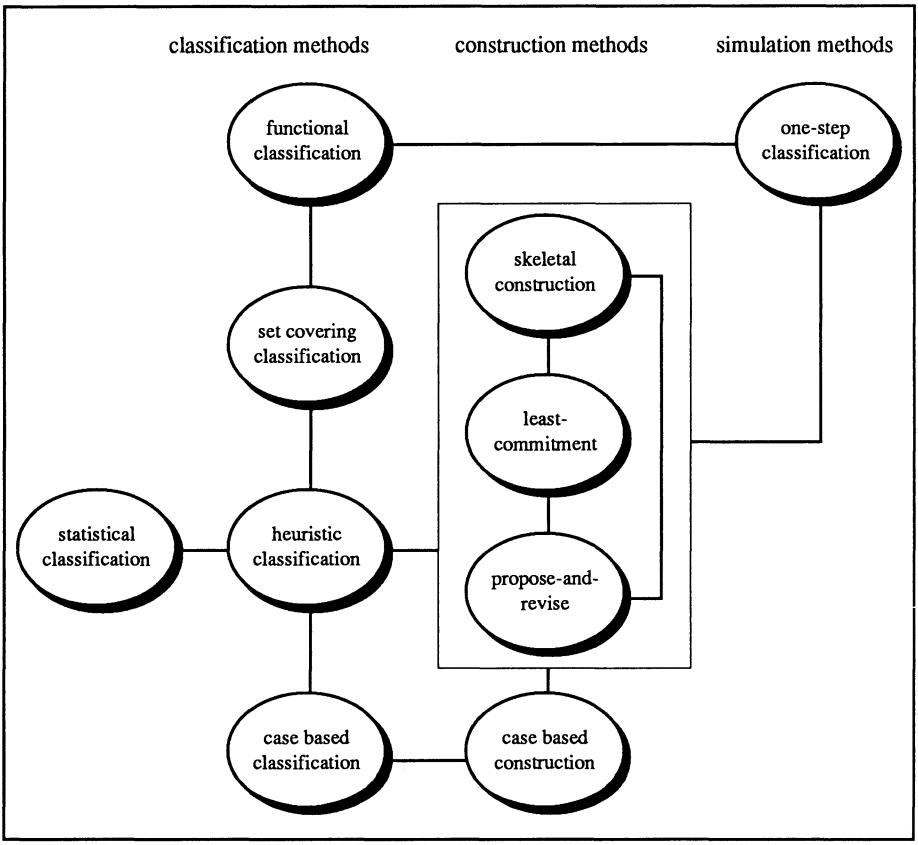



**Figure 33.1** Survey of the integration of problem-solving methods

# 34. Integration of Classification Methods

An integration of classification methods promises good synergetic effects, since classification is the best-understood problem-solving method, and the knowledge forms of the different methods (simple, heuristic, statistical, case-comparing, set-covering and functional classification, see Chaps. 14–20), although they overlap somewhat, are not mutually convertible:

- Statistical and heuristic knowledge resemble one another in the representation of probabilistic assessments. They differ, however, firstly in the variety of knowledge representation they provide, since heuristic classification systems offer mechanisms such as multiple hierarchies, rules with symptom combinations and exceptions, etc., which have no equivalents in statistical knowledge representation systems, and secondly in the precision of the probabilistic assessments, which are calculated from data bases or estimated by experts, respectively. The importance of the difference between calculated and estimated symptom/diagnosis probabilities was shown by experiments with a successful program by de Dombal, based on Bayes' Theorem: the success rate of the program dropped drastically when the statistical probabilities were replaced by experts' estimates [de Dombal 72].

- Heuristic and set-covering classification are both based on empirical knowledge about faulty behavior of the system being investigated. They differ, however, in the direction of derivation (symptom $\rightarrow$ diagnosis and diagnosis $\rightarrow$ symptom, respectively) and also in the fact that the experts' estimates play an important role in heuristic systems, while set-covering systems rather check whether diagnoses are consistent with the observed symptoms.

- Set-covering and functional classification resemble one another in the way they match observed and predicted symptoms. They differ, however, in the conceptual function of symptoms and diagnoses. In set-covering classification the diagnoses and symptoms, represented as states, are fixed concepts in their own right, as in all other types of knowledge, whereas in functional classification they are derived from the functional model: diagnoses are components with an altered input/output behavior, and symptoms are abnormal parameter values of the materials.

- Case-comparing classification differs basically from all other knowledge types, since it is based not so much on an abstraction of case knowledge but rather on a direct comparison of a new case with those stored in a data base.

- Simple classification can be regarded as a compiled form of the other problem-solving methods, in which the uncertainties or causal relationships are no longer explicitly represented.

## 34.1 Strengths and Weaknesses of the Problem-Solving Methods

Classification problems can in principle be solved with any one of the problem-solving methods alone. However, the prerequisites of the methods are different: simple and heuristic methods are based on the empirical knowledge of experts, statistical and case-comparing methods need large case data bases, and the two model-based methods require basic causal knowledge from the literature. Additional empirical knowledge is helpful, especially in set-covering and case-comparing classification.

Empirical knowledge promotes the efficient generation of hypotheses, but it is difficult represent objectively. On the other hand, statistical knowledge and functional models are well suited to an objective representation. Set-covering and case-comparing classification have an intermediate position, since they depend upon empirical knowledge for their optimum function. A big problem with empirical knowledge is the fact that experts usually have difficulty in estimating the uncertainties. For this reason the evidence calculi used in heuristic classification are generally coarser than those used in the statistical methods, and the other methods dispense as far as possible with probabilities.

The quality of the hypothesis test depends upon the amount of knowledge available and can be improved to almost any degree in all methods, e.g., by expanding the case data base in the statistical or case-comparing methods or by further refinement of the functional or fault models. In simple, and to a lesser extent also in heuristic classification, a high quality can be achieved relatively quickly in this way, but further refinement becomes disproportionately difficult because of the dependence of expert experience.

A challenge for classification methods is the ability to arrive at multiple diagnoses. Since there are $2^n$ possible combinations of $n$ individual diagnoses and each combination is therefore rare, the available case data is usually insufficient for statistical or case-comparing classification. To a lesser extent this is also true for empirical knowledge. Thus multiple diagnoses usually have to be assembled from possible overlapping symptoms of the individual diagnoses, which is best done with causal models.

Causal models are also the usual form for explaining derived diagnoses. In the case of functional models one can usually assume the normal model, which

the user knows at least roughly, so that the explainability is even better than for set-covering classification with fault models. It must be remembered, however, that the quality of explanations, unlike solutions, cannot be measured with an objective scale, so that the acceptance varies individually. For example, a user may be satisfied with the explanation that his problem is very similar to another problem, the solution of which can be adopted. Since explanation usually involves referring something new back to something already known, this type of explanation depends upon the previous knowledge of the user.

The properties of the different problem-solving methods are summarized in Fig. 34.1.

| Type of knowledge | Knowledge source | Objectification | Relevance of Probabilities | Hypothesis generation | Hypothesis testing | Multiple diagnoses | Explainability |
|---|---|---|---|---|---|---|---|
| Certain | Experience | − | − | + | − | − | − |
| Heuristic | Experience | − | + | + + | 0 | 0 | 0 |
| Statistical | Cases | + | + + | − | + | − | − |
| Case based | Cases and experience | 0 | − | − | + | − | 0 |
| Set covering | Experience and lit. | 0 | 0 | − | + | + | + |
| Functional | Literature | + | − | − − | + | + | ++ |

**Figure 34.1** Properties of the knowledge types belonging to the different problem-solving methods. The signs − −, −, 0, +, + + indicate the quality or degree of the property on a scale from poor/low (− −) to good/high (+ +).

## 34.2 Knowledge Representation

A central requirement for an integrated shell is the avoidance of multiple representation of objects. The most important object types of the different knowledge types are:

- Heuristic classification: observations (symptoms), data abstractions (symptom abstractions), solution classes (coarse diagnoses), solutions (fine diagnoses) and tests for dialog control.
- Set-covering classification with fault models: states and parameters.
- Functional classification: components, states, material types, materials and parameters.
- Case-comparing classification: observations and solutions.
- Statistical classification: observations and solutions.
- Simple classification: observations, intermediate states and solutions.

There is a direct correspondence between the object types of heuristic, case-comparing, statistical and simple classification, if the intermediate states of simple classification are mapped onto data abstractions or solution classes. The observations of case-comparing and statistical classification should be as expressive as possible and therefore frequently correspond to data abstractions of heuristic classification.

In set-covering classification the distinction between observations and solutions is of secondary importance; instead, there are states which may cause other states. States without causes correspond to solutions, and states without effects to observations or data abstractions; the remaining states may be both. The mapping of the parameters of a state, representing, e.g., its intensity or duration, presents more problems. Although solution parameters are normally not needed in heuristic classification, they could be mapped onto data abstractions assigned particularly to the solutions, e.g., by providing each solution with an observation class containing only its parameters as data abstractions.

Functional models differ from all other knowledge types, because their main representation elements are components and materials, which can be identified as problem observations or solutions only indirectly through their deviations from the normal state. Thus a component in state $X$ may be a solution, and a parameter of a material having a value within a certain interval $Y$ may be an observation, a data abstraction or an intermediate solution. If functional models are compared with the fault models of set-covering classification, we find the following correspondences: fault model states without causes = states; fault model states with causes = existence of abnormal values of the material parameters; fault model parameters (intensity) = parameter values. This mapping is admittedly only partial, since for economy reasons only the most important states and abnormal parameter values appear in the fault model. Normal values and the unimportant abnormal values of the parameters have no correspondences. The basic reason is that functional models map a system relatively completely, while all other knowledge types only take into account those elements which are relevant to the diagnostics. From these considerations we arrive at the following object types for an integrated shell:

1. *Observations and data abstractions.* Observations correspond to the questions of heuristic classification and serve for primary data collection. Data abstractions are derived from the observations with simple rules. The distinction between observations and data abstractions allows both a natural dialog control with observations and the definition of a problem-orientated terminology with data abstractions for the different knowledge types. Their attributes are the union of all the attributes of the observations, data abstractions and parameters of all knowledge types; the name and the value range occur only once, however.

2. *Observation classes.* They group together observations and/or data abstractions and serve for the simplification of the dialog control or for assigning solutions to groups of data abstractions. They include the question classes

of heuristic classification, the materials of the functional models and the set of the parameters of a state in fault models and combine their attributes.

3. *Intermediate and final solutions.* Unlike observations and observation classes they have a probability value. On the other hand they lack the possibility of the definition of an individual value range. They include the combination of the attributes of the solutions and states of all the knowledge types. In particular, they can be valued differently by each knowledge type.

4. *Components and material types.* These sets of states or observation classes are required only for the functional models and are therefore given their attribute structure.

5. *Rules and constraints.* They express the knowledge of the those object-object relationships which cannot be expressed by direct object references. Rules and constraints are connected to the objects concerned by a complete system of pointers. By this means the inference process, including the withdrawal of conclusions, can be implemented very efficiently by message passing between the objects. This requires that all the objects appearing in the rules must be known. Each knowledge type has its own rule syntax.

## 34.3 Knowledge Manipulation

At the problem-solving level great flexibility is necessary, since it cannot be assumed that all knowledge types are provided, which is possible only in few domains. If only one type of knowledge is present, the whole problem-solving process must rely on it, otherwise synergetic effects should be exploited.

Since the individual problem-solving methods are self-contained, one can choose a distributed architecture in which each problem-solving method is represented by its own "agent", i.e., by an independent program section. A blackboard, i.e., a data structure which is available to all agents, serves for communication. The blackboard carries problem observations (symptoms), local abstractions, hypotheses and intermediate and final solutions (see Fig. 34.2).

For the sake of simplicity we will first assume that all problem features (observations) are known from the beginning. They form the initial contents of the blackboard. Since the pre-processing of observations for the derivation of local abstractions is useful for all problem-solving methods, this phase should be carried out centrally as soon as the observations have been recorded on the blackboard. Observations and data abstractions can be read by all problem-solving agents. After processing the data, an agent can write suggested hypotheses or intermediate or final results on the blackboard, and these can then be read and processed by other agents. In this process there are two typical interactions:

- Certain agents, typically the heuristic agent, write suggested hypotheses on the blackboard which are tested by other agents. In this way a focusing of the problem-solving process is achieved.

| Observations (objects with values) | Data abstractions with values |
|---|---|
| $O1 = V1$ | $D1 = V1$ |
| $O2 = V2$ | $D1 = V2$ |
| ⋮ | ⋮ |
| $On = Vn$ | $Dn = Vn$ |

| Solutions | Heuristic | Statistical | Case-comparing | Set-covering | Functional |
|---|---|---|---|---|---|
| Solution 1 | probable (0.6) | very probable (0.9) | — | plausible (0.8) | — |
| Solution 2 | suggested | — | — | implausible (0.2) | — |
| Solution 3 | suggested | neutral (0.2) | — | possible (0.5) | — |
| Solution 4 | certain (1.0) | certain (0.95) | — | plausible (1.0) | — |
| ⋮ | | | | | |

**Fig. 34.2** Structure of the blackboard

- The agents write different intermediate and final results on the blackboard; the decision between them is taken on a global level.

Focusing the problem-solving process is especially important for the causal problem-solving methods, since they do not have an efficient hypothesis generation. Since statistical knowledge is best suited for deciding between a few alternatives (differential diagnostics), it is most effective when the number of possible alternative solutions has already been greatly reduced. In case-based knowledge also, suggested hypotheses can control the pre-selection of comparative cases from a case data base. If no heuristic knowledge is available for the rapid generation of hypotheses, or if all hypotheses turn out to be improbable, an attempt can be made to generate hypotheses with the other methods, despite their inefficiency. In case-based knowledge this depends upon whether the concrete problem involves unusual symptoms which can be used for the pre-selection. In causal models the generation is more efficient the simpler the model is, so the model should be abstracted, especially if heuristic knowledge is lacking. In a certain sense, knowledge transfer from functional models with fault states to fault models (see Sect. 34.4) constitutes such an abstraction. The overall control of the integrated tool[1] should be adjustable on a global level and be able to take into consideration not only various compromises between available computer time and thorough hypothesis

[1]With a multiple-processor architecture the agents can be attached to procedures of their own, so that the different types of knowledge can be processed in parallel. The control problem then reverts to the individual processors, when they are called upon to interrupt their investigations in order to check current hypotheses on the blackboard.

generation but also the existence and the quality of the agents. An example of possible default settings for all knowledge types is shown in Fig. 34.3.

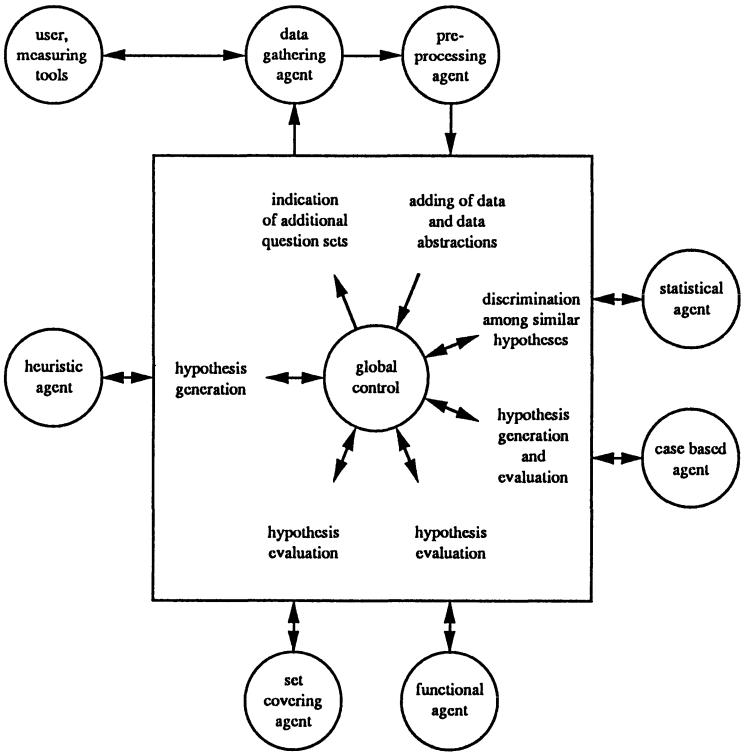



**Figure 34.3** Possible global control of the knowledge types. The observations are recorded together with the data abstractions derived from them on the blackboard and evaluated by the heuristic agent for generating hypotheses. Further hypotheses are suggested by the case-based agent, which also seeks comparative cases for the heuristic hypotheses. All hypotheses are checked by the set-covering agent and by the functional agent. Similarly rated hypotheses are also compared statistically. Depending upon the current state of the suggested hypotheses, further sets of questions to be put to the user or to measuring instruments are indicated.

To decide between competing hypotheses generated by different agents meta-knowledge is needed, i.e., knowledge about the reliability of the hypotheses. A part of this meta-knowledge is the internal estimate of the certainty of the results of a particular agent, e.g., a hypothesis may be probably, highly probable or certain. In addition, the relative reliabilities of the different agents, which may differ considerably according to the quality of the knowledge bases, must be compared. It may be necessary to differentiate between the qualities of different parts of the same knowledge base. If different agents come to the same result, the estimated likelihood of that hypothesis can be increased, and if the results are different that with the best estimate can be adopted. The amplifying effect of two agents

coming to the same result is stronger, the greater the mutual independence of the two knowledge bases.

The combination of the different partial valuations requires a reckoning scheme for dealing with uncertain knowledge (see Chap. 8). For this purpose the MYCIN scheme may be used, for example. The rule precondition then corresponds to the primary hypothesis valuation of an agent, and the rule evaluation to the reliability of the hypothesis itself or the relevant part of the knowledge base. In the MYCIN scheme the precondition is multiplied by the evaluation of the rule, and different rule probabilities $P$ for the same object are connected by the formula $P = P_{\text{old}} + (1 - P_{\text{old}})P_{\text{new}}$. Since the rule probabilities must be independent of one another, it is necessary to distinguish between dependent and independent agents; case-comparing and statistical knowledge, for example, are interdependent because they use the same case data base, and fault models and functional models because they use the same causal knowledge. In the case of interdependent agents it is reasonable to take the maximum of the individual probabilities, whereas the MYCIN formula can be used to determine the overall probability for the remaining, relatively independent groups (causal, heuristic and case-based). The only consequence for the knowledge representation is an additional valuation of each agent or parts of each knowledge base.

The underlying assumption of the blackboard model that the agents are mutually independent in their internal problem-solving processes is not entirely valid in one area: the dialog control. If each agent were to put questions directly to the user, the common data structure of the blackboard would ensure that no question would be put twice, but the ensuing series of questions alternating between quite different aspects of the problem could be confusing. For the coordinated external presentation an agenda is therefore required, in which each request for data is recorded with its degree of urgency and the relevant static and case-specific costs are also evaluated. The data request with the best current profit/costs relationship is selected and executed.

A need for synchronization comes from the fact that one should really wait until each agent has finished its calculations, since it is only then that its data requirements are known. This can be avoided if the dialog control is given to one agent, e.g., the heuristic agent, which considers its own needs and in addition the current situation of the blackboard, so that hypotheses which other agents wish to check are also taken into account. These problems are further reduced if the data requests are standardized and depend upon simple criteria (e.g., directly upon the problem data), which is indeed the case in many domains, e.g., "if the engine is running irregularly, do a power test" or "if the patient has chest pains, do an electrocardiogram and a chest X-ray."

Another simplification comes from the fact that the smallest dialog units are not individual questions but question classes. They are indexed globally and possess local knowledge about which questions should be put to the user and in what order.

## 34.4 Knowledge Acquisition

An integrated classification shell makes it possible for the user to enter knowledge directly with no complicated transformations, whatever the knowledge type. When setting up diagnostics for a new technical system such as a production plant, for example, the user could first construct the functional model, then add to it typical faulty states of the components, simplify it to a fault model, add heuristic knowledge for rapid hypothesis generation and finally proceed to case-comparing and statistical classification when a sufficient case data base has been acquired. The common core of all knowledge types is made up by the concepts, i.e., symptoms and diagnoses. The knowledge base is extended mainly by adding relational knowledge, which is represented differently in the different knowledge types.

In order that the expert may take advantage of the knowledge already acquired in all knowledge types when formalizing knowledge for one type, transformation schemes between the knowledge types must be available. In accordance with the classification section of Fig. 33.1, at least the following transformation schemes should be provided:

a) functional models → fault models,

b) fault models → heuristic knowledge,

c) heuristic knowledge → statistical knowledge,

d) heuristic knowledge → case-based knowledge,

e) case-based knowledge → statistical knowledge,

f) statistical knowledge → heuristic knowledge,

g) heuristic knowledge → fault models,

h) fault models → functional models.

For each of the above transformation schemes we indicate below which features can be taken over from the starting knowledge representation and what knowledge has to be added. Each transformation should be supported by the basic knowledge acquisition components of the individual knowledge types in such a way that the relevant knowledge section of the starting knowledge type (e.g., an object, a group of objects or a relationship) can be selected in the graphical representation, and the required additional knowledge entered by means of a transformation-specific form. In suitable cases default values for the additional knowledge can be derived and offered to the user.

a) In the transition from functional models to fault models the model is simplified. In particular, all knowledge of the functional model which is not required for describing the faults should be eliminated. For this purpose the distinction between "components" (with normal and faulty states) and "materials" is removed

and mapped on an object type "state". The states of the fault models include only the faulty states of the components, i.e., the normal states are dropped, and from the materials only those attributes in which the fault is manifested are included. For example, the material "air" in Sect. 18.5 has the attributes "pressure", "flow rate", "degree of contamination", "oxygen content" and "temperature", of which only the degree of contamination is abnormal in the case of a torn air filter. The fault model therefore has only the new state "air contaminated", which is derived directly from the state "air filter torn". A further simplification is based on the observation that in the functional model abnormal material attributes often pass through many components unchanged before they cause the actual symptoms. Such intermediate steps can be left out of the fault model.

b) In the transition from fault models to heuristic knowledge the model can be further simplified by eliminating most of the gravities of the states and some of the intermediate states. Apart from this all objects can be taken over. The most important change is that the inference direction of the rules must be reversed: instead of cause (diagnosis) → effect (symptom) in the fault models the heuristic rule is symptom → diagnosis. In addition, various new pieces of knowledge must be added: on the one hand empirical knowledge about how strongly the presence or absence of a symptom is evidence for or against a diagnosis, and on the other hand knowledge about the dialog control and symptom abstraction. The dialog control must take into account how the information about the different symptoms can be obtained (e.g., by questions to the user, by technical investigations, etc.), and the symptom abstraction is necessary to bridge the gap between the technical language of the causal models and the primary data obtained from symptom data collection.

c) In the transition from heuristic to statistical knowledge the main change is the replacement of the subjective experience of experts by statistically calculated probabilities. The main problem in this transformation is to ensure that the conditions for the application of Bayes Theorem are fulfilled as far as possible. This requires a careful selection of suitable symptom abstractions (to ensure that the symptoms are independent of one another) and a sufficiently small set of diagnoses (to avoid multiple diagnoses).

d) In the transition from heuristic to case-based knowledge all symptoms and symptom abstractions are taken over, while the heuristic rules are dropped. The only additional knowledge needed is that concerning the significance and the abstraction level of the symptoms. The latter serves also for the selection of the symptoms and symptom abstractions to be used for case comparison, since symptoms may be compared only if they are on the same level of abstraction.

e) The transition from case-based to statistical knowledge corresponds to the statistical evaluation of a case data base and is mentioned here only for the sake of completeness, since it is part of the normal knowledge acquisition process for statistical knowledge.

f) In the transition from case-based to heuristic knowledge the aim is to integrate statistical probabilities into the latter. For this purpose the statistical probabilities are coarsened to heuristic evidence values in a symptom-diagnosis

table and compared with any expert assessments which may be present. The expert decides which of the evidence value derived from the case data base are adopted by the heuristic knowledge.

g) The transition from heuristic knowledge to fault models requires the direction of inference to be reversed, keeping the same objects (see point b). The rule evidence values do not play an important role in the fault models, but new knowledge about the cumulative influence of several causes on one effect must be added. For this purpose the rules must be refined so that they can calculate and propagate the gravities of the states, which may mean that intermediate states not normally needed for heuristic knowledge have to be introduced.

h) The transition from fault models to functional models is the most difficult one of those treated here. The reason is that the objects of the fault models cannot be mapped directly onto objects of the functional models. We therefore assume that the expert first enters the objects of the functional model and their relationships in the normal state independently of the fault model. Then, when the functional model is extended to include the faulty states of the components and their effects, in addition to the normal states, the knowledge of the fault model can be exploited by assigning its starting states to the corresponding components of the functional model as faulty states and finding the corresponding material attributes for the other states.

Further transformation schemes are required if additional knowledge is available. Particularly in the case of functional models, we usually have not just one model, but many, based on different assumptions and also on different knowledge representations.

The knowledge transformations allow a limited consistency check of the knowledge bases, whereby one knowledge type, e.g., causal models, effectively plays the role of a specification. This applies in the first place to the conceptual structure, the consistency of which is guaranteed to a large extent by the application of the transformations. However, certain tests of consistency are possible even for the relational knowledge, which is difficult to analyze on account of its vagueness. Although concrete rule evaluations from different knowledge types can hardly be compared, the existence and the sign of a relationship (positive or negative assessment) can be checked in the following situations:

- For every implicit, simulated fault path in a functional model there must be a corresponding explicit path in the fault model, and vice versa.
- For every diagnosis-symptom relationship in a fault model there must be a corresponding heuristic symptom-diagnosis relationship, and vice versa.
- The evaluations of a statistical and a heuristic symptom-diagnosis relationship can be compared (however, in a heuristic system the statistical evaluation may be distributed over contexts, symptom combinations and exceptions).
- The significance of symptoms in fault models and case comparison may be compared.

On the basis of such rough comparisons it is also possible to localize relatively precisely those parts of the knowledge bases in all knowledge types which must be checked in the event of a change of knowledge (e.g., a technical innovation in the domain), as long as the knowledge change can be described easily in one knowledge type (typically in a functional model).

## 34.5 Explainability

The integration of problem-solving methods places high demands on the explanation component. If only one problem-solving method is used, the explainability is generally determined directly by the knowledge representation[2], which is why they have hardly been mentioned in the discussion of the individual methods.[3] With multiple knowledge types there are several different explanations of the same facts, however, so that transitions and preferences become important, and these in turn depend upon the previous knowledge and interests of the user.[4] The concept of explainability includes various modes in which the user may exploit the system for his own further qualification, including tutorial aspects:

- *Manual mode.* This is to give the user the opportunity to examine the static knowledge of the system. Unlike a "real" manual, the user can read an "electronic" manual in an associative way: with a simple mouse click he can cause information to be displayed concerning any word or picture in which he is interested. This is the idea of hypertext systems.

- *Justification mode.* The user can call for justifications of the solutions of concrete cases derived by the system. In expert systems which are used primarily for problem solving the justification mode is the main constituent of the explanation component.

- *Critique mode.* In this mode the system leaves the user to solve a concrete problem and comments on or criticizes his or her decisions, i.e., suggestions for further investigations, hypotheses and final diagnosis.

[2]Strictly speaking this is true only for "direct explanations", which are derived from the problem solution. However, since "indirect explanations", e.g., comments, require a great deal of maintenance, they do not represent a useful alternative for large expert systems.

[3]This is not meant to detract from the importance of explanations for expert systems, but to emphasize the fact that good explanations are not possible with an unstructured knowledge base. Particularly if expert systems are regarded primarily as knowledge media and only in the second place as problem solvers, the explainability is of decisive importance, and its implementation may be very complicated. The explanation modes mentioned in this section are correspondingly valid also for the individual problem-solving methods.

[4]The dependence of good explanations upon the user is being investigated in the framework of the research field "user modeling" (see, e.g., [Kobsa 90]), where the system makes a model of the user and acts accordingly. We take the opposite position here, namely that the user should find it as easy as possible to make a model of the system and should have corresponding options to control the system.

- *Teacher mode.* This includes the critique mode. In addition, the user can choose from a number of example problems, classified according to difficulty and field of application, and the system scores his or her solutions.

- *Pupil mode.* Since one often learns most when explaining a field to someone else, the knowledge acquisition can also be regarded as the pupil mode: the user passes knowledge on to the expert system and checks the success by seeing whether the system can solve cases with this knowledge.

- *Experimental mode.* The problem-solving ability of the expert system can also help the users in learning, e.g., if they wish to study the effects of small changes in a case, they can have the expert system show them how these changes affect the solution.

The manual, justification and critique modes are discussed below. The remaining modes are already largely contained in the basic expert system: the teacher mode consists basically of the critique mode with given, classified example problems, the pupil mode uses the knowledge acquisition component, the problem-solving component and the justification mode, and the experimental mode the problem-solving component with revision options and an efficient withdrawal procedure.

### 34.5.1 Manual Mode

The manual mode is based on the hypertext technique, in which certain words or regions in a window with text and/or icons are sensitive, i.e., responsive to mouse clicks, whereupon relevant information is displayed in a new window. Since this process may be repeated as often as desired, and it is easy to go back to previous windows, the users can read the information in the order they choose, instead of in a pre-determined order as in books. Suitable starting windows are coarse representations of the knowledge base, e.g., diagnosis and symptom hierarchies in the case of heuristic, statistical and case-based knowledge, state graphs for fault models and functional models and also conventional texts and pictures. From this starting point the users can read associatively, since for any chosen term they can look directly at all the related meanings.

When several knowledge bases for different problem-solving methods are available, the possible associations for any term become more numerous. As an extension of the simple hypertext system a case distinction is therefore required: when an object is clicked the new window does not appear immediately, but first a pop-up menu from which the user can choose the context in which he or she is interested. For such a pop-up menu the associations must be structured, for which purpose the already existing structures in the knowledge bases should be exploited. A suitable structure is the classification into different knowledge types, conventional texts and pictures being regarded as an additional knowledge type. The result is that the meaning in the knowledge base of any symptom or diagnosis in the manual text is immediately available (and vice versa).

In addition, different meanings of object types within a knowledge type should be represented as different options of the pop-up menu. A basic menu might have the following options for an object, of which the irrelevant ones may be blanked out according to the knowledge type and user wishes:

a) Role in the diagnostic process, i.e., derivation and meaning:

1. heuristic knowledge (i.e., rules),
2. case-comparing knowledge (i.e., list of cases),
3. statistical knowledge (i.e., correlations),
4. fault model (i.e., neighboring states),
5. functional model (i.e., component or material in which the symptom or the diagnosis appears).

b) Other roles in expert systems:

6. associated tests (role in dialog control),
7. associated repair or therapy instructions with prognosis (only for diagnoses),
8. defining knowledge (value range, consistency relationships, etc.).

c) Reference to conventional texts and picture:

9. description in textbook,
10. illustration with pictures,
11. entry in encyclopedia.

### 34.5.2 Justification Mode

The purpose of the justification mode is to explain the solutions of concrete problems. This mode may be regarded as a case-specific restriction of the manual mode, in the sense that not all information about an object should be shown, but only that which is relevant to the present case. This means that the manual mode must be provided with a filter which lets through only the relational knowledge actually used, i.e., only the triggered rules and constraints. An additional option is important to explain why certain solutions were not derived, i.e., which rules of a diagnosis were not triggered, and why not. From the manual mode the justification mode adopts the easy switching between justifications in different knowledge types. In the output window of the justification mode, the problem-solving component should always show the currently considered and established intermediate and final diagnoses and the intended tests, since these are the most important objects for which a justification is required.

An explanation component which offers many different justifications for a solution may easily confuse the user. For certain user types (e.g., non-specialists,

students, specialists, knowledge base developers) simplified versions should therefore be provided, e.g., by limiting the justifications to only one knowledge type. The default knowledge type would be functional models or as a substitute fault models, since causal explanations are generally preferred. The amount of detail in the justification should be adjusted to the previous knowledge of the user by presenting the knowledge in abstracted, i.e., shorter and simpler form as appropriate. It is assumed that the users classify themselves according to type and previous knowledge; with the user modeling techniques it would also be possible to have the system classify the users according to their questions.

### 34.5.3 Critique Mode

In the critique mode the expert system criticizes the solutions derived by the user for given symptoms. If these symptoms are not sufficient for determining the solution, the user must first suggest further investigations, and their profit/costs relationships are also criticized.

It may be assumed that the symptoms are presented in text form, but it is also possible first to show pictures of the symptoms, which the user must recognize and relate to the corresponding terms. In this case the critique consists simply of the statement "right" or "wrong"; a detailed justification is not possible. The use of images (and also the associated sounds) is very useful particularly in the teacher mode.

The advantage of having different types of knowledge is that the users can select that which comes nearest to their own problem-solving method. However, the case-based knowledge type is suitable only if the user and the expert system know the same cases. For the criticism the system simply compares its solution in the chosen knowledge type with that of the user, determines the differences, evaluates their gravity and presents the result to the user.

An important aspect of the critique mode is that the users should be able to enter hypotheses and justifications in a simple way without having practically to build up a small knowledge base, but at the same time the multiple-choice menus should not give them too clear an indication of the correct solution.

In the simplest case the users simply states their hypotheses without justifying them. The hypotheses are chosen from the complete or restricted diagnosis hierarchy. The system compares the hypotheses with its own solutions and evaluates any differences. With causal models the system can also carry out a simulation with the user's hypotheses in order to evaluate their plausibility.

If the critique is to go further than the simple confirmation or rejection of a solution, the users must justify their proposal, so that the system can comment on the justification. In a simple version the users simply assign the symptoms to the diagnoses they suggest; they should divide the symptoms into three groups:

1. symptoms which are present and support the diagnosis,
2. symptoms which are present and have nothing to do with the diagnosis, and
3. symptoms which are not present but are actually expected.

The system can then check the individual assignments and value them. In a more detailed version the user would have to qualify the assignments with their significance, i.e., how strongly the symptoms support or oppose the diagnoses.

The relationships can be described even more precisely if the user does not just assign the symptoms directly to the final diagnoses but also specifies the intermediate diagnostic steps. The intermediate process consists of two parts: the local symptom abstractions and the global intermediate diagnoses. In the case of the symptom abstractions, which are local to a question class, the system can provide a menu with all alternatives in a question class, from which the user selects the correct values according to the raw data.

This step can be carried out for all knowledge types, since they all use the same symptom abstractions. One advantage is that the assignment of symptoms to diagnoses can now take place on the more informative level of the symptom abstraction.

The specification of intermediate diagnoses or states by the user is more complicated. For this purpose the user may construct a justification graph from given parts on the screen. Alternatively the system could display all possibly relevant intermediate and final diagnoses as isolated nodes on the screen, whereupon the user deletes the irrelevant nodes and connects the remaining ones with lines. Different types and weights of the relationships are represented by different types of lines. The result is a case-specific justification graph.

The greater the detail of the justification graph, the more precise is the critique. If in heuristic classification, for example, where the symptoms are assigned directly or via only a few intermediate steps to the diagnoses, the justification mode must first resolve the surplus steps in its own justification which serves for comparison, and this decreases the precision of the heuristic values. In causal classification the corresponding removal of the intermediate states has the effect that the possibilities for criticism are restricted, especially if almost everything depends on everything else in the causal model.

An implicit limitation of the above selection-orientated input is that the users can use only the objects of the knowledge bases with the given terminology for their solutions or justifications. However, since justifications in another terminology or with more detail than the current state of the knowledge base could not be criticized anyway, the selection-orientated input seems appropriate. It has the further advantage that the dialog is faster and the implementation much simpler than for a natural language dialog.

## 34.6 Data Collection

The requirement placed on the data collection by the integration is that the symptoms are registered only once, but all problem-solving methods are accessible. The main mechanisms for achieving this have already been mentioned:

1. Data belonging together are grouped into data classes which are always collected together and form a suitable abstraction level (question classes) for dialog control. Within a question class the dialog is controlled locally by further question rules. All problem-solving methods only need to know the name of the question class when requesting further information. In the case of competing requests the best question class is selected via an agenda (see Sect. 34.3).

2. The data are processed to data abstractions, which form a suitable abstraction level for the actual diagnostic interpretation. If the various problem-solving methods use different, overlapping terms, it is not necessary to put redundant questions to the user, but the terms can be derived as different symptom abstractions from the same basic data.

In order to facilitate the integration of the diagnostics expert system with other programs, it should be possible to have the data requested from the user, extracted from a data base or taken over from another program as required. A suitable interface is provided by the question classes, which can be classified into different types and call the interface programs corresponding to their type. The result in all cases is that the observations are given values. The derivation of the symptom abstractions is data-controlled and immediately follows the data collection.

## 34.7 Programming

When looking for program sections in one problem-solving method which can be adopted by another, a distinction must be made between general and specific synergetic effects. General synergetic effects concern those parts which are not specific for the problem-solving method. General basic tools can make these parts available to other methods. Suitable tools of this nature are the implementation of graphical primitives (hierarchies, forms, etc.) in knowledge acquisition (see Sect. 12.5), the hypertext technique in the explanation and a basic rule interpreter and a frame concept in the problem-solving component.

Specific synergetic effects occur when different problem-solving methods can be programmed in a similar way. In the classification methods considered here this applies to the two types of causal models. The simulation mode in particular, including the summation of components and the consideration of feedback loops, is similar for functional and fault models, so that it can be implemented as two versions of the same basic algorithm.

## 34.8 State of the Art

Most approaches to integration try to combine heuristic and causal (usually set-covering) knowledge. At the problem-solving level a heuristic solution is at-

tempted first, and the less efficient causal model is used only if this does not succeed. Knowledge acquisition, on the other hand, usually starts from a causal fault model, from which heuristic rules should be generated automatically, which is why the heuristic knowledge is often called "compiled knowledge" [Chandrasekaran 82]. An example of "knowledge compiling" is given in [Steels 85], but using very simple knowledge representations: fault models without state parameters are transformed to heuristic knowledge without evidence values. This avoids the actual problem of the evidence valuation of heuristic rules, since the rule uncertainties, which constitute an important part of heuristic, empirical knowledge, can hardly be derived from fault models. In our opinion the term "knowledge compiling" is therefore misleading.

In MORE [Kahn 88], on the other hand, the evidence values are treated as additional knowledge for a heuristic expert system: in this case also the expert first develops a fault model, from which MORE generates a heuristic rule for each relationship. The evidence of the rule is requested from the expert, however, and MORE only carries out a few consistency tests. In addition, MORE has seven other strategies for analysing the fault model, with which it can find inconsistencies or gaps in the knowledge base and draw the expert's attention to them. It must be admitted, however, that the experts have largely ignored MORE's comments.

In MORE the fault model served only as an aid in the acquisition of heuristic knowledge, whereas in its successor MOLE [Eshelman 88] fault models are applied directly to problem-solving, though with no heuristic component and no reasoning with uncertain knowledge.

Another way towards integration is the automatic evaluation of knowledge from case data bases for the generation of rules. One important version of this approach is what is known as "similarity-based learning", where the aim is to derive general rules from many individual case. PLANT/DS [Michalski 80] is a typical example, together with the corresponding, commercially available expert system tool RULEMASTER. The main problem is the correct interpretation of statistical correlations, which is possible only with the aid of background knowledge about potential interfering factors. Thus a correlation between the observations $A$ and $B$ may have several interpretations: (1) $A$ causes $B$, (2) $B$ causes $A$, (3) a third factor $C$ causes both $A$ and $B$, (4) the correlation is fortuitous, i.e., not representative. To achieve an efficient system one must therefore strive to include background knowledge. A good example of this is the RX system [Blum 84, 88], which takes into account causal knowledge about "third factors" which can influence a correlation. Another, more recent approach to the inclusion of background knowledge is "explanation-based learning"[de Jong 86], in which a given causal model is used to evaluate the significance of the symptoms of the cases investigated for the diagnosis.

An integration of case-based knowledge and fault models, aimed only at generating hypotheses, not heuristic rules, was implemented in CASEY [Koton 88]. CASEY looks for a known case which is as similar as possible to the new case

and attempts to modify its causal explanation, which is stored with the case, on the basis of the different symptoms of the new case.

# 35. Aspects of the Overall Integration

Starting from the integration of the classification methods discussed in the last chapter, we consider below the additional integration of the construction and simulation methods indicated in Fig. 33.1, namely skeletal construction, the propose-and-revise strategy for configuration, a simple form of the least-commitment strategy, case-comparing construction and single-phase simulation. The common feature of these methods is that they all fit in with the simple basic conception described in Sect. 12.2, in which all objects of the domain are named explicitly. In our opinion the integration of more powerful construction is problematical on account of the increasing complexity, so that the coupling of independent shells may be preferable.

## 35.1 Survey

In the integration of classification, construction and simulation methods we are concerned mainly with cooperating problem-solving methods and less with competing ones as for the classification methods. This has the following consequences for the different aspects of an integration:

- *Programming.* The most significant synergetic effects are produced at the level of the programming of the shell, since the four components (problem-solving, explanation, knowledge acquisition and data collection dialog) display many similarities on account of the same basic conception of all the problem-solving methods.

- *Data collection.* Especially for the integration of diagnosis, therapy and simulation for biological and technical systems, the results of one problem-solving method should be the starting data for the next one.

- *Knowledge acquisition.* Knowledge transfer is necessary only when simulation methods are used to check solutions derived by classification or construction methods. This also includes the transition from construction via simulation to the diagnostics (maintenance) of a technical system.

- *Knowledge representation, knowledge manipulation, explanation.* Since the problem-solving methods solve different problems, no integration is required at this level.

The basic ideas of programming integration have already been explained in earlier chapters and may be summarized as follows:

- For skeletal construction: the main requirement is that a new object type "skeletal plan" must be added to a heuristic classification shell (Sect. 22.6).
- For the simple propose-and-revise strategy: the data pre-processing of a heuristic classification shell must be modified above all by the addition of correction rules (Sect. 23.6).
- For the simple least-commitment strategy with given (not generated) constraints: the rule processing must be modified to allow the propagation of constraints, which would require above all a partial evaluation of the rules.
- For case-comparing construction: the similarity evaluation of case-comparing construction is the same as that of case-comparing classification (Chap. 26).
- For the simulation of classification results: this is part of the technique of classification with functional models (Chap. 18).
- For the simulation of action plans: action plans can also be simulated with a functional model by representing the actions as components with a certain behavior, which gradually modify the states, represented as materials. The material parameters (e.g., pressure or temperature in the motor) correspond to the properties of objects in planning (e.g., the location of a person in travel planning, transformations of a workpiece in production planning, the position of a block in the Blocks World or the current state of genetic substances in experiment planning). Although this representation would be too complicated for the model-based generation of plans, it is sufficient for the simulation of an existing plan.
- For the simulation of configurations: this can also be done by functional models. However, as in the simulation of action plans, additional knowledge concerning the behavior of the configuration elements, regarded as components, is required. In addition, the correct order of the components must be laid down, and the materials must be specified.

The following short descriptions are intended to illustrate the interactions between the different problem-solving methods and therefore represent an extreme simplification of the actual complexity of the domains. The first two examples – decision-making in medicine and the tendering, configuration and maintenance of technical systems – may be said to represent the *leitmotiv* of our integration. In the third example, however, – production planning and quality control – we prefer a coupling of different tools, since an integrated tool would be prohibitively complex.

## 35.2 Decision-Making in Medicine

The two main parts of medical decision-making are diagnostics and therapy. In the diagnostic process hypotheses are first generated and tested, and then one or more working diagnoses are selected for the therapy. Heuristic classification is best suited for the generation of hypotheses, which are tested preferably by simulation with fault models, i.e., with patho-physiological knowledge, since serviceable functional models are seldom available in medicine. In making the final selection of the working diagnosis, statistical and case-based knowledge may also be decisive. Case-based knowledge is also suitable for the prognosis of the illness.

The determination of the therapy is often just another classification step, in which a standardized therapy scheme is selected according to the working diagnosis, but a configuration may be required in some cases in order to adjust different therapy elements to one another. The required knowledge can often be represented in skeletal plans or, if a recursion is required, in flow charts. Since many therapies have side effects and interactions between therapy elements are possible, it is often advisable to use the variant of the propose-and-revise strategy described in Sect. 23.5 for parameters adjustment using diagnosis/therapy/side-effect tables and to test the effects of the therapy in a simulation. Finally the actual effects of the applied therapy should be evaluated to see whether they confirm the working diagnosis or indicate that a revision is necessary. The loop between diagnosis and therapy continues until recovery is achieved.

In all, the following problem-solving methods participate: (a) set-covering (patho-physiological), heuristic, statistical and case-comparing classification; (b) configuration with skeletal plans, with flow charts and with the propose-and-revise strategy on the basis of diagnosis/therapy/side-effect tables; (c) qualitative, single-phase simulation with fault models (see Fig. 35.1).

A concrete example for these numerous interactions is the treatment of diabetes. First the diagnosis diabetes mellitus must be established, and then the type of diabetes must be determined in a further classification step. This in turn determines the therapy, which consists of several elements (see Sect. 22.6) and must therefore be configured. The precise adjustment of the insulin dose can best be checked in a simulation model.

Another example is the treatment of heart diseases. First a diagnosis must be established. However, since the primary causes of the disease are often untreatable, medication and other measures are aimed at minimizing the harmful consequences. Since the medicines have side effects, their combination and dosage depend upon many parameters and can be optimized with a configuration component and a simulation model.

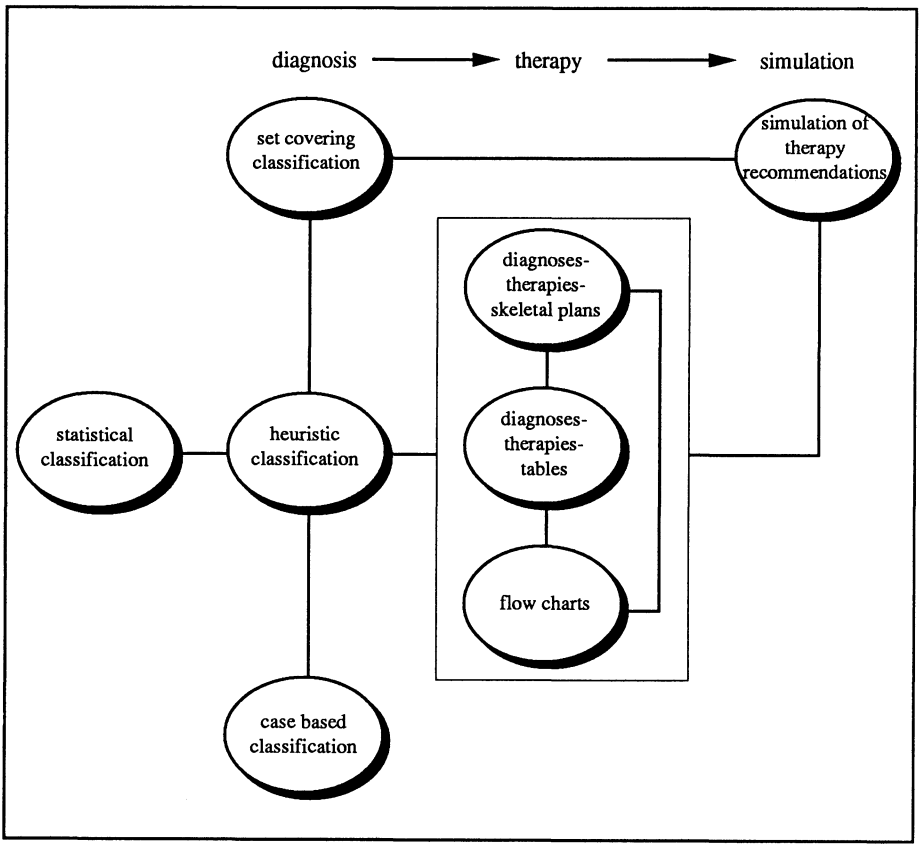



**Figure 35.1** The problem-solving methods taking part in medical decision-making

## 35.3 Tendering, Configuration and Maintenance of Technical Systems

The life cycle of a technical system (e.g., a production machine) consists of the preparation of the offer, the customer's order, the complete configuration, the production, the installation and the subsequent maintenance. We will assume that the system is assembled from relatively few modules and discuss below the support of all phases except the production, which is treated in the next section, and the installation.

Following a customer enquiry a manufacturer of technical systems prepares an offer which is tailored to the customer's requirements. Expert systems for the preparation of tenders would be especially useful, since they would markedly reduce the work load, which is often quite considerable, and thus make it much easier to provide alternative offers, so that the customer can examine different compromises between his ideal requirements and price considerations. Since the manufacturers generally configure their systems from prefabricated modules, suitable problem-solving methods are propose and revise, least commitment or (with

a hierarchical knowledge structure) skeletal construction. Case-comparing configuration is also a possibility, with the additional advantage that a suitable comparative case can be useful in all the subsequent problem-solving methods. When the offer has been prepared the system must be configured, which involves a more detailed specification of the tender and requires additional knowledge; however the data and the results of the offer preparation should be passed on.

After the production and installation of the system the maintenance begins. Since no empirical knowledge is available for fault-finding in new systems, the functional model must be used for diagnostics at first. Ideally the functional model is obtained from the result of the configuration. In addition, the behavior of the components given by the configuration must be described, and their relationships to the respective materials with their relevant properties must be given. In model-based construction this knowledge would already be present, but it is not necessary in the heuristic configuration assumed here and must therefore be added at this point at the latest.

Unfortunately, not all cases are ideal in the above sense, since the degree of abstraction in the maintenance may be different from that in the configuration, e.g., because the configuration is concerned with modules and the causal model with the functional units. In unfavorable cases the transition requires considerable additional knowledge. It is worth the trouble, however, also for another reason: we obtain a model with which the function of the configuration may be tested by simulation.

For maintenance purposes information concerning typical faulty states and the behavior of the components must be added to the simulation model. Since we are interested only in the propagation of faulty states in maintenance, the value ranges of the materials should be transformed from absolute values to deviations from the normal values.

Considerable support for the transition from configuration to diagnostics with functional models can be provided by a library of components with descriptions of their behavior in the normal and faulty states.

With increasing experience the functional model for the diagnostics is simplified by first converting it into a fault model and then adding heuristic knowledge. This also makes the fault-finding process more efficient. As soon as the symptom investigation has been standardized, successfully solved cases are collected to form a case data base, which then makes possible statistical and case-comparing fault-finding in addition.

The subsequent repair usually consists simply of the replacement of the faulty part and a test to check that the fault has been corrected. If there are several plausible hypotheses, the system should determine the order of the tests, based on the probabilities of the hypotheses and the associated replacement costs. Apart from this very simple form of configuration, skeletal construction or the variant of the propose-and-revise strategy for parameter adjustment using diagnosis/repair/side-effect tables (see Sect. 23.5) may be useful in more complex situations where the repairs are adjustable parameters of a production system.

In all, the following problem-solving methods participate: (a) configuration with propose-and-revise, least-commitment and skeletal construction; (b) single-phase simulation; (c) functional, set-covering, heuristic, statistical and case-comparing classification; (d) again configuration with skeletal construction and with propose-and-revise using diagnosis/repair/side-effect tables, whereby the repair represent adjustable system parameters (see Fig. 35.2).

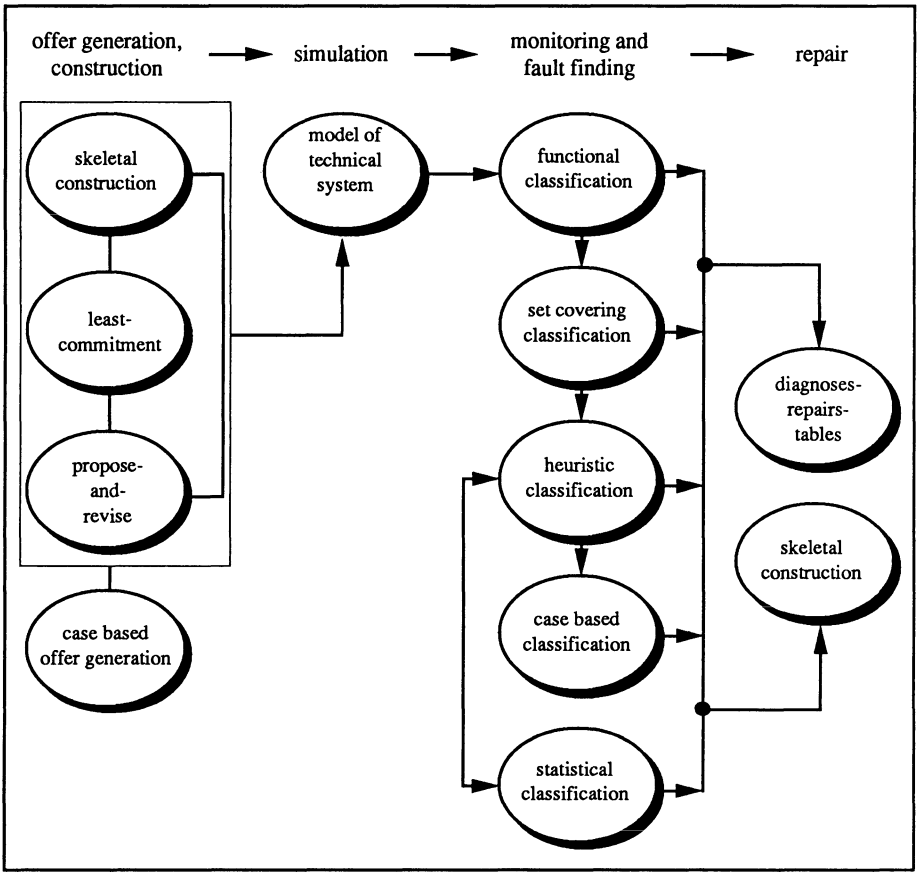



**Figure 35.2** Knowledge transfer between the problem-solving methods in the offer generation, configuration and maintenance of technical systems

Examples are to be found particularly in the mechanical engineering industry, e.g., in the sale of a complete production plant or a part of such a plant, such as a test rig, for example. If a customer requires a test rig for his own production it must be configured for his special requirements. For this purpose basic modules of the tendering company are put together in a suitable way. The manufacturer is responsible for the maintenance for the lifetime of the test rig.

In many cases the preparation of an offer and the configuration, on the one hand, and the maintenance on the other hand are separated and regarded as different problems. This is especially true if the difference between the configuration

result and the functional model for the maintenance is too great for a reasonable transformation. Even then, however, the two phases are themselves still so complex that they each require an integration of problem-solving methods. Thus the number of participating problem-solving methods in the maintenance is hardly reduced. The maintenance of technical systems has a structure similar to that of the medical diagnostics/therapy problem discussed above. The main differences lie (a) in the degree of complexity, since biological systems are usually much more complicated than technical systems; (b) in the knowledge fluctuation, since technical systems are continually changing; (c) in the degree of understanding, since for technical systems we usually have better functional models but less empirical knowledge than for biological systems; (d) in the ease of finding a therapy, since in technical systems faulty components can often simply be exchanged.

## 35.4 Job and Production Planning and Quality Control

In the previous section the construction of technical systems was limited to configuration with given modules. In this section the emphasis lies on the planning of the actual production of the ordered items (workpieces) in a factory in which many other tasks must be performed at the same time. First an analysis must be carried out to specify the individual steps in the production of the workpiece and to determine which machines are required and in what state. In the general case this job planning would require model-based construction methods, but it can often be regarded as a constraint problem in which the machining elements of the workpiece or the corresponding elementary machine operations are mapped onto a minimal set of different machine states, using either the least-commitment (cf. ExAP, Sect. 25.5) or the propose-and-revise strategy to fulfill the constraints. A case-based production planning is also possible, in which a previously manufactured workpiece is sought with the greatest possible similarity to the new product (cf. CcC, Sect. 20.5), in order to use its construction plans as far as possible.

The local optimization of the production process for the individual workpieces (job planning) is followed by the global production planning, where the aim is to optimize the execution of several production tasks with the available machines. Like timetable generation, production planning is a complex assignment problem with a huge solution space and numerous constraints such as the local job plans and number of units for each order, the utilization or shortage of machines, the suitability of machines, time limits and the possibility of machine break-downs. The propose-and-revise strategy is a suitable method for this problem. The production plan should be tested with a simulation model, for which purpose the method of numerical, multiple-phase simulation is suitable.

The production is followed by the quality control of the workpieces (e.g., on a test rig), which is a double classification problem: the faults in the workpieces must be classified, and the cause of a fault in the production must be found. The first classification task is often relatively easy under the controlled conditions of the quality control and can be adequately solved with decision tables or

by direct heuristic classification. Finding the cause in the production process is often much more difficult and may require all classification methods, in particular a functional model, which may be derived partially from the local and global production plans. Remedying the cause, e.g., repairing a machine, may make a revision of the production plan necessary to minimize the production loss.

Although in this view of the production process complex construction methods such as least commitment, propose-and-exchange and multiple-phase simulation predominate, classification methods for recognizing and remedying faults also play an important role in quality control (see Fig. 35.3).

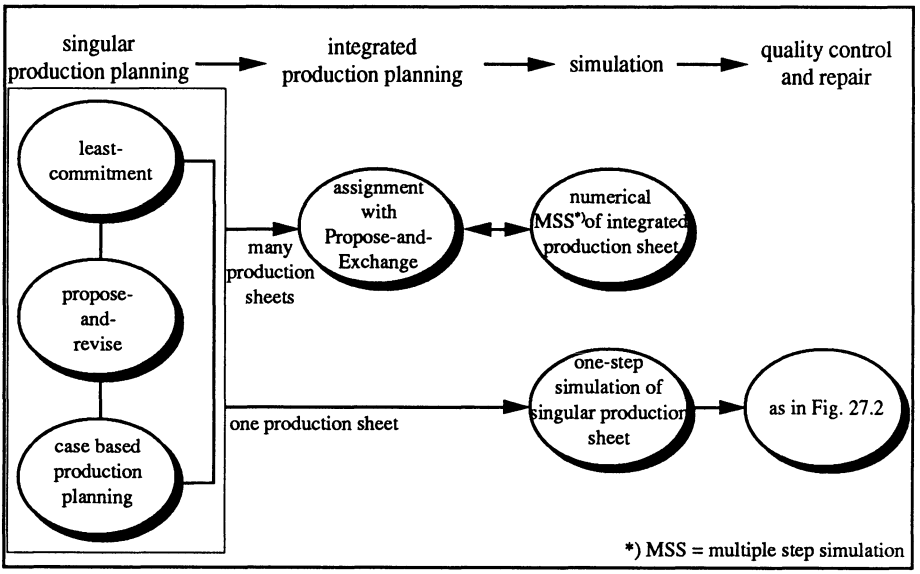



**Figure 35.3** Knowledge transfer between the problem-solving methods used in local and global production planning and quality control

This example shows the limits of our integration model. Since the basic tools required for propose-and-exchange, least-commitment (cf. examples in Sect. 24.5 and 25.5) and numerical multiple-phase simulation are themselves relatively complex, a coupling of independent tools is preferable to integration in this case.

# Appendix: Survey of Knowledge Representation Formalisms

The most important aspects of basic knowledge representation are summarized briefly below. The summary may also be used as a check list to evaluate the power of a general expert system tool. Further explanations of the various aspects can be found in Chaps. 5–10.

## 1. **Rules** (see Chap. 5)

- Expressive power:
  - look-up in the knowledge base
  - look-up and calculate
  - pattern matching (and calculation)
  - unification
  - call of functions in programming language.
- Structuring:
  - in the precondition: core, context, exceptions
  - classification of action types.
- Processing:
  - forward chaining
  - backward chaining
  - type of conflict resolving strategy.
- Rule activation:
- through associated objects
- as rule package with no direct object reference.

## 2. Objects/Frames (see Chap. 6)

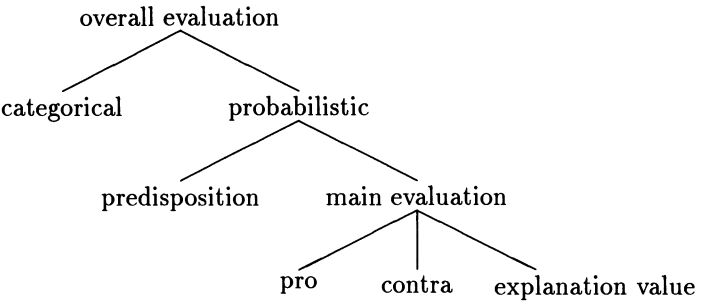
- Inheritance mechanism:
  - simple inheritance
  - multiple inheritance
  - selectivity of the inheritance
  - mapping function in inheritance.
- Attached procedures.
- Message passing.
- Expected values.
- Automatic classification.

## 3. Constraints (see Chap. 7)

- Representation of the constraints:
  - rules
  - tables
  - functions in programming languages.
- Propagation algorithm:
  - values
  - value sets
  - terms with variables.
- Ability to generate and withdraw plausible assumptions.
- Weighting and removability of the constraints and their division into priority classes.

## 4. Probabilistic Reasoning (see Chap. 8)

- Representation
  - granularity of the evidence scales
  - splitting of the evaluation:




- Reckoning method:
  - Bayes' Theorem
  - Dempster-Shafer theory
  - MYCIN model
  - INTERNIST model
  - MED1 model
  - ...
- Decision criteria
  - threshold values
  - differential diagnostics.

## 5. **Belief Revision** (see Chap. 9)

- Basis:
  - JTMS with direct justifications
  - ATMS with basic assumptions as justifications.
- Worlds (comparison of conclusions based on different assumptions).
- Treatment of justification loops.
- Treatment of exceptions.
- Efficiency and generality:
  - Are all inference mechanisms supported?
  - Does the revision mechanism require further information?

## 6. Temporal Data Bases and Temporal Reasoning (see Chap. 10)

- Basic representation:
  - point based
  - interval based.
- Precision:
  - exact
  - qualitative
  - quantitative, inexact.
- Reference:
  - time scale
  - simple reference events
  - multiple reference events.
- Are time units represented?
- Predicates and functions for evaluation:
  - validity of a fact during a time interval
  - temporal relationship between facts
  - changes with time.

# Glossary

## I. Problems: General Basic Terms

**Problem:** Question or set of questions characterized by certain **problem features** (for short: **features** or *observations*) and for which a **problem solution** (for short: **solution**) is sought. The problem solution may be selected or put together from solution elements.

**Problem domain:** Set of similar *problems* which refer to the same *problem features* and *problem solutions* in different combinations.

**Special field:** Set of *problem domains* with the same *technical terminology*, e.g., medicine, loan advising.

**Technical terminology (technical language):** Terms for important concepts of the *special field*, which, unlike the concepts of everyday language, are precisely defined.

**Problem type:** Set of different *problem domains* which are similar from the point of view of the user, e.g., fault-finding, monitoring, interpretation, *planning, assignment.* However, the similarity need not result in common *problem-solving methods*, as in the case of *problem-solving types.*

**Problem-solving type:** Set of different *problem domains* which can be solved by the same *problem-solving methods.* Some important, general problem-solving types are *classification (diagnostics), construction* and *simulation.*

**Problem class:** Generic term for *problem type* and *problem-solving type*, used when it is not required to make a distinction between the latter two terms.

## II. Problem Solving: General Basic Terms

**Knowledge:** Data with implicit or explicit information about their use for solving *problems.* The *knowledge representation* and the *problem-solving methods* have a reciprocal influence on each other and should not be treated independently.

**Knowledge representation:** Formal framework for encoding the *knowledge*, with which procedures for interpreting the knowledge, i.e., *problem-solving methods* are associated. We distinguish between *basic* and compound knowledge representations.

**Basic knowledge representation:** Fundamental *knowledge representations*, with which *inference strategies* are coupled. The most important basic knowledge representations in *expert systems* are (a) *rules* with *forward* or *backward chaining,* (b) *frames* with inheritance and attached procedures, (c) *constraints* with local propagation, (d) evidence values with combining scheme, (e) justification networks with revision procedure (*TMS*) and (f) time representations with the associated propagation techniques.

**Problem-solving method:** Algorithm specifying how domain-specific *knowledge* is used for problem solving. It implies a certain determination of the *knowledge representation*, unlike a sorting algorithm, for example, which does not refer to any external domain-specific knowledge. The strength of a problem-solving method increases with the extent to which it determines the use and representation of the domain-specific knowledge.

**Weak (basic) problem-solving method:** *Problem-solving method* which is open for a great variety of domain-specific *knowledge representations* and functions and is therefore widely applicable, but offers little assistance in *knowledge acquisition.*

**Strong problem-solving method:** *Problem-solving method* which defines the representation and function of the domain-specific knowledge to a large extent and therefore has only a limited field of application, but offers comprehensive assistance in *knowledge acquisition.* Strong problem-solving methods are often specializations of *weak problem-solving methods.*

**Inference (derivation) strategy:** *Problem-solving method* for the evaluation of *basic knowledge representations,* e.g., *forward* or *backward chaining* in the case of *rules.* Inference strategies are relatively *weak problem-solving methods,* since there are often many variations within the *basic knowledge representations,* and quite different function types are possible, e.g., in the case of *rules,* different types of post-condition, such as implication or action rules or meta-rules.

**Agenda:** Ordered list of objects, usually used to find the currently best object when frequent changes are being made. An agenda may be regarded as a task queue, from which always the currently most important task is selected and carried out.

**Data pre-processing:** processing the *problem features (observations)* to data abstractions, which are generally terms of the *technical terminology* and facilitate the solution of the *problem.*

**Solution space:** Set of all theoretically possible *problem solutions.* In *construction* it is considerably greater than in *classification.*

## III. Basic Knowledge Representations

**Rule:** *Basic knowledge representation* by means of which directional relationships between objects can be represented. Non-directional relationships are represented by *constraints.* A rule (e.g., $A \rightarrow B$) always has a pre-condition and an action, which may be carried out when the pre-condition is fulfilled. The two fundamental *inference strategies* for rule systems are **forward** and **backward chaining.** In forward chaining the rules are selected according to whether their pre-conditions are fulfilled by the data in the data base, and in backward chaining according to whether their actions contributes towards reaching a given goal.

**Frame:** *Basic knowledge representation* in which all the knowledge about an object is collected together in a data structure. Typically a frame consists of pointers to higher-ranking frames, attributes (slots), facets and values. The facets may contain knowledge about value ranges, expected values (defaults) and attached procedures (actions to be performed whenever a value is entered or requested). The pointers enable values and facets to be inherited from the predecessor frame.

**Constraint:** *Basic knowledge representation* for expressing non-directional relationships between objects. A constraint consists of a name, a definition and variables (e.g., "Adder-1", "$A+B=C$", $\{A, B, C\}$). A set of constraints which are connected

by common variables is called a **constraint network**, and a constraint network with an initial assignment of a subset of the variables is called a **constraint problem**.

**TMS** (truth maintenance system): Generic term for techniques of belief revision for preserving consistency in the event of a change of the situation (e.g., correction of a fact by the user, replacement of a default assumption due to new information). Simple, but inefficient techniques are the complete revision of all conclusions on the basis of the latest data or chronological backtracking. More efficient techniques use an additional data structure for storing justifications. In **JTMS** (justification-based TMS) the direct justifications of a conclusion are stored; in **ATMS** (assumption-based TMS) the basic assumptions on which the justifications are based are also stored.

## IV. Classification (Diagnostics)

**Classification (Diagnostics):** *Problem-solving type* in which the *solution* is selected from a set of given *solutions*, possibly via a *diagnostic intermediate structure*, by a *problem-solving method*. In diagnostics the *problem features (observations)* are also called **symptoms**, and the *solutions* **diagnoses**. With $n$ possible diagnoses, the size of the solution space is $n$ if no *multiple diagnoses* are allowed (**single-fault assumption**), otherwise $2^n$.

**Simple (certain) classification:** *Classification* on the basis of straightforward *rules* of the type *"symptom(s)* $\rightarrow$ *diagnoses"*, with *diagnostic intermediate structure*.

**Heuristic classification:** *Classification* on the basis of uncertain *rules* of the type *"symptom(s)* $\rightarrow$ *diagnosis* with the empirical probability $X$" and a *diagnostic intermediate structure*. The value attached to a diagnosis increases with the accumulated empirical probability derived from the observed *symptoms*.

**Set-covering classification:** *Classification* on the basis of *rules* of the type *"diagnosis* (with a certain gravity) $\rightarrow$ *symptom* (with a certain gravity)" and a *diagnostic intermediate structure*. The value attached to a diagnosis is greater, the better it explains (i.e., covers) the observed symptoms, and the fewer non-observed symptoms it covers.

**Functional classification:** *Classification* on the basis of a model of the normally functioning system, in which input materials are transformed into output materials by the components. The *symptoms* are discrepancies between the expected and observed values of the (output) materials, and the *diagnoses* are normally possible changes in the behavior of the components. As in the case of *set-covering classification*, the value attached to a diagnosis is greater, the better it explains the observed discrepancies.

**Model-based (causal) classification:** Generic term for *functional* and *set-covering classification*.

**Case-based classification:** *Classification* on the basis of a collection of successfully solved cases. For each new case, one is selected from the collection which is as similar as possible, and if the similarity is sufficient its *diagnoses* are adopted.

**Statistical (probabilistic) classification:** *Classification* on the basis of a statistical evaluation of successfully solved cases.

**Establish-refine strategy:** *Problem-solving method* for *classification*, by which, in a strict diagnosis hierarchy, a coarse diagnosis is first established and then refined by establishing one of its successors, etc.

**Hypothesize-and-test strategy:** *Problem-solving method* for classification, by which hypotheses are first generated from the given *symptoms*; the best of these is then explicitly tested (possibly also by requesting additional symptoms), and the loop is repeated until some terminating criterion is satisfied (e.g., there are no more hypotheses to be tested, or all *symptoms* are explained by established *diagnoses*).

**Diagnostic intermediate structure:** All intermediate results on the way from *symptoms* to *diagnoses* in *classification*. Intermediate results may be the *diagnoses* of one diagnostics problem and at the same time the *symptoms* of a succeeding diagnostics problem.

**Multiple diagnoses:** Set of *solutions* in *classification*. Multiple diagnoses can be regarded as a sort of connecting link between *classification* (multiple selection of *solutions*, i.e., single diagnoses) and *construction* (building up the solution from solution elements, i.e., single diagnoses).

## V. Construction

**Construction:** *Problem-solving type* in which the *solution* cannot be selected as a single entity as in *classification*, but must be put together from solution elements with a *problem-solving method*. The *solution space* is usually very large (e.g., in a timetable problem with $n$ classes and $m$ lessons per class of the order of $m!^n$.

**Planning:** *Problem type*, belonging to *construction*, in which it is required to find a sequence of operators which transforms the initial state into a goal state (cf. *configuration*). Example: planning robot actions. In a general sense, solving any problem constitutes a planning process.

**Configuration:** *Problem type*, belonging to *construction*, in which it is required to find a solution object which fulfills certain requirements. It thus differs from *planning*, where the target state (i.e., the solution object) is given, and the way to achieve it has to be found. In configuration, the solution objects must be instantiated, or their properties determined, in contrast to *assignment*, where the task is to find a certain arrangement of given solution objects.

**Assignment:** *Problem type*, belonging to *construction*, in which at least two disjunctive sets of given objects must be mapped onto each other in such a way that certain requirements are fulfilled. Examples: timetables or machine occupancy plans.

**Skeletal construction:** *Problem-solving type* for *construction*, in which a skeletal plan is selected on the basis of the *problem features*, expanded and refined. A skeletal plan is an abstract solution, the elements of which must be refined. Skeletal construction is a special form of the *generate-and-test strategy*.

**Generate-and-test strategy:** *Problem-solving method* for *construction*, in which the solution is generated step by step by adding solution elements to the existing partial solution and testing each addition immediately to see if it is correct. In contrast to the *hypothesize-and-test strategy*, solution elements, and not solutions, are tested.

**Propose-and-revise strategy:** *Problem-solving method* for *construction*, where, in contrast to the *generate-and-test strategy*, no immediate test of a solution step is possible during the generation of the solution. Erroneous approaches to the solution may therefore be detected very late and have to be revised. The special feature of this strategy compared to other backtracking approaches is that domain-specific revision *knowledge* is available for the efficient correction of false solution attempts.

**Propose-and-exchange strategy:** Specialization of the *propose-and-revise strategy* for *assignment* problems.

**Least-commitment strategy:** *Problem-solving method* for *construction*, in which the value ranges of the parameters are restricted as far as possible by the generation and propagation of *constraints*. In contrast to the *propose-and-revise strategy*, no definite value is assigned to a parameter until it is absolutely necessary.

## VI. Simulation

**Simulation:** *Problem-solving type* where the solution consists of the prediction of the effects of given causes in a *model*. This contrasts with model-based *classification*, in which the task is to deduce the assumed causes (*diagnoses*) from the given effects (*symptoms*).

**Single-phase simulation:** A subset of the *problem-solving type simulation*, in which only one value is calculated for each simulation parameter. In **multiple-phase simulation**, on the other hand, a series of values valid for different instants or intervals of time is calculated for the parameters.

## VII. Knowledge Types

**Heuristic knowledge:** *Knowledge*, often uncertain, which points from *problem features* to *problem solutions*, possibly via intermediate steps, e.g., rules of the type "If problem-feature-1 then solution-element-1". Heuristic knowledge is aim-specific, i.e., easy to interpret, but rarely applicable to different aims.

**Model-based (causal) knowledge:** *Knowledge* about general relationships between *problem solutions* and *problem features* that can potentially be used in different ways, in particular also for *simulation*. However, in *classification* or *construction*, problem-solving is usually more complicated with causal knowledge than with *heuristic knowledge*.

**Statistical (probabilistic) knowledge:** Uncertain *knowledge* gained from a statistical analysis of a set of successfully solved cases.

**Case-based knowledge:** *Knowledge* about successfully solved cases, i.e., a set of *problem features* with the correct *problem solution*. It is used for analogous reasoning, whereby a known case which comes as close as possible to the new case is chosen, and its *solution* directly adopted or modified.

## VIII. Expert Systems

**Knowledge-based system:** Program for solving *problems*, in which the *problem-solving methods* and the *knowledge* are kept separate, which makes it amenable to modification by exchanging the *knowledge* with the same *problem-solving method* and contributes to the explainability by enabling the *knowledge* used for deriving the *problem solution* to be given.

**Expert system:** *Knowledge-based system* which is intended to reproduce the reasoning ability and the domain-specific *knowledge* of experts.

**Expert system tool:** Programming environment for the construction of *expert systems*, which is intended to reduce the working time required in comparison with the

use of universal programming environments. An expert system tool supports certain *knowledge representations* and consists of a *problem-solving*, a *knowledge acquisition*, an *explanation* and an *interview component.* **Simple expert system tools** provide only one *knowledge representation* with the appropriate *problem-solving method*, whereas *hybrid expert system tools* combine several *knowledge representations.*

**General expert system tool:** *Expert system tool* which supports *basic knowledge representations* and *inference strategies.*

**Problem-specific expert system tool (shell):** *Expert system tool* which is designed specially for certain *problem-solving types* and provides the required *knowledge representations* and *strong problem-solving methods.*

**Domain-specific expert system tool:** *Problem-specific expert system tool* which in addition contains knowledge about a *special field.*

**Knowledge base:** Part of the *expert system* in which the *knowledge* is stored. The knowledge base is edited with the *knowledge acquisition component* and interpreted with the *problem-solving component.*

**Knowledge acquisition:** Formalization and input of *knowledge* for an *expert system.*

**Knowledge acquisition component:** Part of an *expert system* which supports the *knowledge acquisition.* The knowledge acquisition components of *problem-specific expert system tools* should not require too much learning effort on the part of the expert for their operation.

**Problem-solving component:** Part of an *expert system tool* which interprets the *knowledge* required for the *solution* of a *problem* specific by *problem features.*

**Explanation component:** Part of an *expert system tool* which reveals with which *knowledge* the *problem solution* was derived from the *problem features.*

**Interview component:** Part of an *expert system tool* which serves for registering the *problem features.* If the *problem features (observations)* are read from measuring instruments or files without any direct help from the user, the *expert system* is called **embedded**, otherwise **interactive.**

**Dialog component:** A compound component consisting of the *knowledge acquisition, explanation* and *interview components.*

# References


### Chap. 1: Characterization and History of Expert Systems

Barr, A. and Feigenbaum, E.: *The Handbook of Artificial Intelligence,* Vol. 1, Morgan Kaufmann, 1981

Dreyfus, H. and Dreyfus, S.: *Mind over Machine,* Free Press, 1986.

Duda, R. and Shortliffe, E.: *Expert Systems Research,* Science **220**, 261–263, 1983.

Newell, A.: *The Knowledge Level,* AI Journal **18**, 86–127, 1982.

Newell, A. and Simon, H.: *GPS, a Program that Simulates Human Thought,* reproduced in Feigenbaum, E. and Feldmann, J.: *Computers and Thought,* McGraw-Hill, 279–293, 1972 (1963).

Nilsson, N.: *Principles of Artificial Intelligence,* Springer, 1982.

### Chap. 2: Programming Languages and Expert System Tools

Karbach, W. and Linster, M.: *Knowledge Acquisition for Expert Systems: Techniques, Models and Software Tools* (in German), Hanser, 1990.

Newell, A.: *The Knowledge Level,* AI Journal **18**, 86–127, 1982.

Rich, C. and Waters, R.: *Artificial Intelligence and Software Engineering,* in Grimson, E. and Patil, R. (eds.): *AI in the 1980s and Beyond: An MIT Survey,* MIT Press, 1987.

### Chap. 3: Use and Usability of Expert Systems

Barker, V. and O'Connor, D.: *Expert Systems for Configuration at Digital: XCON and Beyond,* CACM **32**, No. 3, 298–317, 1989.

Coy, W. and Bonsiepen, L.: *Experience and Calculation: Critique of the Expert System Technique* (in German), Springer Informatik-Fachberichte **229**, 1989.

ESS: *Choosing the Right Tools for Your Task,* Expert System Strategies **4**, No. 12, 1–7, 1988.

Feigenbaum, E., McCorduck, P. and Nii, P.: *The Rise of the Expert Company,* Times Book, 1988.

Mertens, P., Borkowski, V. and Geis, W.: *Industrial Expert System Applications* (in German), Springer, 2nd edition, 1990.

Slage, J. and Wick, M.: *A Method for Evaluating Candidate Expert System Applications,* AI Magazine **9**, 44–53, 1988.

## Chap. 4: Logic

Bibel, W.: *Automated Theorem Proving,* Vieweg, 1983/1987.

Charniak, E. and McDermott, D.: *Introduction to Artificial Intelligence,* Addison-Wesley, 1985.

Clocksin, W. and Mellish, C.: *Programming in PROLOG,* Springer, 1981.

Genesereth, M. and Nilsson, N.: *Logical Foundations of Artificial Intelligence,* Morgan Kaufmann, 1987.

Lloyd, J.: *Foundations of Logic Programming,* 2nd edition, Springer, 1987.

Melvin, M. and Fitting, F.: *First Order Logic and Automating Theorem Proving,* Springer, 1991.

Minsky, M.: *A Framework for Representing Knowledge,* in Winston, P. (ed.): *The Psychology of Computer Vision,* McGraw-Hill, 1975.

Nilsson, N.: *Principles of Artificial Intelligence,* Springer, 1982.

Robinson, J.: *A Machine-Orientated Logic Based on the Resolution Principle,* JACM **12**, 23–41, 1965.

Siekmann, J.: *Unification Theory,* J. Symbolic Computation, 1987.

Walker, A., McCord, M., Sowa, J. and Wilson, W.: *Knowledge Systems and PROLOG,* Addison-Wesley, 1987.

## Chap. 5: Rules

Brownston, L. et al.: *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming,* Addison-Wesley, 1985.

Buchanan, B. and Shortliffe, E.: *Rule-Based Expert Systems – the MYCIN Experiments,* Addison-Wesley, 1984.

Clancey, W.: *The Epistemology of a Rule-Based Expert System – a Framework-For Explanation,* AI Journal **20**, 215–293, 1983.

Clocksin, W. and Mellish, C.: *Programming in PROLOG,* Springer, 1981.

Davis, R. and King, J.: *An Overview on Production Systems,* Machine Intelligence **8**, 300–322, 1978.

Forgy, C.: *OPS5 User's Manual,* Carnegie Mellon University, 1981.

McDermott, J.: *R1: A Rule-Based Configurer of Computer Systems,* AI Journal **19**, 39–88, 1982.

McDermott, J. and Bachant, J.: *R1 Revisited: Four Years in the Trenches,* AI Magazine **5**, 21–32, 1984.

Newell, A. and Simon, H.: *Human Problem Solving,* Prentice-Hall, 1972.

Nilsson, N.: *Principles of Artificial Intelligence,* Springer, 1982.

Post, E.: *Formal Reductions of the General Combinatorial Decision Problem,* American Journal of Mathematics **65**, 197–268, 1943.

Puppe, F.: *Diagnostic Problem Solving with Expert Systems,* Sect. 4.4: *Basic Implementation Principle,* (in German), Springer Informatik-Fachberichte **148**, 1987.

Shortliffe, E.: *Computer-Based Medical Consultations: MYCIN,* American Elsevier, 1976.

Van Melle, W.: *System Aids in Constructing Consultation Programs,* UMI Research Press, 1981.

## Chap. 6: Objects/Frames

Aho, A., Hopcroft, J. and Ullmann, J.: *Data Structure and Algorithms,* Addison-Wesley, 1983.

Brachmann, R., Fickes, R. and Levesque, H.: *KRYPTON: A Functional Approach to Knowledge Representation,* IEEE Computation **16** (10), 67–73, 1983.

Brachmann, R.: *I Lied about Trees: Defaults and Definitions in Knowledge Representation,* AI Magazine **6** No. 3, 80–93, 1985 (a).

Brachmann, R. and Schmolze, J.: *An Overview of the KL-ONE Knowledge Representation System,* Cognitive Science **9**, 171–216, 1985 (b).

Goldberg, A. and Robson, D.: *Smalltalk-80 – The Language and its Implementation,* Addison-Wesley, 1983.

Haimowitz, I.: *Using NIKL in a Large Medical Knowledge Base,* MIT/LCS/TM-348, MIT, Cambridge, 1988.

Levesque, H. and Brachmann, R.: *A Fundamental Tradeoff on Knowledge Representation and Reasoning,* in Brachmann, R. and Levesque, H. (eds.): *Readings in Knowledge Representation,* Chap. 4, Morgan Kaufmann, 1985.

Minsky, M.: *A Framework for Representing Knowledge,* in Winston, P. (ed.): *The Psychology of Computer Vision,* McGraw-Hill, 1975.

Nilsson, N.: *Principles of Artificial Intelligence,* Springer, 1982.

Roberts, B. and Goldstein, I.: *The FRL Primer,* AI Memo **408**, MIT, 1977.

Stonebraker, M.: *Triggers and Inference in Database Systems,* IMAG, Grenoble, 1985.

von Luck, K. and Owsnicki-Klewe, B.: *KL-ONE – An Introduction,* in Struß, P.: *Knowledge Representation* (in German), Oldenbourg, 1991.

## Chap. 7: Constraints

Güsgen, H.-W.: *CONSAT: A System for Constraint Satisfaction,* Dissertation, University of Kaiserslautern, 1988.

Hentenryck, P.: *Constraint Satisfaction in Logic Programming,* MIT Press, 1989.

Stallman, R. and Sussman, G.: *Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis,* AI Journal **9**, 135–196, 1977.

Sussman, G. and Steele, G.: *Constraints – A Language for Expressing Almost-Hierarchical Descriptions,* AI Journal **14**, 1–39, 1980.

Voss, A. and Voss, H.: *A Uniform View on Local Constraint Propagation Methods,* Proceedings KIFS-87, Springer Informatik-Fachberichte **202**, 1989.

Waltz, D.: *Understanding Line Drawings of Scenes with Shadows,* in Winston, P. (ed.): *The Psychology of Computer Vision,* McGraw-Hill, 19–91, 1975

## Chap. 8: Probabilistic Reasoning

Charniak, E. and McDermott, D.: *Introduction to Artificial Intelligence,* Sect. 8.2: *Statistics in Abduction,* Addison-Wesley, 1985.

Doyle, J.: *Methodical Simplicity in Expert System Construction: the Case of Judgements and Reasoned Assumptions,* AI Magazine **4** No. 2, 39–43, 1983.

Genesereth, M. and Nilsson, N.: *Logical Foundations of Artificial Intelligence,* Chap. 8: *Reasoning with Uncertain Beliefs,* Morgan Kaufmann, 1987.

Gordon, J. and Shortliffe, E.: *A Method for Managing Evidential Reasoning in a Hierarchical Hypotheses Space,* AI Journal **26**, 323–357, 1985.

Kulikowski, C. and Weiss, S.: *Representation of Expert Knowledge for Consultation: the CASNET and EXPERT Projects,* in Szolovits, P. (ed.): *Artificial Intelligence in Medicine,* AAAS Selected Symposium **51**, Westview Press, 1982.

Miller, R., Pople, H. and Myers, J.: *INTERNIST1, an Experimental, Computer-Based Diagnostic Consultant for General Internal Medicine,* New England Journal of Medicine **307**, 468–476, 1982.

Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference,* Morgan Kaufmann, 1988.

Puppe, F.: *MED1: A Heuristic Diagnostic System with Efficient Control Structure,* Sect. 3.2.6: *Dealing with Uncertainties and Evidence Amplification* (in German), University of Kaiserslautern, Internal Report 71/83, 1983.

Puppe, F.: *Hybrid Diagnosis Evaluation* (in German), GWAI-86, Springer Informatik-Fachberichte **124**, 332–342, 1986.

Shortliffe, E. and Buchanan, B.: *A Model of Inexact Reasoning in Medicine,* Math. Bioscience **23**, 351–379, 1975.

Szolovits, P. and Pauker, S.: *Categorical and Probabilistic Reasoning in Medical Diagnosis,* AI Journal **11**, 115–144, 1978.

Zadeh, L.: *Fuzzy Logic and Approximate Reasoning,* Synthese **30**, 407–428, 1975.

## Chap. 9: Non-Monotonic Reasoning

Bobrow, D.: *Managing Reentrant Structures Using Reference Counts,* ACM Trans. on Programming Languages and Systems **2**, 269–273, 1980.

Brewka, G.: *Non-Monotonic Logics: an Introductionary Overview,* Proceedings KIFS-87, Springer Informatik-Fachberichte **202**, 1989.

de Kleer, J.: *Choices Without Backtracking,* AAAI-84, 79–85, 1984.

de Kleer, J.: *An Assumption-Based TMS,* AI Journal **28**, 127–162, 1986.

Doyle, J.: *A Truth Maintenance System,* AI Journal **12**, 231–272, 1979.

Genesereth, M. and Nilsson, N.: *Logical Foundations of Artificial Intelligence,* Chap. 6: *Non-Monotonic Reasoning,* Morgan Kaufmann, 1987.

Goodwin, J.: *An Improved Algorithm for Non-Monotonic Dependency Net Update,* LITH-MAT-R-82-23, Linköping University, 1982.

McAllester, D.: *An Outlook on Truth Maintenance Systems,* AI Memo **551**, MIT, 1980.

Puppe, F.: *Belief Revision in Diagnostics,* GWAI-87, Springer Informatik-Fachberichte **152**, 175–184, 1987.

## Chap. 10: Temporal Reasoning

Allen, J.: *Maintaining Knowledge about Temporal Intervals,* CACM 26, No. 11, 832–843, 1983.

Charniak, E. and McDermott, D.: *Introduction to Artificial Intelligence,* Sect. 7.4: *Reasoning Involving Time,* Addison-Wesley, 1985.

Dean, T. and McDermott, D.: *Temporal Data Base Management,* AI Journal **32**, 1–57, 1987.

Fagan, L., Kunz, J., Feigenbaum, E. and Osborn, J.: *Extensions to the Rule-Based Formalism for a Monitoring Task,* in Buchanan, B. and Shortliffe, E. (eds.): *Rule-Based Expert Systems,* Chap. 22, Addison-Wesley, 1984.

Puppe, F.: *Diagnostic Problem Solving with Expert Systems,* Sect. 4.2.10: *Evaluation of Subsequent Sessions* (in German), Springer Informatik-Fachberichte **148**, 1987.

Vilain, M.: *A System for Reasoning about Time,* AAAI-82, 197–201, 1982.

## Chap. 11: Previous Approaches to Problem Classification

AI Journal 84, *Special Volume on Qualitative Reasoning about Physical Systems,* AI Journal **24**, 1984.

Buchanan, B. and Smith, R.: *Fundamentals of Expert Systems* in Barr, A., Cohen, P. and Feigenbaum P. (eds.): *Handbook of Artificial Intelligence,* Vol. IV, Chap. XVIII, 149–191, Addison-Wesley 1989.

Breuker, J. et al.: *Model-Driven Knowledge Acquisition: Interpretation Models,* Memo 87 of the Esprit Project 1098, University of Amsterdam, 1987.

Breuker, J. and Wielinga, B.: *Models of Expertise in Knowledge Acquisition,* in Guida, G. and Tasso, C. (eds.): *Topics in Expert System Design, Methologies and Tools,* North Holland, 265–295, 1989.

Chandrasekaran, B.: *Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design,* IEEE Expert **1** (3), 23–30, 1986.

Chandrasekaran, B.: *Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks,* IJCAI-87, 1183–1192, 1987.

Clancey, W.: *Heuristic Classification,* AI Journal **27**, 289–350, 1985.

ESS: *Choosing the Right Tool for Your Task,* Expert System Strategies **4**, No. 12, 1–7, 1988.

Hayes-Roth, F., Waterman, D. and Lenat, D. (eds.): *Building Expert Systems,* Addison-Wesley, 1983.

Jackson, P.: *Introduction to Expert Systems,* Addison-Wesley, 1986 (2nd edition 1990).

Karbach, W., Linster, M. and Voss, A.: *Model-Based Approaches, one Label – one Idea?* EKAW-90 (European Workshop on Knowledge Acquisition), 1990.

Klinker, G.: *KNACK: Sample-Driven Knowledge Acquisition for Reporting Systems,* in [Marcus 88], 125–174, 1988.

Marcus, S. (ed.): *Automating Knowledge Acquisition for Expert Systems,* Kluwer Academic Publishers, 1988.

McDermott, J.: *Preliminary Steps Toward a Taxonomy of Problem-Solving Methods,* in [Marcus 88], 225–256, 1988.

Offutt, D.: *SIZZLE: A Knowledge-Acquisition Tool Specialized for the Sizing Task,* in [Marcus 88], 175–200, 1988.

Stefik, M. et al.: *The Organisation of Expert Systems – a Tutorial,* AI Journal **18**, 135–173, 1982.

Stoyan, H.: *Programming Methods of Artificial Intelligence* (in German) Vol. 1 + 2, Springer, 1988 + 1991.

Wielinga, B., Schreiber, A. and Breuker, J.: *KADS – A Modelling Approach ti Knowledge Engineering,* in: Knowledge Acquisition – Special Issue on KADS, 1992.

## Chap. 12: Principles of Problem-Solving Methods

Gappa, U.: *A Toolbox for Generating Graphical Knowledge Acquisition Environments,* Proc. of the first World Congress on Expert Systems, Vol. 2, 787–810, Pergammon Press, 1991.

de Groote, A.: *Thought and Choice in Chess,* Basic Books, New York, 1965.

Meyer, B.: *Object Oriented Software Construction,* Prentice Hall, 1988.

Newell, A. and Simon, H.: *GPS, a Program that Simulates Human Thought,* reproduced in Feigenbaum, E. and Feldmann, J.: *Computers and Thought,* McGraw-Hill, 279–293, 1972 (1963).

## Chap. 13: Survey of the Problem-Solving Type Classification

Achtstätter, J.: *Diagnostic Expert Systems in Circuit Board Production* (in German), in [Puppe 89], 3.28–3.29, 1989.

Aikins, J., Kunz, J., Shortliffe, E. and Fallat, R: *PUFF: An Expert System for Interpretation of Pulmonary Function Data,* Computers in Biomedical Research, **16**, 199–208, 1983, also in Clancey, W. and Shortliffe, E. (eds.): *Readings in Medical Artificial Intelligence,* Chap. 19, Addison-Wesley, 1984.

Borrmann, H.-P. and Winklhofer, F.: *Expert System for Early Damage Detection in Turbomachinery* (in German), Qualität und Zuverlässigkeit, **33**, 139–141, 1988.

Dickmann, H.-J.: *Tuning of Data Bank Applications* (in German), Praxis der Informationsverarbeitung und Kommunikation, **9**, 122-126, 1986.

Elstein, A., Shulman, L. and Sprafka, S.: *Problem Solving,* Harvard University Press, 1978.

Ernst, G.: *Expert Systems in the Motor Test Bed - Application Report* in Bläsing, J. (ed.): Practical Handbook of Quality Control, Vol. 4 (in German), Baustein F2, GMFT, 1988.

Fagan, L., Kunz, J., Feigenbaum, E. and Osborn, J.: *Extensions to the Rule-Based Formalism for a Monitoring Task,* in in Buchanan, B. and Shortliffe, E. (eds.): *Rule-Based Expert Systems,* Chap. 22, Addison-Wesley, 1984.

Gasching, J.: *PROSPECTOR: an Expert System for Mineral Exploration,* in Michie, D. (ed.): *Introductionary Readings in Expert Systems,* Gordon and Breech, 1982.

Keim, M. and Nordmann, R.: *Application of an Expert System for Diagnosis of Rotordynamic Problems in Turbomachinery,* Proceedings of the American Society of Mechanical Engineering (ASME), Design Technical Conference on Mechanical Vibrations and Noise (DE-VOL-18/5), 85–91, 1989.

Mertens, P., Borkowski, V. and Geis, W.: *Industrial Expert System Applications - a Compendium* (in German), Springer, 2nd edition, 1990.

Miller, R., Pople, H. and Myers, J.: *INTERNIST1, an Experimental Computer-Based Diagnostic Consultant for General Internal Medicine,* New England Journal of Medicine **307**, 468–476, 1982.

Miller, R., McNeil, M., Challinaor, S., Masari, F. and Myers, J.: *The INTERNIST1/ Quick Medical Reference Project - a Status Report,* West J. Med. **145**, 816–822, 1986.

Mütter, S.: *FAKTA - a System for the Construction and Evaluation of Diagnostic Case Data Banks and its Application to the Example of an Advising System for Data Banks* (in German), Diploma Thesis, University of Karlsruhe, 1989.

Nedeß, C. and Plog, J.: *EFFEKT - Diagnostic Expert System for the Spray Casting of Elastomers* (in German), Kunststoffe **78**, No. 12, 1988.

Neis, B.: *PIDAS - Development and Evaluation of an Expert System for the Identification of Fungi* (in German), Diploma Thesis, University of Kaiserslautern, 1988.

Nossem, B.: *Multiple Sclerosis Expert Systems* (in German), Dissertation, University of Erlangen-Nürnberg, 1989.

Plog, J.: *Expert System Supported Quality Control in the Production of Elastomer Particles* (in German), Fortschrittsberichte VDI, Reihe 2: Fertigungstechnik, No. 200, VDI-Verlag, 1990.

Premauer, T.: *IXMO - an Expert System for Motor Diagnostics in the Automobile Industry* (in German), INTERKAMA-86, Fachberichte für Messen, Steuern, Regeln **14**, 532–539, Springer, 1986.

Puppe, F.: *Experience from three Application Projects with MED1* (in German), GI-Kongreß Wissensbasierte Systeme, Springer Informatik-Fachberichte **112**, 234–245, 1985 (a).

Puppe, B. and Puppe, F: *MED1 - an Intelligent Computer Program for Thoracic Pain Diagnosis,* Klinische Wochenschrift **63**, 511–517, 1985 (b).

Puppe, F.: *Diagnostic Problem Solving with Expert Systems* (in German), Springer Informatik-Fachberichte **148**, 1987.

Puppe, F. and Voss, H.: *Abstracts on Diagnostic Expert Systems, Collected Contributions to the 1st and 2nd Workshops Diagnostic Expert Systems* (in German), Arbeitspapiere der GMD **375**, 1989.

Puppe, F., Legleitner, T. and Huber, K.: *DAX/MED2 - A Diagnostic Expert System for Quality Assurance of an Automatic Transmission Control Unit,* in Zarri, G. (ed.): *Operational Expert Systems in Europe,* Pergammon Press, 1991.

Schewe, S., Herzer, P. and Krüger, K.: *Prospective Application of an Expert System for the Medical History of Joint Pain,* Klinische Wochenschrift, 1990.

Shortliffe, E., *Computer-Based Medical Consultations: MYCIN,* American Elsevier, 1976.

Trendelenburg, Chr. and Wieland, H.: *Routine Use of a Clinical Chemistry Expert System with a Knowledge Base on Disorders in Lipoprotein Metabolism,* Biochim. Clin. **10**, 928–929, 1986.

Walser, P.: *Formalization of Heuristic, Causal, Case-Based and General Knowledge for a Tutorial Diagnostic Expert System Exemplified with the Automobile Motor Domain* (in German), Diploma Thesis, University of Karlsruhe, Institute of Logic, 1991.

## Chap. 14: Simple Classification

Fairley, R.:*Software Engineering Concepts,* McGraw-Hill, 1985.

## Chap. 15: Heuristic Classification

Boose, J.: *Personal Construct Theory and the Transfer of Human Expertise,* AAAI-84, 27–33, 1984.

Boose, J. and Bradshaw, J.: *Expertise Transfer and Complex problems: Using AQUINAS as a Knowledge-Acquisition Workbench for Knowledge-Based Systems,* International Journal of Man-Machine Studies **26**, 3–28, 1987.

Chandrasekaran, B. and Mittal, S.: *Conceptual Representation of Medical Knowledge for Diagnosis by Computer: MDX and Related Systems,* Advances in Computers **22**, 217–293, 1983.

Clancey, W.: *Classification Problem Solving,* AAAI-84, 49–55, 1984.

ESS: *Two Problem-Specific Tools for Diagnosis,* Expert System Strategies **4**, 7–12, 1988.

Gappa, U.: *CLASSIKA, a Knowledge Acquisition Tool for Use by Experts,* Proceedings of the AAAI Workshop on Knowledge Acquisition, Banff, Canada, 1989.

Puppe, F. and Puppe, B.: *Overview on MED1: an Heuristic Diagnostics Expert System with an Efficient Control Structure,* GWAI-3, Springer Informatik-Fachberichte **76**, 11–20, 1983.

Puppe, F.: *Diagnostic Problem Solving with Expert Systems* (in German), Springer Informatik-Fachberichte **148**, 1987 (a).

Puppe, F.: *Requirements for a Classification Expert System Shell and their Realization in MED2,* Applied Artificial Intelligence **1**, 163–171, 1987 (b).

Puppe, F. and Gappa, U.: *Towards Knowledge Acquisition by Experts,* Proc. of 5th IEA/AIE (Industrial and Engineering Applications of Artificial Intelligence and Expert Systems), Springer, Lecture Notes in Artificial Intelligence **604**, 546–555, 1992.

Van Melle, W.: *System Aids in Constructing Consultation Programs,* UMI Research Press, 1981.

Walser, P.: *Formalization of Heuristic, Causal, Case-Based and General Knowledge for a Tutorial Diagnostic Expert System Exemplified with the Automobile Motor Domain* (in German), Diploma Thesis, University of Karlsruhe, Institute of Logic, 1991.

Weis, S. and Kulikowski, C.: *A practical Guide to Designing Expert Systems,* Roman & Allanheld, 1984.

## Chap. 16: Heuristic Classification: Additional Mechanisms

de Kleer, J.: *An Assumption Based TMS,* AI Journal **28**, 127–162, 1986.

Pearl, J.: *How to Do with Probabilities What People Say You Can't,* Technical Report CSD 85003, University of California, and Proceedings of the 2nd Conference on Artificial Intelligence Applications, 1985.

Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference,* Morgan Kaufmann, 1988.

Puppe, F.: *Diagnostic Problem Solving with Expert Systems* (in German), Springer Informatik-Fachberichte **148**, 1987.

## Chap. 17: Set-Covering Classification

Eshelman, L.: *MOLE: A Knowledge Acquisition Tool for Cover-and-Differentiate Systems,* in Marcus, S. (ed.): *Automating Knowledge Acquisition for Expert Systems,* Chap. 3, Kluwer Academic Publishers, 1988.

Goos, K.: *FEMO: an Expert System Framework for Accommodating Causal Fault Models* (in German), Diploma Thesis, University of Karlsruhe, 1989.

Goos, K.: *Feedback in a model-based diagnostics shell* (in German), Proceedings of GWAI-90, Springer Informatik-Fachberichte **251**, 1990.

Kahn, G.: *MORE - From Observing Knowledge Engineers to Automating Knowledge Acquisition,* in Marcus, S. (ed.): *Automating Knowledge Acquisition for Expert Systems,* Chap. 2, Kluwer Academic Publishers, 1988.

Long, W., Naimi, S., Criscitiello, M. and Jayes, R.: *The Development and Use of a Causal Model for Reasoning about Heart Failure,* in Miller, P. (ed.): *Selected Topics in Medical Artificial Intelligence,* Chap. 4, Springer, 1988. Miller, P. and Fisher, P.: *Causal Models for Medical Artificial Intelligence,* in Miller, P. (ed.): *Selected Topics in Medical Artificial Intelligence,* Chap. 2, Springer, 1988.

Patil, R.: *Causal Representation of Patient Illness for Electrolyte and Acid-Base Diagnosis,* LCS-TR-267, MIT, 1981.

Patil, R., Szolovits, P. and Schwartz, W.: *Modelling Knowledge of the Patient in Acid-Base and Electrolyte Disorders,* in Szolovits, P. (ed.): *Artificial Intelligence in Medicine,* AAAS Selected Symposium **51**, Westview Press, 1982.

Patil, R.: and Senyk, O.: *Compiling Causal Knowledge for Diagnostic Reasoning,* in Miller, P. (ed.): *Selected Topics in Medical Artificial Intelligence,* Chap. 3, Springer, 1988.

Peng, Y. and Reggia, J.: *Abductive Inference Models for Diagnostic Problem Solving,* Springer, New York, 1990.

Reggia, J., Nau, D. and Wang, P.: *A New Inference Method for Frame-Based Expert Systems,* AAAI-83, 333–337, 1983 (a).

Reggia, J., Nau, D. and Wang, P.: *Diagnostic Expert Systems Based on a Set Covering Model,* Int. J. Man-Machine Studies **19**, 437–460, 1983 (b).

Steels, L.: *Second Generation Expert Systems,* Future Generation Computing Systems **1**, 213–221, 1985.

Zadeh, L. A.: *Outline of a New Approach to the Analysis of Complex Systems and Decision Processes,* IEEE Trans. on Systems, Man and Cybernetics **3**, 28–44, 1973.

subsection*Chap. 18: Functional Classification

AI-Journal 84, *Special Volume on Qualitative Reasoning about Physical Systems,* AI Journal **24**, 1984.

Davis, R.: *Diagnostic Reasoning Based on Structure and Function,* AI Journal **24**, 347–411, 1984.

de Kleer, J.: *An Assumption-Based TMS,* AI Journal **28**, 127–162, 1986.

de Kleer, J. and Williams, B.: *Diagnosing Multiple Faults,* AI Journal **32**, 97–130, 1987.

de Kleer, J. and Williams, B.: *Diagnosis with Behavioral Models,* IJCAI-89, 1324–1330, 1989.

Hamscher, W. and Davis, R.: *Issues in Model-Based Troubleshooting,* AI Memo **893**, MIT, 1987.

Hnyk, R.: *Design and Implementation of an Expert System Shell for Qualitative Simulation on the Example of the Automobile Motor* (in German), Diploma Thesis, University of Karlsruhe, 1988.

Reiter, R.: *A Theory of Diagnosis from First Principles,* AI Journal **32**, 57–96, 1987.

Struss, P. and Dressler, O.: *"Physical Negation" – Integrating Fault Models into the General Diagnostic Engine,* IJCAI-89, 1318–1323, 1989.

## Chap. 19: Statistical Classification

Charniak, E. and McDermott, D.: *Introduction to Artificial Intelligence,* Sect. 8.2: *Statistics in Abduction,* Addison-Wesley, 1985.

Cooper, G.: *Expert Systems Based on Belief Networks - Current Research Directions,* Memo KSL-87-51, Stanford University, 1987.

de Dombal, F., Leaper, D., Horrocks, J., Staniland, J. and McCann, A.: *Computer-Aided Diagnosis of Acute Abdominal Pain,* British Med. Journal **2**, 9–13, 1972.

Gasching, J.: *PROSPECTOR: an Expert System for Mineral Exploration,* in Michie, D. (ed.): *Introductionary Readings in Expert Systems,* Gordon and Breech, 1982.

Lusted, L.: *Introduction to Medical Decision Making,* Thomas Books, 1968.

Mütter, S.: *FAKTA - a System for the Construction and Evaluation of Diagnostic Case Data Banks and its Application to the Example of an Advising System for Data Banks* (in German), Diploma Thesis, University of Karlsruhe, 1989.

Pearl, J.: *How to Do with Probabilities What People Say You Can't,* Technical Report CSD 85003, University of California, and Proceedings of the 2nd Conference on Artificial Intelligence Applications, 1985.

Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference,* Morgan Kaufmann, 1988.

Szolovits, P. and Pauker, S.: *Categorical and Probabilistic Reasoning in Medical Diagnosis,* AI Journal **11**, 115–144, 1978.

## Chap. 20: Case-Comparing Classification

Fries, J.: *Time-Oriented Patient Records and a Computer Data Bank,* Journal of the Medical Association **222**, 1536–1542, 1972.

Hestermann, C.: *CcC: Design and Implementation of an Expert System Component for Case Comparison and its Application to the Example of Similarity Search with Prismatic Workpieces* (in German), Diploma Thesis, University of Karlsruhe, 1990.

Kolodner, J. and Kolodner, R.: *Using Experience in Clinical Problem Solving: Introduction and Framework,* GIT-ICS-85/21, Georgia Institute of Technology, 1985.

Koton, P.: *Using Experience in Learning and Problem Solving,* Dissertation, Lab. for Computer Science, MIT, 1988.

Porter, B., Barciss, R. and Holte, R.: *Concept Learning and Heuristic Classification in Weak-Theory Domains,* AI Journal **45**, 229–263, 1990.

Puppe, F. and Goos, K.: *Improving Case-Based Classification with Expert Knowledge,* Proc. GWAI-91, Springer Informatik-Fachberichte **285**, 196–205, 1991.

## Chap. 21: Survey of the Problem-Solving Type Construction

Barker, V. and O'Connor, D.: *Expert Systems for Configuration at Digital: XCON and Beyond,* CACM **32**, 298–317, 1989.

Barr, A. and Feigenbaum, E.: *The Handbook of Artificial Intelligence,* Vol. 1, Morgan Kaufmann, 1981

Dyckhoff, T.: *ExAP: an Approach to an Expert System Tool for Work Scheduling in the Machining of Prismatic Parts* (in German), Diploma Thesis, University of Karlsruhe, 1989.

Fox, M.: *Constrained Directed Search: A Case-Study of Job Scheduling,* Dissertation, Carnegie Mellon University, Computer Science Department, 1983.

Friedland, P.: *Knowledge-Based Experiment Design in Molecular Genetics,* Dissertation, Memo HPP-79-29, Stanford University, 1979.

Friedland, P. and Iwasaki, Y.: *The Concepts and Implementation of Skeletal Plans,* Journal of Automated Reasoning **1**, 161–208, 1985.

Hickam, D. et al.: *The Treatment Advice of a Computer-Based Cancer Chemotherapy Protocol Advisor,* Annals of Internal Medicine **103**, 928–936, 1985.

Marcus, S.: *SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems,* in Marcus, S. (ed.): *Automating Knowledge Acquisition for Expert Systems,* Chap. 4, Kluwer Academic Publishers, 1988.

McDermott, J.: *R1: A Rule-Based Configurer of Computer Systems,* AI Journal **19**, 39–88, 1982.

McDermott, J. and Bachant, J.: *R1 Revisited: Four Years in the Trenches,* AI Magazine **5**, 21–32, 1984.

Musen, M. et al.: *OPAL: Use of a Domain Model to Drive an Interactive Knowledge-Editing Tool,* International Journal of Man-Machine Studies **26**, 105–121, 1987.

Nilsson, N.: *Principles of Artificial Intelligence,* Springer, 1982.

Poeck, K.: *Conception and Implementation of a Problem-Specific Expert System Tool for Assignment with the Propose-and-Exchange Strategy* (in German), Diploma Thesis, University of Karlsruhe, 1990.

Sacerdoti, E.: *A Structure for Plans and Behavior,* Elsevier/North Holland, 1977.

Smith, S., Fox, M. and Ow, P.: *Constructing and Maintaining Detailed Production Plans: Investigation into the Development of Knowledge-Based Factory Scheduling Systems,* AI Magazine **7**, 45–61, 1986.

Stefik, M.: *Planning with Constraints (MOLGEN: Part I),* AI Journal **16**, 111–139, 1981.

## Chap. 22: Skeletal Construction

Friedewald, A.: *Functional Integration of Work Planning and Materials Construction for Polymer Parts* (in German), Dissertation manuscript, Technical University Hamburg-Harburg, 1990.

Friedland, P.: *Knowledge-Based Experiment Design in Molecular Genetics,* Dissertation, Memo HPP-79-29, Stanford University, 1979.

Friedland, P. and Iwasaki, Y.: *The Concepts and Implementation of Skeletal Plans,* Journal of Automated Reasoning **1**, 161–208, 1985.

Lindsay, R., Buchanan, B., Feigenbaum, E. and Lederberg, J.: *Application of Artificial Intelligence for Organic Chemistry: the DENDRAL Project,* McGraw-Hill, 1980.

Nedeß, C., Friedewald, A. and Plog, J.: *Diagnostic Strategy: Solution Approach also for Configuration Problems on the Example of the Construction of Spray casting Tools* in: *Design of CIM-capable Firms* (in German), Research Report **1** of the Hochschulgruppe Arbeits- und Betriebsorganisation, GFMT, 1988.

Nilsson, N.: *Principles of Artificial Intelligence,* Springer, 1982.

## Chap. 23: Propose and Revise

Hartinger, M. and Wedel, T.: *Stand des Expertensystems PAREX-CO zur wissensbasierten dynamischen Konfiguration von Parametern des PPS-Modularprogramms COPICS,* Memo 1/1990, University of Erlangen-Nuremberg, Department of Business Computer Science, 1990.

Marcus, S.: *SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems,* in Marcus, S. (ed.): *Automating Knowledge Acquisition for Expert Systems,* Chap. 4, Kluwer Academic Publishers, 1988 (a).

Marcus, S., Stout, J. and McDermott, J.: *VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking,* AI Magazine **9**, 95–111, 1988 (b).

Marcus, S. and McDermott, J.: *SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems,* AI Journal **39**, 1–38, 1989.

## Chap. 24: Propose and Exchange

Ernst, T.: *Prototyping for a Knowledge-Based System for Test Planung,* Diploma Thesis, University of Kaiserslautern, 1990.

Marcus, S.: *SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems,* in Marcus, S. (ed.): *Automating Knowledge Acquisition for Expert Systems,* Chap. 4, Kluwer Academic Publishers, 1988.

Poeck, K.: *Conception and Implementation of a Problem-Specific Expert System Tool for Assignment with the Propose-and-Exchange Strategy* (in German), Diploma Thesis, University of Karlsruhe, 1990.

## Chap. 25: Least-Commitment Strategy

Descotte, Y. and Latombe, J.: *Making Compromises among Antagonist Constraints in a Planner,* AI Journal **27**, 183–217, 1985.

Dyckhoff, T.: *ExAP: an Approach to an Expert System Tool for Work Scheduling in the Machining of Prismatic Parts* (in German), Diploma Thesis, University of Karlsruhe, 1989.

Stefik, M.: *Planning with Constraints (MOLGEN: Part I),* AI Journal **16**, 111–139, 1981.

## Chap. 26: Model-Based Planning

Cohen, P. and Feigenbaum, E.: *The Handbook of Artificial Intelligence* Vol. 3, Chap. XV: *Planning and Problem Solving,* 1982.

Hertzberg, J.: *Planning Methods of Artificial Intelligence* (in German), Informatik-Spektrum **9**, 149–161, 1986.

Hertzberg, J.: *Planning: Introduction to the Planning Methods of Artificial Intelligence* (in German), BI, 1989.

Ginsberg, M. and Smith, D.: *Reasoning about Action I: A Possible World Approach,* AI Journal **35**, 165–195, 1988.

Nilsson, N.: *Principles of Artificial Intelligence,* Springer, 1982.

Sacerdoti, E.: *A Structure for Plans and Behavior,* Elsevier/North Holland, 1977.

Wilkins, D.: *Domain Independent Planning: Representation and Plan Generation,* AI Journal **22**, 269–302, 1984.

Wilkins, D.: *Practical Planning. Extending the Classical AI Planning Paradigm,* Morgan Kaufmann, 1988.

## Chap. 27: Case-Comparing Construction

Hammond, K.: *Case-Based Planning: An Integrated Theory of Planning, Learning and Memory,* Dissertation, Yale University, YALE-CSD-RR-448, 1986.

Hammond, K.: *Explaining and Repairing Plans that Fail,* IJCAI-87, 109–114, 1987.

Hammond, K.: *Case-Based Planning: Viewing Planning as a Memory Task,* Academic Press, 1989.

Kolodner, J.: *Extending Problem Solving Capabilities through Case-Based Inference,* Proceedings of the International Machine Learning Workshop, 1987, and in Kolodner, J. (ed.): *Case-Based Inference, a Collection of Papers,* Georgia Institute of Technology, GIT-ICS-87/18, 1987 (a).

Kolodner, J.: *Capitalizing on Failure through Case-Based Inference,* Proceedings of the Conference of the Cognitive Science Society, and in Kolodner, J. (ed.): *Case-Based Inference, a Collection of Papers,* Georgia Institute of Technology, GIT-ICS-87/18, 1987 (b).

Simpson, R.: *A Computer Model of Case-Based Reasoning in Problem Solving,* Dissertation, Georgia Institute of Technology, GIT-ICS-85/18, 1985.

## Chap. 28: Partial Integration of Construction Methods

Cunis, R., Günter, A., Syska, I., Bode, H. and Peters, H.: *PLAKON – An Approach to Domain-Independent Construction,* in Proc. 2nd IEA/AIE, Tenesse, 1989.

Cunis, R., Günter, R. and Strecker H. (eds.): *The PLAKON Book: An Expert System Kernel for Planning and Configuration Tasks in Technical Domains* (in German), Springer Informatik-Fachberichte **266**, 1991.

## Chap. 29: Review of the Problem-Solving Type Simulation

Bratley, P., Fox, B. and Schrage, L.: *A Guide to Simulation,* Springer, 1987.

Griffin, R.: *Choosing and Using a Simulation System,* in Hurison, R. (ed.): *Simulation,* 261–270, Springer, 1986.

Round, A.: *Knowledge-based Simulation,* in Barr, A., Cohen, P. and Feigenbaum, E. (eds.): *Handbook of Artificial Intelligence,* Vol. IV, Chap. XXII, 415–518, Addison-Wesley, 1989.

## Chap. 30: Single-Phase Simulation

Brown, J., Burton, R. and de Kleer, J.: *Pedagogical Natural Language and Knowledge Engineering Techniques in SOPHIE I, II and III,* in Sleeman, D. and Brown, J. (eds.): *Intelligent Tutoring Systems,* 227–282, Academic Press, 1982.

Stallman, R. and Sussman, G.: *Forward Reasoning and Dependency Directed Backtracking in a System for Computer-Aided Circuit Analysis,* AI Journal **9**, 135–196, 1977.

## Chap. 31: Numerical Multiple-Phase Simulation

Becker, D.: *An Object-Oriented Simulation System with Parameter-Based Network Modelling for Production Processes with Piece-Goods Character* (in German), Dissertation, FhG-IPA, University of Stuttgart, 1990.

Round, A.: *Knowledge-based Simulation,* in Barr, A., Cohen, P. and Feigenbaum, E. (eds.): *Handbook of Artificial Intelligence,* Vol. IV, Chap. XXII, 415–518, Addison-Wesley, 1989.

## Chap. 32: Qualitative Multiple-Phase Simulation

AI-Journal 84, *Special Volume on Qualitative Reasoning about Physical Systems,* AI Journal **24**, 1984.

AI-Journal 91, *Special Volume on Qualitative Reasoning about Physical Systems II,* AI Journal **51**, 1991.

Bredeweg, B. and Wielinga, B.: *Integrating Qualitative Reasoning Approaches,* ECAI-88, 195–201, 1988.

Davis, E.: *Order of Magnitude Reasoning in Qualitative Differential Equations,* New York University Technical Report **312**, 1987.

de Kleer, J.: *How Circuits Work,* AI Journal **24**, 205–280, 1984.

Forbus, K.: *Qualitative Process Theory,* AI Journal **24**, 85–168, 1984.

Kuipers, B.: *Common Sense Reasoning about Causality: Deriving Behaviour from Structure,* AI Journal **24**, 169–203, 1984.

Kuipers, B. and Chin, C.: *Taming Intractable Branching in Qualitative Simulation,* IJCAI-87, 1079–1085, 1987.

Iwasaki, Y.: *Qualitative Physics,* in Barr, A., Cohen, P. and Feigenbaum, E. (eds.): *Handbook of Artificial Intelligence,* Vol. IV, Chap. XXI, 323–414, Addison-Wesley, 1989.

Sacks, E.: *Automatic Qualitative Analysis of Ordinary Differential Equations Using Piecewise Linear Approximation,* Dissertation, MIT, Lab. of Computer Science, Technical Report **416**, 1988.

Sacks, E.: *A Dynamic Systems Perspective on Qualitative Simulation,* AI Journal **42**, 349–362, 1990.

Struß, P.: *Qualitative Reasoning,* in Proc. KIFS-89, Springer Informatik-Fachberichte **203**, 224–259, 1989 (a).

Struß, P.: *Structuring of Models and Reasoning about Quantities in Qualitative Physics,* Dissertation, University of Kaiserslautern, 1989 (b).

Voß, H.: *Representing and Analysing Causal, Temporal and Hierarchical Relations of Devices,* Dissertation, University of Kaiserslautern, 1986.

## Chap. 34: Integration of Classification Methods

Blum, R.: *Discovery, Confirmation and Incorporation of Causal Relationships from a Large Time-Oriented Clinical Data Base: The RX Project,* in Clancey, W. and Shortliffe, E. (eds.): *Readings in Medical Artificial Intelligence: the First Decade,* Chap. 17, Addison-Wesley, 1984.

Blum, R.: *Modeling and Encoding Clinical Causal Relationships in a Medical Knowledge Base,* in Miller, P. (ed.): *Selected Topics in Medical Artificial Intelligence,* Chap. 9, Springer, 1988.

Chandrasekaran, B. and Mittal, S.: *Deep versus Compiled Knowledge Approaches to Diagnostic Problem Solving,* AAAI-82, 349–354, 1982.

de Dombal, F., Leaper, D., Horrocks, J., Staniland, J. and McCann, A.: *Computer-Aided Diagnosis of Acute Abdominal Pain,* British Med. Journal **2**, 9–13, 1972.

de Jong, G. and Mooney, R.: *Explanation Based Learning: An Alternative View,* Machine Learning **1**, 145–176, 1986.

Eshelman, L.: *MOLE: A Knowledge Acquisition Tool for Cover-and-Differentiate Systems,* in Marcus, S. (ed.): *Automating Knowledge Acquisition for Expert Systems,* Chap. 3, Kluwer Academic Publishers, 1988.

Kahn, G.: *MORE - From Observing Knowledge Engineers to Automating Knowledge Acquisition,* in Marcus, S. (ed.): *Automating Knowledge Acquisition for Expert Systems,* Chap. 2, Kluwer Academic Publishers, 1988.

Kobsa, A. and Wahlster, W. (eds.): *User Models in Dialog Systems,* Springer, 1990.

Koton, P.: *Using Experience in Learning and Problem Solving,* Dissertation, Lab. for Computer Science, MIT, 1988.

Michalski, R. and Chilausky, R.: *Learning by Being Told and Learning from Examples: an Experimental Comparison of Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soyabean Disease Diagnosis,* Policy Anal. and Inf. Systems **4**, 125–160, 1980.

Steels, L.: *Second Generation Expert Systems,* Future Generation Computing Systems **1**, 213–221, 1985.

# System Index


ABEL 49, 165
ABSTRIPS 91
Allen's temporal calculus 83ff
AQUINAS 140
ARAMIS 200
ART 10, 151
ATMS 72, 78

BACK 47

C 50
C++ 50
CASEY 193, 200, 201, 312
CASNET 67
CcC 201ff, 320
CcC+ 201
CHEF 254
CLASIKA 11, 109, 140ff
CLOS 50
COKE 235ff
CYC 290

DABAS 119, 188
DAX 116
de Dombal's system 185
DENDRAL 7, 91, 93, 121
DIACONS 118

EFFEKT 116
EL 51, 56, 91, 270
EMYCIN 8, 38, 91, 140
ETS 140
ExAP 201, 208, 211, 212, 244ff, 320
EXPERT 140

FAKTA 188ff, 201
FEMO 167ff
FRL 44ff, 50

GA1 91
GARI 244, 245
GDE 179
GDE+ 179
General Problem Solver 6, 101
GRUNDY 119

HARPY 91
HASP 93
HEARSAY II 8, 91
HTE 95, 178, 179

INGRES 45, 50, 190
INTERNIST 8, 58, 61, 64ff, 118, 132, 140, 155, 185, 324
ISIS 208
IXMO 116

JULIA 254

KEE 10, 151, 276
KL-ONE 44, 47ff, 50
KL-TWO 47
KNACK 99
KRYPTON 47, 49

LADIS 117
LEDIS 116
LISP 11, 43ff, 50, 75
Long's system 165

MACSYMA 7
MDX 140
MED1 58, 61, 67, 116, 118, 140, 324
MED2 11, 38, 70, 80ff, 85, 116, 140, 148, 152, 180, 188, 201, 222, 228
MEDIATOR 254
MESON 47
MODEX 117, 140

MOLE 99, 166, 312
MOLGEN 91, 208, 210ff, 221, 244, 252
MORE 99, 166, 291, 312
MSX 118
MYCIN 7, 8, 41, 42, 45, 58, 61, 66ff, 81, 91, 93, 118, 132, 140, 150, 185, 302, 324

NIKL 47, 49
NOAH 91, 208, 211, 212, 250

ONCOCIN 208, 211
OPAL 11, 208, 211
OPS5 10, 38, 40, 42, 91, 227

PAREX-CO 228
PIDIAS 119
PIP 140
PLAKON 221, 261
PLANT/DS 312
PRO.M.D 118
PROLOG 32ff, 38, 42, 45, 72, 250
PROSPECTOR 8, 91, 120, 150, 187
PUFF 8, 118

QMR 8, 118

R1 41, 42, 91
R1-XCON 18, 100, 208, 211, 212
REST 207, 211, 212, 236ff
RETE 38, 41, 91
RULEMASTER 312
RX 312

SALT 11, 99, 208, 211, 212, 214, 227, 235
Schewe's system 118
SHERLOCK 179
SIMKIT 276
SIMPLE 276, 277
SIMUL 179ff
SIPE 250
SIUX 117
SIZZLE 99
SMALLTALK 45, 50
SOPHIE 270
SPEX 221
SYN 91

TADIS 117
TESTBENCH 140
TMM 82ff
TMS 72, 75
TUMAD 117

VADIS 117
VAX Decision Expert 130
VM 80ff, 85, 86, 91, 118
VT 208, 211, 212, 227

XPS advising 119

# Subject Index


A (assertion) box 47, 49
a priori probabilities 183ff
abnormality 193, 197
abstract data types 46
abstraction 132, 213, 249, 251
– level of 194, 198
agenda 37, 102, 106, **327**
– solution 136
– test 136
AKO 44
Allen algorithm 56
AND/OR graphs 214, 216, 221, 222, 253, 255, 256
assessment 95
– multiple 121
– simple 120
assignment 209, **330**
ATMS 76ff, 151, 179, 324, **328**
attached procedures 11, 43, 323

backtracking 72, 94
Bayes' Theorem 58ff, 125, 184, 185, 188, 189, 324
– variants 186ff
Bayesian networks 70, 150, 186, 187
belief revision 72, 91, 153, 122,324
blackboard 91, 299
Blocks World 208, 211, 250
bottom-up aggregation 255
Busacker-Gowen algorithm 235, 236

calculus, adequacy 31
calculus
– completeness 30
– consistency 31
– decidability 30
– efficiency 31
– Horn clausal 32
– propositional 29, 30
– resolution 31
– soundness 30
cancer therapy 208, 211
causal models 68, 161ff
– consistency mode 161, 163
– diagnostic mode 161, 164
– simulation mode 161, 162
certainty factors 131
chess 106
classification 90, 91, 103, 115ff, **328**
– applications 118ff
– automatic 47ff
– case-based 125, 191ff, **329**
– causal **329**
– domains 115ff
– functional 125, 170ff, **329**
– heuristic 49, 90ff, 95, 124, 222, 228, 131ff, **328**
  additional mechanisms 149ff
– integration of methods 295ff
  data collection 310ff
  explainability 306ff
  knowledge acquisition 303ff
  knowledge manipulation 299ff
  knowledge representation 297ff
  knowledge transformation 303
  programming 311ff
– problem characteristics 121ff
– problem types 120ff
– problem-solving methods 123ff
– set-covering 124, 156ff, **328**
– simple 93, 95, 128ff, **328**
– statistical 95, 125, 183ff, **329**
– systematic 95
classifier 48
clause 31

cluster 80, 84
component
- dialog **332**
- explanation 15, 306ff, **332**
  - critique mode 306, 309
  - experimental mode 307
  - justification mode 306, 308
  - manual mode 306, 307
  - pupil mode 307
  - teacher mode 307
- interview 15, **332**
- knowledge acquisition 15, **332**
- problem-solving 14, **332**
configuration 92, 209, 288, 317, **330**
- bottom-up 259
- case-comparing 259
- of computers 208, 211
- of elevators 208, 211, 227
- top-down 259
conflict resolving 322
confluence, local 31
constraints, 11, 51ff, 323, **328**
- network 52, **328**
- problem **328**
- propagation 52, 94, 240
construction 94, 103, 207ff, **329**
- steps 216, 218
- assembly 92
- case-comparing 253ff
- design 92
- domains 207ff
- heuristic 93
- integration 255ff
- interactive 255
- least commitment 240ff
- model-based 249ff
- problem characteristics 209ff
- problem types 209
- problem-solving methods 212ff
- propose and exchange 229ff
- propose-and-revise 223ff
- skeletal 49, 216ff, **330**
- specification 92
control 90, 92
current support strategy 75

data
- abstractions 92, 132, 135
- pre-processing **327**
- parametrized 153
- temporal 152ff
- uncertain 149ff
debugging 90, 94, 97
decision tables 98, 124, 129
decision trees 124, 128ff
default values 44
Dempster-Shafer Theory 58, 61ff, 324
design 90, 97
diabetes 222, 316
diagnoses, multiple 296, **329**
diagnostic intermediate structure 140, **329**
diagnostics, see classification
dialog control 122

establish-refine strategy 123, 133, **329**
evidences 11, 131
evoking strength 64
exceptions 72, 68, 152, 295, 305, 322, 324
expert systems **331**
- architecture of 14ff
- as knowledge media 21ff
- characterization of 3ff
- domains 23ff
- embedded **332**
- history 6ff
- interactive **332**
- tools 8ff, 98, **331**
- use of 17ff
expressive power 322

facets 44
fault models 125, 156
fault-finding 120,140
feedback loops 161, 168
flow diagrams 98
frames 11, 43ff, 323, **328**
- cognitive meaning of 46ff
functional models 177

generate-and-test strategy 91, 93, 221ff, **330**
generic tasks 96
genetics, molecular 208, 221, 244

heart diseases 316
hierarchies 43, 93, 94
hill-climbing 221
hypertext 307

hypothesis generation 164
hypothesize-and-test strategy 102, 124, 133, **329**

identification 92, 121
import value 64
inference strategy **327**
inheritance 11, 43, 323
instruction 90
Integration of problem-solving methods 287ff
– data collection 291
– domains 288
– explainability 291
– knowledge acquisition 290
– levels 289
– object references 289
– programming 292
interpretation 90

JTMS 324, **328**
justification networks 11

knowledge **326**, **331**
– acquisition 9ff, 108ff, **332**
– base 15, **331**
– compiling 91, 312
– engineer 9
– manipulation 108
– representation 11, 107ff, 322ff, **326**
– sources 94
– types **331**
– uncertain 77, 121

landmark values 267
learning 312
least-commitment strategy 91, 94, **330**
likelihood ratio 187
logic 29ff
– predicate 29ff, 104
loops, 74, 250, 267

machine capacity 208
maintenance 288, 317
matching, heuristic 92, 93
means-ends-analysis 252
medicine, decision-making in 288, 316
meta-classes 94
meta-knowledge 301
meta-planning 221
modification 94
modularization 213, 249
modus ponens 29
monitoring 90, 92, 96ff

no-good nodes 77

objects, active 45ff
observation groups 192
observations 121ff, 151ff
– abstraction 122
– false 152
– incomplete 122
– subjective 151
– time-dependent 122
– unreliable 121
odds 187
odds-likelihood form 187

parametrization 123
pattern matching 40, 322
planning 90, 92, 209, **329**
– hierarchical 251ff
– model-based 249ff
– non-hierarchical 250ff
– non-linear 252
precendence selection 121
prediction 90, 92, 95, 96
prediction value 131
problem **326**
– classification 89ff
  Breuker et al. 94ff
  Chandrasekaran 96ff
  Clancey 91ff
  Harmon 97ff
  Hayes-Roth et al. 90ff
  McDermott 98ff
  Stefik et al. 90ff
problem-solving methods 101ff, **327**
problem-solving type 90, 326
procedural problems 97
production planning 208, 211, 244ff, 288, 320
programming 4, 108
propose-and-exchange strategy 229ff, **330**
propose-and-revise strategy 223ff, **330**

quality control 288, 320

realizer 48

reasoning
- backward chained 11, 36, 38ff, 123, 133, 163, 322, **327**
- forward-chained 11, 36ff, 123, 224, 226, 243, 322, **327**
- hierarchical 91
- non-monotonic 11, 70ff
- probabilistic 57ff, 323
- temporal 11, 79ff, 91, **325**
- uncertain 91
repair 90
revision 213
roles 47
rules 35ff, 136, 322, **327**
- refinement 159
- structuring 41ff
- heuristic 68
- inconsistent 77

scheduling 98, 209
searching 6, 93
- A* algorithm 6
- alpha-beta strategy 6
- bidirectional 93
- data-controlled 93
- goal-controlled 93
- means-ends analysis 6
semantic nets 46, 107
sensitivity 131
similarity 192, 195
similarity type 195, 197
simulation 103, 265ff, **330**
- domains 265
- multiple-phase **330**
  - numerical 271ff
  - qualitative 278ff
- problem characteristics 266ff
- problem types 266
- problem-solving methods 267ff
- problem-solving 267
- single-phase 269ff, **330**
single-fault assumption **328**
skeletal plans 216, 218
slots 44
solution
- abstractions 92
- classes 132, 135
  - uncertainties of 150
- space **327**
- multiple 123, 154ff
special field **326**
specifity 131
state refinement 159
symptoms **328**
system
- analysis 91
- construction 91
- development 92
- maintenance 92
- knowledge-based **331**

T (terminological) box 47, 49
temporal predicates 82
tendering 288, 317
terminology, technical **326**
tests 136
theorem proving 6, 31
therapy planning 227ff
timetable generation 207, 211, 237
truth maintenance system (TMS) 72, 78, **328**
- assumption-based (ATMS) 72, 73, 76
- immediate check (ITMS) 76
- justification-based (JTMS) 72ff

unification 32, 40, 322

variables, restriction of 103ff

Waltz algorithm 56
weak-link analysis 266
work planning 288

zero probabilities 185, 188