A Knowledge-Intensive, Integrated Approach to Problem Solving and Sustained Learning

Agnar Aamodt

A doctoral dissertation submitted to the University of Trondheim, Norwegian Institute of Technology (NTH), Division of Computer Science and Telematics, 1991.
The author's address: SINTEF-DELAB\textsuperscript{1}

N-7034 Trondheim-NTH

NORWAY

Telephone, at work:  +47 7 592965

at home:  +47 7 977365

Network address:  agnar.aamodt@delab.sintef.no

(aamodt@delab.sintef.no)

Telefax:  +47 7 532586

Telex:  55 552 ELAB

Teletex:  52 03 75 ELAB

\textsuperscript{1}The name of this SINTEF division has recently changed from ELAB-RUNIT to DELAB. In the document text, ELAB-RUNIT is also used.
A Knowledge-Intensive, Integrated Approach to Problem Solving and Sustained Learning

A dissertation submitted to the degree Doctor Ingeniør

by

Agnar Aamodt

University of Trondheim
Norwegian Institute of Technology
Department of Electrical Engineering and Computer Science
Division of Computer Systems and Telematics
and
SINTEF-DELAB
Knowledge Engineering and Image Processing Group
The author’s address: SINTEF-DELAB\(^1\)
N-7034 Trondheim-NTH
NORWAY

Telephone, at work: +47 7 592965
at home: +47 7 977365

Network address: agnar.aamodt@delab.sintef.no
(aamodt@delab.sintef.no)

Telefax: +47 7 532586
Telex: 55 552 ELAB
Teletex: 52 03 75 ELAB

\(^1\)The name of this SINTEF division has recently changed from ELAB-RUNIT to DELAB. In the document text, ELAB-RUNIT is also used.
Abstract

Improved methods for development and maintenance of real world knowledge-based systems are strongly needed. A computer system targeted to be an active assistant in solving practical problems in, e.g., industries and governmental institutions, should be able to capture a variety of knowledge types (concept definitions, principles, heuristics, previous experiences), and effectively utilize this extensive body of knowledge in its problem solving method. Further, such a system should be able to improve over time, by extending and refining its knowledge as more problems are solved. It is a challenge for artificial intelligence research to develop methods that will make the building of such systems feasible. The work described in this dissertation is a contribution to that research.

The problem addressed in this research is that of developing a method which integrates problem solving with learning from experience within an extensive model of different knowledge types. A unified framework is developed through an analysis of various types, aspects and roles of knowledge relevant for the kind of systems described above. The framework contains a knowledge representation platform and a generic model of problem solving. It further specifies a general reasoning approach that combines reasoning within a deep model with reasoning from heuristic rules and past cases. Finally, it provides a model of learning methods that retain concrete problem solving cases in a way that makes them useful for solving similar problems later. The framework emphasizes knowledge-intensive case-based reasoning and learning as the major paradigm. A comprehensive and thorough knowledge model is the basis for generation of goal related explanations that support the reasoning and learning processes. Reasoning from heuristic rules or from 'scratch' within the deeper model is regarded partly as supporting methods to the case-based reasoning, partly as methods to 'fall back on' if the case-based method fails. The purpose of the framework is to provide an environment for discussion of different approaches to knowledge intensive problem solving and learning.

The framework is used to analyse four existing systems. The four systems focus on different methodological issues of knowledge intensive problem solving and learning. Each system represents interesting solutions to subproblems, but none of them provide a scope that is broad enough to represent the type of method requested for developing and maintaining complex applications in a practical, real world setting. A new system architecture - called CREEK - is proposed, as an improvement to current approaches.

CREEK specifies a structural and functional architecture based on an expressive, frame-based knowledge representation language, and an explicit model of control knowledge. It has a reasoning strategy which first attempts case-based reasoning, then rule-based reasoning, and, finally, model-based reasoning. CREEK learns from each problem solving session by updating its collection of cases, irrespective of which reasoning method that succeeded in solving the problem. The system interacts with the user during both problem solving and learning, e.g. by asking for confirmation or rejection of unexplained facts. The knowledge representation system, including an explicit model of basic representational constructs and basic inference methods, has been implemented. Otherwise, CREEK is an architectural specification - a system design. Its main characteristics are demonstrated by an example from the domain of diagnosis and treatment of oil well drilling fluid (mud).
Acknowledgements

Many colleagues and friends have provided valuable contributions to the result presented here. This study would not have been undertaken and completed without the help and support, the stimulating discussions, and the constructive critiques of my colleagues in the Knowledge Engineering Laboratory at ELAB-RUNIT: Geir Aakvik, Tore Amble, Jorun Eggen, Ole Jacob Mengehol, Inge Nordbø, Mette Vestli, and Ingeborg Sølvberg. Discussions concerning automated knowledge acquisition within the METAKREK project triggered the research reported here. The testing, performed by this group, of the knowledge acquisition method and implemented tools which came out of the METAKREK project gave valuable insight into the difficulties of knowledge modelling for practical, real world applications. The continuing discussions on these and other relevant topics of knowledge acquisition, and development and maintenance of knowledge-based systems - has continued through the group's participation in ACKnowledge (an ESPRIT-II project), and has provided a fruitful environment for knowledge acquisition and machine learning research throughout the dissertation period.

I would also like to thank Leif Stinessen in the Psychology Departement, University of Trondheim, and Magne Dybvik in the Philosophy Departement. They provided a basic understanding and took part in many discussions related to the human cognitive aspects of knowledge representation, problem solving, and learning, as well as to the impact of cognitive theories and models for artificial intelligence and knowledge engineering.

Basic ideas of this work emerged during a visiting year in the Department of Computer Sciences, University of Texas at Austin. I would like to thank Bruce Porter for many enlightening discussions and useful pointers relevant to all parts of this research, and Ray Bareiss for discussions on knowledge acquisition and case-based reasoning in general, and Protos in particular. Brad Blumenthal, Karl Branting, Rita Duran and Dan Dvorak provided interesting viewpoints to combined reasoning methods, and Ray Mooney and Rob Holte contributed significantly to my understanding of machine learning fundamentals. The cooperation with James Lester, Ken Murray, Karen Pitman and Art Souther on the Botany project conducted at MCC, provided highly valuable insights and experience in knowledge-intensive problem solving and learning. I would also like to thank Doug Lenat and the rest of the 'Cyclists' for their help and assistance during my association with their group.

In addition to my colleagues at ELAB-RUNIT, highly appreciated critiques and comments on earlier drafts of this dissertation have been given by Steve Adey, Eric Barber, Magne Dybvik, Velitchko Koulitchev, Christian Larssen, Øystein Nytrø, Han Reichgelt, and Arne Sølvberg.

Thanks to Bjørnung Jensen, Terje Tømmervik and Kjell Korsan for providing the domain knowledge of drilling fluid diagnosis which formed the basis for the application example discussed in the dissertation, and to Pål Skalle for additional hints and guidelines related to this work.

I am also indebted to Joe Gorman for his general advice on the English language, his patiently reading of the chapters as they appeared in almost finished form, as well as his questions and comments on technical matters.

A special thank to my supervisor, Professor Arne Sølvberg, for support and guidance through the entire study and research period, and for constructive comments and suggestions to the focusing of topics and the structuring of results presented in this dissertation.

Finally, a warm thank to my wife Inger and my children Gunhild, Sigmund, and Solveig for their encouraging support during the entire period of study and production of this dissertation.

Financial support for this research was provided by my employer, SINTEF - ELAB-RUNIT, and by the Norwegian Foundation for Technical and Scientific Research (NTNF) via the METAKREK project and through personal research stipends (grant numbers ST 10.12.220023 and ST 30.31.220238).
## CONTENTS

### Abstract

### Acknowledgements

### PART I - INTRODUCTION .............................................................................................................. 1

#### 1. Background and Motivation ................................................................................................. 3

- 1.1. Knowledge-based systems, problem solving competence and learnability ......... 3
- 1.1.1. Problem solving in real world domains ................................................................. 5
- 1.1.2. The goal of this research .................................................................................... 7
- 1.2. Knowledge modelling and learning in weak theory domains .......................... 7
- 1.2.1. The knowledge acquisition problem ................................................................. 10
- 1.2.2. Knowledge representation ............................................................................... 13
- 1.2.3. Sustained learning ......................................................................................... 14
- 1.3. Towards a system design for competent problem solving and sustained learning .................................................................................................................. 18

#### 2. Overview of the report ......................................................................................................... 21

### PART II - FRAMEWORK AND COMPARATIVE ANALYSIS .......................................................... 25

#### 3. A Framework for Knowledge Modelling, Problem Solving and Sustained Learning ........ 27

- 3.1. System requirements and framework perspective ................................................ 28
- 3.2. Reasoning and inference in problem solving and learning ................................ 30
- 3.3. Descriptive dimensions ............................................................................................ 34
- 3.4. Representational platform ..................................................................................... 36
- 3.4.1. Modelling of expertise .................................................................................... 37
- 3.4.1.1. Knowledge types ................................................................................... 38
- 3.4.1.2. From knowledge to expertise ................................................................ 43
- 3.4.2. A model of representational primitives ....................................................... 48
- 3.4.3. A network view of knowledge ....................................................................... 57
- 3.5. Problem solving and learning - a knowledge intensive view ......................... 58
- 3.5.1. The concept of explanation ......................................................................... 59
- 3.5.2. Associational and elaborative reasoning ...................................................... 60
- 3.5.3. What constitutes a good explanation? ........................................................... 62
- 3.5.4. A Computational model of explanation ....................................................... 64
- 3.5.4.1. Explanations and Coherence of knowledge ........................................... 65
- 3.5.4.2. Explanation structures ........................................................................... 66
- 3.5.5. Phases in problem solving ............................................................................. 67
- 3.5.6. An explanation based model of reasoning ................................................... 71
- 3.5.6.1. Reasoning from past cases .................................................................... 78
- 3.5.7. An explanation based model of sustained learning ...................................... 79
- 3.5.7.1. Learning by retaining past cases ........................................................... 83
- 3.5.8. An integrated model of problem solving, reasoning and learning ............. 84
- 3.6. Chapter conclusion ................................................................................................. 85
4. Competent Problem Solving and Sustained Learning
   - A Review of Current Approaches ................................................................. 87
   4.1. Essential properties and current systems ..................................................... 87
   4.2. Critical factors ............................................................................................ 88
   4.3. PROTOS ........................................................................................................ 90
       4.3.2. The problems addressed ...................................................................... 92
       4.3.3. The knowledge model ......................................................................... 93
       4.3.4. The problem solving process ................................................................. 98
       4.3.5. The reasoning ........................................................................................ 98
       4.3.6. The learning .......................................................................................... 100
       4.3.7. Relating Protos to the critical factors .................................................... 102
       4.3.8. Relating Protos to the framework requirements ....................................... 103
   4.4. CASEY .......................................................................................................... 105
       4.4.1. General ................................................................................................. 105
       4.4.2. The problems addressed ...................................................................... 106
       4.4.3. The knowledge model ......................................................................... 108
       4.4.4. The problem solving process ................................................................. 113
       4.4.5. The reasoning ........................................................................................ 114
       4.4.6. The learning .......................................................................................... 115
       4.4.7. Relating CASEY to the critical factors .................................................. 118
       4.4.8. Relating CASEY to the framework requirements ..................................... 119
   4.5. CHEF ............................................................................................................ 121
       4.5.1. General ................................................................................................. 121
       4.5.2. Knowledge model and problem solving process ..................................... 123
       4.5.3. The learning process ............................................................................ 125
       4.5.4. Relating CHEF to the framework requirements ....................................... 126
   4.6. JULIA ............................................................................................................ 127
       4.6.1. General ................................................................................................. 127
       4.6.2. Knowledge model and problem solving process ..................................... 129
       4.6.3. The learning process ............................................................................ 130
       4.6.4. Relating JULIA to the framework requirements ....................................... 130
   4.7. Chapter conclusion ...................................................................................... 131

5. System overview and knowledge structure ..................................................... 137
   5.1. The CREEK architecture .......................................................................... 137
       5.1.1. Structural architecture ....................................................................... 139
       5.1.2. Functional architecture ....................................................................... 141
   5.2. Knowledge representation .......................................................................... 145
       5.2.1. Representing different knowledge types in Creek .................................. 145
       5.2.2. Creek's model of representational primitives ......................................... 149
       5.2.3. The representation language ................................................................ 152
       5.2.4. Inference in Creek ............................................................................... 153
           5.2.4.1. Inference methods ......................................................................... 154
           5.2.4.2. Plausible inference ........................................................................ 156
           5.2.4.3. Inference control .......................................................................... 158
       5.2.5. Expressing different forms of knowledge within the representation language .................................................................................. 159
           5.2.5.1. The Representation of descriptive concepts and facts....................... 159
           5.2.5.2. The Representation of cases ............................................................ 162
           5.2.5.3. The representation of Heuristics ....................................................... 166
           5.2.5.4. The representation of explanations ............................................... 167
       5.2.6. The CreekL implementation ................................................................. 168
APPENDIX

A1. An introduction to case-based reasoning ................................................................. 255
   A1.1. General ............................................................................................................. 255
   A1.2. The CBR cycle ............................................................................................... 257
   A1.3. Organization of the cases in memory .............................................................. 261
      A1.3.1 The dynamic memory model (Schank/Kolodner) ..................................... 262
      A1.3.2 The category & exemplar model (Porter/Bareiss) .................................... 264
   A1.4. Concluding remarks ....................................................................................... 265

A2. A summary of the CreekL knowledge representation language ................................ 267
   A2.1. General ........................................................................................................... 267
   A2.2. Implementation ............................................................................................... 270
   A2.3. Loading and saving a knowledge base ............................................................ 271
   A2.4. Building of concepts ...................................................................................... 271
   A2.5. Removing frame contents ............................................................................. 272
   A2.6. Changing frame contents ............................................................................. 273
   A2.7. Displaying concepts ..................................................................................... 273
   A2.8. Constraints enforcement ............................................................................. 276
   A2.9. Retrieval of value content .......................................................................... 277
   A2.10. Value inheritance ....................................................................................... 277
   A2.11. Traversing relational links ......................................................................... 279
   A2.12. Spreading of activation .............................................................................. 279

A3. A description of Creek terms ............................................................................... 281

REFERENCES ............................................................................................................. 285

FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The knowledge acquisition cycle ................................................................. 11</td>
</tr>
<tr>
<td>1.2</td>
<td>Sustained learning through problem solving ................................................. 16</td>
</tr>
<tr>
<td>3.1</td>
<td>Structure of knowledge based processes ......................................................... 30</td>
</tr>
<tr>
<td>3.2</td>
<td>Dimensions of knowledge ............................................................................... 39</td>
</tr>
<tr>
<td>3.3</td>
<td>Some representational dimensions ................................................................. 41</td>
</tr>
<tr>
<td>3.4</td>
<td>A component model of expertise ...................................................................... 44</td>
</tr>
<tr>
<td>3.5</td>
<td>Layers of expertise in KADS ......................................................................... 45</td>
</tr>
<tr>
<td>3.6</td>
<td>Inference- and task-structures for systematic diagnosis ..................................... 46</td>
</tr>
<tr>
<td>3.7</td>
<td>Integrated four-layered expertise model .......................................................... 47</td>
</tr>
<tr>
<td>3.8</td>
<td>Representations vs. world objects ................................................................... 49</td>
</tr>
<tr>
<td>3.9</td>
<td>A taxonomy of conceptual relations .................................................................. 51</td>
</tr>
<tr>
<td>3.10</td>
<td>A subclass hierarchy of facet types ................................................................. 52</td>
</tr>
<tr>
<td>3.11</td>
<td>Representational primitives ............................................................................. 54</td>
</tr>
<tr>
<td>3.12</td>
<td>Integrating cases and rules into the conceptual network ................................... 57</td>
</tr>
<tr>
<td>3.13</td>
<td>Main phases of problem solving ...................................................................... 71</td>
</tr>
<tr>
<td>3.14</td>
<td>Clancey's heuristic classification model .......................................................... 72</td>
</tr>
<tr>
<td>3.15</td>
<td>The explanation-based reasoning model ............................................................ 75</td>
</tr>
<tr>
<td>3.16</td>
<td>Main steps in reasoning from past cases ........................................................... 78</td>
</tr>
<tr>
<td>3.17</td>
<td>A model of learning from experience ............................................................... 80</td>
</tr>
<tr>
<td>3.18</td>
<td>Main steps in learning by retaining past cases .................................................. 83</td>
</tr>
<tr>
<td>3.19</td>
<td>The framework's model of integrated learning, problem solving, and reasoning ... 84</td>
</tr>
</tbody>
</table>
PART I

INTRODUCTION
Chapter 1

Background and Motivation

1.1. Knowledge-Based Systems, Problem Solving Competence and Learnability

Recently, knowledge-based systems - and particularly expert systems - have gradually moved from research environments into practical use. However, the vast majority of applications developed to date\(^1\) are relatively simple compared to problems that industry and governmental bodies in general are facing. It has been a common experience amongst builders and users of expert systems that the enthusiastic support their system received while being developed and initially tested soon faded away. "The system was interesting, and showed some promising approaches to computer assisted problem solving within our domain, but it has not actually been used". Such statements are frequently heard when the story of a particular expert system project is told. Admittedly, some of these systems were developed for competence building or demonstration purposes only, but unfortunately this has not always been the case.

Success or failure in developing a knowledge-based system depends upon how well expectations map to what is achievable within the given limits. The limits are, to some extent, imposed by the set of methods, techniques and tools that constitute the state-of-the-art in knowledge engineering and artificial intelligence (AI) in general. High quality, expert level, knowledge-based systems are difficult and time consuming to develop. They typically address complex real world problems - like medical diagnosis, fault-finding and repair of industrial processes, design of technical production equipment, or planning of financial investments. The requirements for robustness\(^2\), user-friendliness and efficiency are high, since the systems are supposed to be used daily in cooperation with people and/or other systems. They may also have to cope with problems on the border of (or slightly outside) their special domain of

---

\(^1\)Market overviews and trends, including lists of existing systems, are discussed in [Buchanan-86], [Reddy-88], [Harmon-88], [Harmon-89].

\(^2\)Robustness refers to the ability to cope with unexpected situations.
In order to meet these challenges, knowledge-based systems need to become more competent; they need a deeper understanding of their domain of operation - than what is generally the case for systems in use today.

Human experts are competent and robust problem solvers since they possess a fairly coherent body of general world knowledge, specific knowledge of domains, and a huge library of specific past problem solving cases. People are good problem solvers because we base our methods on a general understanding of the world we operate in. We also maintain a memory of past problem-solving episodes integrated into this fundamental knowledge structure. While solving problems, we are frequently reminded of similar past problems. Using more general knowledge as support, we are able to adapt the solution (or solution path) of a previous case to solving a new problem. Problem solving strategies help us to focus the use of knowledge within the current context and according to our goals. Through success and failure in achieving our task, we learn to do things better the next time.

Computer problem solvers - as well as people - need to function within a continually evolving environment. A system's knowledge needs frequent updating and refinement. If not, the system will gradually degenerate and become useless. It is for practical reasons unfeasible to rely solely on manual updating and refinement procedures in a large and complex body of knowledge. The problem of knowledge maintenance is therefore a major challenge for the AI community. If a system is to continually maintain and improve its problem solving competence, it will need to incorporate methods for automated learning from experience. For the learning to be meaningful and effective, it must be based on - and guided by - an understanding of the learning domain. Hence, both the ability of effective learning from experience, and competent problem solving, should be based on a thorough and comprehensive knowledge model of the application domain.

These prospects for future knowledge-based systems have motivated an increased interest in the AI subfields of knowledge modelling (e.g., [Clancey-89, Steels-89, Breuker-89, Sølvberg-88]), and machine learning (e.g., [Michalski-86, Schank-86b, Carbonell-89, Rissland-89]). An interesting observation over the last few years has been a gradual shift from toy domains to real world problems within both subfields, and an emphasize on knowledge-intensive methods in

---

1 The term competence refers to an agent's ability to solve a variety of problems within a domain at a generally acceptable level of quality.

2 The term knowledge model, as used in this dissertation, refers to a total model of the knowledge (facts, assumptions, heuristics, past experiences, methods, strategies) that an agent capable of solving problems possesses. Knowledge, competence, and expertise should in general be regarded as synonyms, since none of them excludes any type of information or method present in the others. However, the three terms represent slightly different perspectives, the latter two indicating an emphasis on the use of knowledge to achieve a result. But this is knowledge, too, only of another kind. A distinction sometimes made between competence and expertise is that expertise is regarded as being more focused, more specialized and operationalized, than competence.
learning. During the last few years there has been a rapid growth of interest and activities in machine learning techniques that enable continual improvement of a system as a natural part of the problem solving process itself [Mitchell-85, Kolodner-87, Van de Velde-88a, Bareiss-88a, Koton-89]. The ability to learn from problem solving experience - continually improving the knowledge by updating the knowledge base after each problem solving session - is here referred to as sustained learning. A fast growing research area addressing this problem is case-based reasoning, where important parts of past problems and problem solving sessions are extracted and saved (as cases) for future use. Thus, case-based reasoning is not only a learning paradigm, but also an approach to problem solving: A new problem generates a reminding to a previously solved case, which is retrieved in order to apply the past solution - usually after some modification - to the new problem [Kolodner-88, CBRW-89].

1.1.1. Problem Solving in Real World Domains

Most real-world problems that knowledge-based systems are addressing, are open problems. An open problem is characterized by having a weak or intractable domain theory. That is, there is no theory that completely describes the problem domain, and from which correct solutions can be derived or proved. Nevertheless, human beings are clever at inferring plausible conclusions and deriving acceptable and useful solutions to problems in these domains. Typical weak theory domains are medical diagnosis, geological interpretation, investment planning, and most engineering domains (i.e. domains that involve interaction with the external world). A weak domain theory is characterized by uncertain relationships between its concepts (as pointed out in [Porter-89]). The stronger the theory, the more certain are its relationships. At the very far end of this spectrum are the complete theories. The relationships of complete theories are certain, e.g. as expressed by standard logical or structural relations. Domains with strong domain theories are, for example, mathematical domains, ‘block worlds’ and some games - like checkers and chess. Even some strong theory domains may incorporate problems that turn out to be open when addressed by a problem solver. Chess, for example, has a strongest possible - a complete - domain theory, but solving the problem of chess-playing by an implementation of the theory is violated by the its intractability. The concept of a ‘winning plan’ in chess is theoretically deducible, but there is no efficient algorithm to infer it in the general case.

1 The term case-based reasoning implies that not only the learning, but also the reasoning process undertaken when learned knowledge is used to solve new problems, is case-based. The term case-based learning is sometimes used synonymously, but this term is ambiguous since it occasionally refers to non-sustained methods where a set of cases is used to refine a knowledge-base, in a batch-like manner, by inducing general knowledge.

2 A relationship is an expression that relates two or more concepts to each other via a relation. For example, "John owns a car" is a relationship between the concepts "John" and "car" via the (binary) relation "owns".
Adding to the problems of weak or intractable theories, problem solving in the real world is in general complicated by *incomplete and noisy problem specifications*. This puts two important requirements on an intelligent problem solver:

- The ability to infer missing information, to generate and check expectations, and to justify whether a piece of information is relevant or not.
- The ability to interact with the information source (usually a human being) in order to extract additional information, and for confirmation or rejection of suggested conclusions.

Therefore, a real world problem solver should be able to dynamically change the scope and goals of a problem during problem solving, and to deal with the contradictions and inconsistencies that are introduced in this process. The basic method that enables a system to infer meaningful results, justify its own decisions, and interact intelligibly with the user, may be viewed as a process of *generating and evaluating explanations*. In real world, weak theory domains, it is seldom the case that an inferred result can be *proved* true or false. Instead, the problem solver generally has to produce and justify its conclusion by explaining to itself why the conclusion is a plausible one [Schank-86c]. This role of explanations extends from the understanding of a problem description into other parts of the problem solving process. A result derived from a previous problem solving case, for example, may need to be supported by an explanation based on more general domain knowledge. The emphasis on explanation in the reasoning process, as just described, requires a thorough and deep knowledge model in order to produce plausible, meaningful explanations.

Results from cognitive psychology [Andersson-85, Carbonell-83, Mayer-89] indicate that people combine general domain principles and world knowledge with associations in the form of general rules, or specific previous experiences, when solving non-trivial problems. The 'traditional' AI approach to modelling weak theory domains is by heuristic rules. A more recent approach is reasoning from past cases. A specific experience, as well as a heuristic rule (or rule set), represents a shallow piece of knowledge, i.e. a mere association between some problem descriptors and a solution. The advantages of a such an approach are simplicity and efficiency. The main disadvantage is in depth of understanding, which leads to insufficient problem solving competence, superficial dialoguing with users, and difficulties in learning. To be able to handle these problems, shallow associational reasoning (from specific cases or generalized rules) should be combined with reasoning methods based on deeper, more thorough knowledge. This is also a direction that current research is taking: Case-based methods have began to explore the use of deeper knowledge as support for the case-based reasoning process [Bareiss-

---

[The kind of explanations referred to here are internal explanations that are part of the system's reasoning process, not external explanations produced for the benefit of the user.]}
88a, Hammond-87, Branting-89], as well as a knowledge source to fall back on if the case-based method fails [Koton-88]. Within the world of rule-based reasoning, an increasing number of research activities address the problem of combining heuristic, shallow knowledge with deeper knowledge models¹ (a collection of recent papers are presented in [Expert-89]).

1.1.2. The Goal of this Research

The problem addressed in this dissertation is that of building and maintaining highly competent expert systems for real world applications. The goal of the research is twofold:

1. To develop a **framework** for describing problem solving and sustained learning in knowledge rich environments. The framework should
   - define a knowledge-intensive, case-based approach to problem solving and sustained learning
   - provide a language and a set of generic models in which existing approaches may be described and analysed

2. To develop a **system**² for problem solving and learning in knowledge rich environments, within the specifications of the framework. The system should contain
   - an expressive, extendible representation system for different types of knowledge
   - a multi-paradigm problem solving method that is able to combine reasoning from past cases with reasoning from more general knowledge
   - a knowledge-intensive case-based method for sustained learning

Given this goal, the next section of this introductory chapter discusses what impact this goal may have on the way knowledge should be viewed, structured, and modelled. This is further elaborated by describing the relationships between this research and the more general problem of knowledge acquisition. Regarding machine learning as a kind of knowledge acquisition method, this is followed by a discussion of different approaches to sustained learning, related to the current state of the art in machine learning.

---
¹Sometimes referred to as second generation expert systems, although this term is also used with slightly different meanings (e.g. a system that learns [Steels-87], a system that explicates its reasoning [Johnson-88]).
²The term *system* should not be interpreted in the strict sense of a fully implemented computer system, but more generally as a system design, i.e. an architecture and a set of methods operating on the architecture.
1.2. Knowledge Modelling and Learning in Weak Theory Domains

The type of knowledge-based systems aimed at by this research are expert systems. Expert systems are defined in a general sense as systems that are able to solve particular types of problems at the level of a human expert. An additional criterion is an ability to interact with users in a cooperative and intelligible way. This criterion reflects a view of expert systems as interactive, open knowledge-based decision support systems. That is, expert systems which are regarded as interactive, user-controlled discussion partners instead of a type of system which prompts the user according to a fixed, system-driven dialogue and finally presents a solution. The type of applications primarily addressed in the work reported here are diagnosis and repair problems.

The notion of knowledge-based decision support systems also indicates that these systems are not stand alone expert systems 'boxes'. Typically, they are information systems in a more general sense, in which knowledge-based components are integrated (sometimes referred to as 'embedded expert systems'). Activities within the fast growing, interdisciplinary field of AI and Information Systems (e.g. [Sølvberg-88b]) has provided valuable input to this view of real world systems, and it is important that the AI methods targeted for such systems are developed with an 'integrated information system' view in mind.

In order to enable high quality, active decision support in real world, weak theory domains, problem solving needs to be based on an extensive body of domain knowledge. To enable a sufficient degree of competence and efficiency of a problem solver, the relevant domain concepts, their inter-relationships and semantic constraints need to be thoroughly modelled into a tightly coupled knowledge structure, and at different levels of detail and depth. It should serve as a fundament for interpretation of information, inferring of consequences, justification of conclusions, etc. The model should be able to solve problems on its own, as well as to support problem solving based on more shallow, heuristic associations. Heuristic rules, for example, may be efficient for deriving a solution, since a rule chain may be triggered that lead directly to a solution. But the match of problem descriptors to rule terms should be a semantic match, and not be based on superficial similarities. In a competent and robust problem solver, matching should be based on an explanation of similarities, rather than a mere equality of concept names.

---

1 Although the focus is on expert systems, it should be noted that the methodology described in this dissertation is potentially applicable to other kinds of knowledge-based systems as well. Being an integrated approach to knowledge modelling, problem solving, and learning, it applies to any knowledge-based system where a knowledge intensive approach to learning from experience is relevant (e.g. robot systems, natural language understanders).

2 Even if the domain does not have a strong theory, it is assumed that there is a substantial amount of knowledge (facts, concept definitions, relationships, heuristic, experienced cases) which it is possible to describe explicitly. If not, the knowledge-based paradigm will fail.
In most current systems, knowledge is described as a set of heuristic rules, sometimes supported by a structural description of some terms referred to in the rules (e.g. an isa and/or part-of hierarchy). This gives the system a model of its domain which is highly tuned to the anticipated questions given to the system. Because these systems generally lack a thorough knowledge model of their domain, they are not able to handle unexpected situations in an intelligible manner. A thorough knowledge model offers both a deep and broad description of the application domain. A deep knowledge model describes the concept definitions and basic principles underlying the more operational knowledge, typically by representing a comprehensive static and/or dynamic model of a domain. Examples are models of physical systems as described by "qualitative physics" approaches, and component models describing structural and functional properties of systems. Deep models typically expand their depth along one or two dimensions or 'views', e.g. structural, causal and/or functional relations. Another perspective to deep knowledge is detailed knowledge: A model is regarded as deep if it contains a thorough description of specific facts tightly coupled to each other and to more general concepts. A broad knowledge model captures knowledge along multiple perspectives and types of dimensions, also bringing knowledge of the general world and of related neighbouring domains into the model.

As previously indicated, human problem solving involves several mechanisms working in a focused way towards a solution. There is an emerging view in AI that the problem solving paradigm from the early days of knowledge-based systems - with a purely declarative knowledge base controlled and accessed by a single, domain independent inference machine - does not hold (e.g. [Schank-86b, Chandrasekaran-87, McDermott-87, Steels-89]). Knowledge is of minor value 'in itself'. There are a number of different perspectives to a piece of knowledge, and various dependencies and constraints related to the interpretation of its knowledge content, i.e. its meaning within a particular context. For a knowledge-intensive approach to problem solving and learning, knowledge should be viewed within a problem solving context and related to its purpose and use.

The importance of having a thorough and extensive model of both special and general knowledge is formulated by Lenat and Feigenbaum [Lenat-87] in the following two statements:

---

1The importance of incorporating broad and general world knowledge into future knowledge-based systems is emphasised in the CYC-project [Lenat-86], where this idea is taken to the extreme of constructing a huge knowledge base incorporating essentially all existing common sense knowledge. The idea is that this common sense knowledge fundaments will form an environment for developing more specific knowledge bases.

2An overview and a collection of articles is presented in [Bobrow-84], and a comparison of three different approaches is given in [Bonissone-85].

3E.g. component models of electrical circuits, as described in [Mitchell-85, Davis-82a, Brown-82].
The Knowledge Principle: A system exhibits intelligent understanding and action at a high level of competence primarily because of the specific knowledge that it can bring to bear: the concepts, facts, representations, methods, models, metaphors, and heuristics about its domain of endeavour.

The Breadth Hypothesis: Intelligent performance often requires the problem solver to fall back on increasingly general knowledge, and/or to analogize to specific knowledge from far-flung domains.

The time and effort that is needed to develop a thorough model of competence in an expert system will be large compared to most current approaches. Of course, the degree of depth and breadth has to be decided according to the type and role of the system that is to be developed. Nevertheless, such a modelling task is a very difficult one, and improved methodologies for modelling of knowledge at the desired degree of competence are needed. During the last few years a lot of research has concentrated on analysis and modelling of knowledge, and on methods for developing an explicit, total knowledge model (expertise model) of a domain - including factual knowledge, strategies, and problem solving methods [e.g., Clancey-84a, Wielinga-86, McDermott-86, Keravnou-86, Breuker-87, Steels-87, Sticklen-87, Aamodt-87, Gruber-88, Clancey-89, Steels-89]. This area has been the major focus of a four-year research program within the Knowledge Engineering Laboratory at ELAB-RUNIT, in cooperation with the Norwegian Institute of Technology - resulting in the METAKREK methodology and tool [Sølvberg-88, Aakvik-88, Vestli-89, Nordbø-89, Nordbø-90]. Recently, a large European effort has been undertaken to develop an integrated knowledge engineering workbench, incorporating tools and techniques for elicitation, analysis, design and maintenance of knowledge-based systems. (The ACKnowledge project, where ELAB-RUNIT is one of the partners [Marty-89, Eggen-90]).

1.2.1. The Knowledge Acquisition Problem

The difficulties experienced in building and maintaining knowledge-based systems are generally referred to as the knowledge acquisition problem. Knowledge acquisition is a cyclic process running throughout the entire life of a system. Broadly, knowledge acquisition may be split into two main tasks:

* Initial knowledge modelling, where the initial competence model of domain knowledge, problem solving strategies, and reasoning methods are developed. The objective of this task is to capture - within a computing system - the relevant knowledge that a human being would use to solve problems within the domain. Incomplete as this model may be,

\footnote{Also called the knowledge acquisition bottle-neck.}
it constitutes the knowledge environment wherein problem solving initially takes place, and from which subsequent learning methods get their support.

* **Knowledge maintenance**, where refinement of the initial knowledge model and learning of new knowledge takes place. The objectives of the task are
  - to correct errors, and improve the knowledge quality
  - to improve performance efficiency
  - to adjust to a changing external environment

Figure 1.1 shows a process-and-storage model of the knowledge acquisition cycle, where the two tasks of initial knowledge modelling and knowledge maintenance have been broken down into subprocesses. The processes of initial knowledge modelling are contained within the inner, light-gray box, while knowledge maintenance - viewed as a gathering of experience followed by a looping back into the knowledge modelling processes - is illustrated by the surrounding box and its links into the inner box.

The knowledge elicitation process attempts to extract knowledge from a human expert. This initially unstructured knowledge is analysed and structured into an *intermediate knowledge model*. The intermediate knowledge model may be viewed as a specification for design and implementation of the target system.

---

**Figure 1.1: The Knowledge Acquisition Cycle**

The iteration process of developing and maintaining models of knowledge and problem solving expertise in knowledge-based systems. The rounded boxes are processes, and the sharp-cornered boxes are virtual stores of knowledge and information. Ellipses are means of updating knowledge. Arrows indicate flow of knowledge and information.
The purpose of the intermediate model is to capture and structure various types of knowledge from the perspective of the domain and the task of the application system, without being constrained by implementational limitations. The result of this intermediate stage should be a model of domain knowledge and problem solving methods at the knowledge level\(^1\), i.e. at the level of knowledge content, as opposed to the symbol level which deals with manipulation of symbols that represent knowledge.

The feedback from the intermediate model to the mental model highlights an important property of the knowledge acquisition process: A person’s mental model is not a fixed or complete body of knowledge ready to be ‘transferred’ into some explicit representation. Therefore, knowledge acquisition should be viewed as an interactive modelling process, where the expert’s knowledge - or at least the explicit expression of it - is gradually expanded and refined during the acquisition process. This is sometimes referred to as the cooperative view, or constructive view, to knowledge acquisition [Morik-89, Littman-89]. It follows that knowledge acquisition methods and tools should provide a flexible environment for updating, enhancing and revising the model currently being developed, and not be constrained by, for example, a strict top down refinement strategy. A knowledge acquisition environment - including manual as well as (partially) automated methods - should be an assistant that enables a user\(^2\) to continually restructure and further develop his\(^3\) mental model of the domain. Thus, an important role of a knowledge acquisition tool should be to derive consequences of new knowledge entered, suggest modifications, etc. in order to integrate\(^4\) the new knowledge into the existing structure. This is particularly important for the knowledge maintenance part of the acquisition cycle (described below), since the system then is assumed to have an existing knowledge base of a significant size. During the knowledge integration process, contradictions may be identified within various types and levels of knowledge. Some contradictions may be resolvable by the system itself, some may need to be resolved by the user. This interaction is an example of a type of feedback that may improve the person’s mental model of the domain, which in turn leads to an improved knowledge model within the system.

The difficult and challenging task of the design and implementation phase is to transform the intermediate model into a computationally tractable model, i.e. into a representation and an accompanying program that is able to solve the problems that the system is aimed at. In addition to problem solving, the computer model should be able to take advantage of its

\(^1\)The term knowledge level was introduced by Allen Newell [in Newell-82]) as a separate computer system level above the symbol level (program level).

\(^2\)The user may be an expert (or sufficiently competent end user), a knowledge-engineer, or both.

\(^3\)In this dissertation, the term ‘he’ (and derivatives like ‘his’ and ‘him’) is adopted, for simplicity reasons, as a neutral term referring to a person in general. Hence, whenever such terms are encountered they should be read as ‘he or she’, ‘his or hers’, etc.

\(^4\)Different aspects of - and methods for - knowledge integration are described in [Murray-89, Eggen-90, Brazdil-90].
experience by continually extending and refining its knowledge content as new problems are solved. Design and implementation involves choosing or developing languages and tools for knowledge representation, and for programming the reasoning and learning components.

Two different types of maintenance processes are indicated in the figure: One that directly updates the knowledge model of the operating target system each time a new problem is solved, and one involving a more substantial revision process that, ideally, should be done within an implementation independent knowledge model. Sustained learning by retaining experience in the form of past cases necessarily has to take place within the implemented target system.

The two main tasks of knowledge modelling and knowledge maintenance involve the following three basic problems:

1. The knowledge elicitation problem, where the focus of the problem is a competent person (the knowledge source). The problem is how to make a person reveal and explicate the knowledge he uses to solve domain problems. People are good problem solvers, but not always good at explaining their problem solving strategies, reasoning paths, and exactly what knowledge they use in solving a particular problem. The knowledge elicitation problem is a problem of verbalization, de-compiling of automated knowledge\(^2\), and of knowledge analysis. According to a cooperative, or constructive, view to knowledge acquisition, success in solving the elicitation problem (for a complex application) will depend on whether an appropriate mode of interaction with the person involved is achieved. This interaction should assist the expert and knowledge engineer in constructing an explicit representation - an intermediate model - of the expert's gradually emerging mental model of the problem domain. Elicitation, therefore, should be regarded as part of an elicitation-modelling-feedback cycle, rather than as an independent first step.

2. The knowledge representation problem\(^3\), where the focus of the problem is both the expert/knowledge engineer involved and the computer system to be developed. The problem is how to formally describe the knowledge in a way expressive enough to capture all relevant knowledge, efficient enough to constitute a useful computational model, and close enough to a human interpretable language to enable effective manual inspection and refinement.

\(^1\)In the work presented here, only the knowledge-based aspects of a system are addressed. Of course, in developing a knowledge-based system, other requirements than purely AI related ones are important as well. A knowledge based system for real-world applications is subject to all the general requirements of computer systems that are to operate in, e.g., an industrial environment.

\(^2\)Automated knowledge [Anderson-85] is operational knowledge where the underlying principles have been forgotten (have been compiled or automated).

\(^3\)The representation problem is actually two subproblems - one involving an intermediate representation and one involving knowledge representation in the target system (computer internal model). Since this dissertation primarily addresses problems of sustained learning, representing knowledge in a computer internal model is the representation problem of main concern here.
3. **The learning problem**, where the focus of the problem is the knowledge based system under development or operation. The problem is how to enable a computer system to acquire knowledge in a way that meaningfully integrates it into the existing knowledge, and leads to improvements in knowledge quality, problem solving performance, and further learning. The learning may be fully automated, or (more typically) semi-automated, involving interaction with a teacher.

The research reported here concentrates on the *representation problem* and the *learning problem* with respect to the computer internal model.

### 1.2.2. Knowledge Representation

As indicated earlier, the representation problem emphasized in this work is the issue of *expressiveness*. Current representation tools are to a large extent based on a rule-based paradigm. Even if many tools enable some kind of structured descriptions of objects referred to in the rules, the ability to build a comprehensive knowledge model of all relevant domain relationships is limited\(^1\). In many expert system projects today, the important knowledge modelling phase of the knowledge acquisition cycle hardly exists. Instead, the effort is put into squeezing the expert’s knowledge into a predefined representation and reasoning formalism. Thorough knowledge modelling requires a comprehensive set of relations (e.g. structural, causal, functional, user-specified relation types), which should all have their semantic interpretation described in the model.

Expressiveness of a representational system, as used here, refers to the ability of capturing the *knowledge content* of the domain in a way that the expert and knowledge engineer views as adequate for the current domain and application. This notion of expressiveness is similar to what is denoted ‘epistemological adequacy’ in [McCarthy-69]: "A representation is called epistemologically adequate for a person or machine if it can be used practically to express the facts that one actually has about the aspects of the world." This implies two critical properties of an expressive representation: The ability to express all the relevant *types of knowledge* (descriptive, heuristic, case-specific, procedural, uncertain, etc.), and the ability to express this knowledge at a sufficiently *detailed level*. The interpreters and inference procedures that operate on the knowledge structures should be explicitly defined within the representation system, according to the type of inferences that is suitable for the domain. An expressive

---

\(^1\)Even in development environments like KEE and Nexpert Object, there is no easy way to explicitly model user-defined relations, with particular inheritance properties, constraints, etc.
representation system, with appropriate inference methods, is a requirement both for competent problem solving and for automated knowledge maintenance through sustained learning.

Of course, expressiveness is not all that is needed of a knowledge representation method and system. An expressive representation language is of little use if the resulting representation is not implementable in a runnable computational model, and a minimum requirement of an implemented representation is that it is tractable. In this research, the representation problem is viewed basically as a problem of expressiveness and flexibility. Therefore, assuming this minimum requirement is satisfied, efficiency issues will not be further elaborated.

1.2.3. Sustained Learning

The problems of actively maintaining a knowledge-based system have until now received surprisingly little attention in AI, although there is a strong awareness of this problem - as illustrated, e.g., by the maintenance difficulties experienced by the XCON group [McDermott-86]. Improving manual maintenance methods is one of the motives behind a structured modelling approach to knowledge acquisition (two different approaches are discussed in [McDermott-86] and [Breuker-89]). In manual methods, as well as some automated methods for "knowledge base refinement" (e.g. the SEEK systems [Politakis-85, Ginsberg-85]), updating and refining the knowledge is typically a separate self-contained process, not a by-process of solving actual problems.

Figure 1.2 shows a basic scheme for sustained learning, i.e. learning as a natural subprocess of problem solving. The middle box illustrates - at a general level - the necessary steps: First, make an internal description of the problem, i.e. try to understand the problem by integrating it into the system's internal knowledge model. Next, use whatever type of knowledge and reasoning method appropriate to produce a satisfactory solution to the problem. Finally, retain knowledge learned from this experience, so that the same or a similar problem may be solved with less user interaction, or more efficiently, in the future.

A goal of machine learning research is to enable self-learning in computer systems. This goal is, in general, approached by developing theories and methods for processing and retaining observations (facts that are input to the system) in a way that enables them to be used in future problem solving. This is done either by inducing and saving generalized forms of the observed facts (using methods of inductive learning or explanation-based generalization) or by keeping them as concrete cases from which solutions to similar problems later may be derived (using methods of case-based reasoning or analogy). Learning always involves some kind of

---

1This trade-off problem is discussed in [Levesque-85].
generalization\(^1\). The difference is whether the generalization steps are performed before the new knowledge is stored (as in induction-based learning), or as a part of the matching process when new problems are solved (as in case-based reasoning).

A fundamental principle for sustained learning is that each problem solving experience is a powerful source of learning. A necessary criterion for the learning to be successful is that a mistake once made will not be repeated. The role of a robust competence model is to give the system a rich knowledge fundament for problem solving and learning: It will enable knowledge intensive case-based learning and reasoning by generating explanations to justify reasoning steps, extracting relevant features, retrieving relevant cases, indexing cases and deciding which parts of a case to store and which previous cases to remove.

None of today’s commercially available machine learning systems are able to continually learn from experience in real world environments. Nor are classical induction methods like Version Space [Mitchell-82], AQ [Michalski-83] and ID3 [Quinlan-86] suitable for this kind of

\(^1\)Otherwise, the 'learned' knowledge would only be applicable to solving exactly the same problems.
learning. Besides being unable to take advantage of an evolving knowledge base in their learning process, these methods also assume a classical view of concept definition. This view defines a concept as a set of attributes that are singly necessary and jointly sufficient to classify examples of a concept. Such definitions may be adequate for mathematical and other strictly formal concepts, but as shown by several authors [Wittgenstein-53, Smith-81, Schank-86a] it is extremely difficult to present classical definitions of naturally occurring concepts like 'game', 'bridge', 'chair', 'car', 'bird', etc. Smith & Medin [Smith-81] distinguish between the 'classical', the 'probabilistic' (e.g. degree of prototypicality) and the 'exemplar' (purely extensional) view of concept definitions. Non-classical concepts are defined intensionally by a set of typical properties - i.e. a set of default properties - and/or extensionally as the set of individuates, or exemplars, that are classified by the concept. Methods that try to learn non-classical concept definitions need support from an extensive domain model in order to explain why an exemplar is within the range of a concept, why some features are more relevant than others, etc.

An increasing amount of machine learning research is now concentrating on incremental methods that enable learning while solving real problems. This includes approaches for learning of generalized knowledge (e.g. Van de Velde-88b, Tecuci-88) as well as learning of specialized knowledge (Rissland-89, Porter-90). These activities have shown promising results that should encourage more intensive research into sustained learning methods for real world knowledge-based systems applications. Support for this statement is given by the results achieved within the following three research areas:

1. **Knowledge-intensive learning.** This represents a major paradigm shift that has taken place within the machine learning field during the last few years. Earlier machine learning methods addressed concept learning by induction, based on superficial similarities of features rather than on their meaning within a knowledge-rich model. Most of the research on knowledge intensive learning methods is gathered under the term *explanation-based learning*, of which an overview is given in [DeJong-88]. Knowledge intensive learning methods include purely deductive methods based on a complete domain theory (EBL/EBG, see [Mitchell-86 and DeJong-86]), methods to generate and use plausible explanations in an incomplete knowledge base [Schank-86b, Branting-88], knowledge-intensive case-based learning/case-based reasoning [Porter-86, Kolodner-87, Hammond-87, Branting-88, Barletta-88] and analogical reasoning and learning methods [Gentner-83, Kedar-Cabelli-86, Lenat-87].

2. **Case-based reasoning.** As already indicated, this represents an integrated approach to learning and reasoning. By retaining problem solving cases - as explicit descriptions containing problem descriptors, the solution arrived at, parts of the derivation path, etc. -
previous cases can be used to solve new problems. Case-based reasoning represents the major approach to sustained learning in today's machine learning research. Learning becomes a process of extracting relevant information from a problem solving experience, and indexing this case in the system's knowledge structure in a way that facilitates retrieval when a similar problem is encountered later. The case based approach to reasoning and machine learning has had a considerable growth during the last couple of years [Porter-86, Hammond-87, Rissland-87, Stanfill-88]. The first meeting of some size was sponsored by DARPA and took place in Florida in May 1988 [Kolodner-88]. Earlier work of importance to this field includes Schank’s and Kolodner’s work on memory structures for learning and reasoning [Schank-82, Kolodner-83] and the work on transformational and derivational analogy by Jaimee Carbonell [Carbonell-83, Carbonell-86].

3. Apprenticeship learning. The notion of learning apprentice systems was introduced in [Mitchell-85] as "interactive knowledge-based consultants that directly assimilate new knowledge by observing and analysing the problem solving steps contributed by their users through their normal use of the system". This does not represent a particular learning method or set of methods, rather an approach to sustained learning where new knowledge is continually acquired through observation and analysis. This approach is also well-suited to the application of semi-automatic learning methods, since an apprentice must be allowed to ask questions in order to increase its understanding (see [Murray-88a], [Murray-89] for an overview of a knowledge intensive learning apprentice system developed within the CYC environment).

These three ‘data points’ in machine learning research are complementary. They represent three important characteristics for future learning methods in knowledge based systems for practical applications. So far, very few attempts have been made to develop systems based on a methodology where these approaches to learning are integrated. An objective of the work reported here is to push this research a significant step forward.

1.3. Towards a System Design for Competent Problem Solving and Sustained Learning

The insight which changed the focus of AI research from general problem solving to explicit knowledge modelling was an important one. During the last ten to fifteen years the notion of an explicit knowledge model has been substantially extended and refined. Starting from a view of explicit knowledge as a collection of If-Then rules, it has been realized that the knowledge models which are needed to develop competent and robust real world systems, is substantially
more complex. Methods are needed to capture and utilize the various types of knowledge available: Heuristic rules as well as deeper, principled domain knowledge, generalized knowledge as well as specialized knowledge related to problem cases, and object level knowledge as well as knowledge to control the problem solving and learning processes.

Any attempt to also include learning methods into such a complex environment may seem to reach beyond a realistic level of ambition. We will argue that the contrary is the case, for two reasons:

First, the kind of systems we are aiming at may turn out to be too complex to be fully developed before they are put into operation. Therefore, methods that are able to capture a system’s experience as it is used, and improve its knowledge and behavior as more problems are being solved, may be what is required in order to facilitate a practical realization of such systems. It is a reasonable assumption that the users of data interpretation or decision support systems are more likely to accept a system with some weaknesses, as long as the system is able to improve over time. And particularly so, if the system is able to interact with the user in an intelligable way. So, incorporating methods for sustained learning into future knowledge-based systems, will help the development and user acceptance of the systems, rather than impair it.

Second, in order to examine the potential of machine learning methods for gradually improving a system’s behavior during normal operation, the methods need to address learning in a highly knowledge-intensive environment. Our learning methods should be able to take advantage of the various types of existing knowledge to improve its learning. The only way to study learning for this type of problem solving is to build integrated problem solving and learning systems within real world domains, and supply these systems with comprehensive and thorough knowledge models. The development of such systems should be based on a system architecture that enables the expert and knowledge engineer to explicitly express the various types of knowledge relevant for a particular application. This architecture should also contain problem solving and learning methods which are able to effectively utilize a continually improving body of knowledge.

The research reported here is a step in this direction. A framework for knowledge-intensive learning and problem solving has been developed, and existing approaches has been analyzed within this framework. Further, a system - called CREEK\(^1\) - has been designed, and a representation system with basic inference methods has been implemented, based on the requirements given by the framework.

\(^1\)Case-Based Reasoning through Extensive Explicit Knowledge
Compared to current architectures, CREEK suggests an improved approach to competent knowledge-based systems that continually learn from experience. Hopefully, the considerations, evaluations, suggestions, and discussions presented in this dissertation will provide a valuable contribution to further research in this important field.
Chapter 2

Overview of the Report

This dissertation contains three parts, including this introduction.

Part II presents a framework for knowledge-intensive case-based reasoning and learning, and an analysis of existing approaches by use of the framework.

Chapter 3 describes the framework. A set of requirements specifies the properties that systems must satisfy in order to fall within the framework. The requirements are related to the expressiveness of the knowledge representation, the ability of the problem solving methods to utilize a presumably thorough and comprehensive knowledge base, and the ability of the learning methods to use general knowledge to support the learning from a specific problem case. A set of characteristic knowledge dimensions is defined, in order to identify and compare significant systems properties. Based on that, a structure of representational concepts is described, aimed at capturing knowledge of the world in a representation system - at the level of thoroughness needed for the knowledge-intensive methods described later. This model will be used to analyse and compare the representational properties of current methods relevant to this research, as well as to specify a representation system for the Creek architecture.

Following the representational model, a sub-framework for describing strategies and control for problem solving and learning is outlined. The central role of explanations in supporting reasoning and learned steps within the framework is discussed, followed by generic models of problem solving, reasoning and learning, respectively. Finally, these models are joined into the framework’s integrated model of problem solving, reasoning and learning.

Chapter 4 gives an analysis of existing approaches that address the problem of knowledge-intensive, case-based problem solving and learning in real world domains. Based on a study of current research in the field, four systems representing state of the art has been chosen for
analysis using the framework described in chapter 3. Among the case-based approaches reported so far, the systems PROTOS [Bareiss-88b], CASEY [Koton-89], CHEF [Hammond-86b], and JULIA [Kolodner-87] were found to satisfy the requirements for analysis by the framework. All the systems combine case-based reasoning and learning with methods that make use of more general domain knowledge. Their strengths and weaknesses are discussed by relating them to the framework requirements, and to a set of critical factors.

Part III presents the Creek architecture, a suggested improvement over existing approaches to integrated problem solving and sustained learning.

Chapter 5 gives an overview of the Creek system design, and describes its knowledge representation platform. Creek attempts to satisfy the requirements of the framework presented in chapter 3, and could be viewed as a refinement, a specialization, of the framework's integrated problem solving, reasoning and learning model. Creek extends some ideas of existing systems, but puts more emphasis on representing and utilizing general domain knowledge in its case-based reasoning and learning processes.

An overview of the system architecture is given, in which the main structural and functional building blocks are described. Creek solves problems by combining specialized cases with generalized knowledge (conceptual model, heuristics). Generalized knowledge is used both as support for case-based reasoning, and as a separate method to fall back on if the case-based process fails to produce a satisfactory solution. Creek learns by retaining experience in the form of cases, and by interacting with the user.

The description of the knowledge representation system in Creek starts with its underlying semantics. To enable representation of various types of knowledge in a common implementational model, a procedural semantic has been chosen. Since the semantics of a knowledge structure is defined by interpreters which operate on the knowledge, the characteristics of each interpreter should be explicitly defined within the unified knowledge model.

Chapter 6 describes the problem solving and learning methods in more detail. The Creek architecture is aimed at solving diagnosis and repair type of problems1, and the problem solving and learning processes are guided by a generic model of diagnostic problem solving.

---

1Essentials of the framework and methods developed in this dissertation may be generalized to planning and design type problems. This is elaborated in the final chapter.
Most systems to date have viewed diagnosis and repair basically as two separate tasks: First finding the fault, and then proposing a repair. For many applications - such as mud diagnosis and treatment - this approach is too simple. A more realistic approach in a computational model of diagnosis-and-repair problems is to view diagnosis-and-repair as consisting of two highly inter-connected subtasks. That is, as a cyclic process that moves back and forth between diagnosis - i.e. proposing and evaluating diagnostic hypotheses, repair - i.e. proposing and evaluating repair actions, and testing - applying treatment actions and evaluating results.

Following the diagnosis and repair model, the problem solving and reasoning methods are discussed. The basic algorithm for focusing of attention and limiting of context - called goal-focused spreading activation - is described, followed by the underlying methods of the processes UNDERSTAND-GENERATE-SELECT, ACTIVATE-EXPLAIN-FOCUS, and EXTRACT-CONSTRUCT-STORE.

Chapter 7 describes Mud Creek, an example of how Creek may be used to develop and maintain a system for diagnosis and treatment of oil well drilling problems. The specific problem that has been addressed in order to focus and exemplify the design of Creek, is that of diagnosis and treatment of drilling fluid (mud) problems. A characteristic property of this problem is that the task of fault-finding is interleaved with the treatment task. A mud-engineer typically arrives at a plausible diagnosis which is confirmed or rejected after a treatment of the proposed fault has been attempted. A strategy for fault-finding guides this process, where the strength of belief in a fault hypothesis and its potential damaging impact are the most important assessment factors. Parts of a domain knowledge model is described, and an example is given of how a problem presented to this example system is solved within the system architecture described in earlier chapters.

Chapter 8 concludes the report by comparing Creek with the approaches analyzed in chapter 4, discussing strengths and limitations of the Creek design, and suggesting directions for future research.

These chapters are followed by three short appendices:

Since case-based reasoning is a relatively new problem solving and learning paradigm, a brief introduction is given in Appendix 1. Some important terms are defined, a high level process model is outlined, and two alternative storage structures for holding cases in the case memory are described.
Appendix 2 describes the representation language of Creek, illustrated by examples of how a body of knowledge is entered, modified, and retrieved. The basic frame system inference methods are also illustrated.

Appendix 3 is a glossary of basic Creek terms related to knowledge and inference.
PART II

FRAMEWORK

AND

COMPARATIVE ANALYSIS
Chapter 3

A Framework for Knowledge Modelling, Problem Solving and Sustained Learning

Frameworks for specifying methodologies and systems generally have two roles: To serve as a descriptive framework for describing, analyzing and comparing current methods, and to serve as a modelling approach for specifying a particular method or set of methods. Examples of such frameworks are Clancey's frameworks for heuristic classification [Clancey-85] and for explanations in rule based expert systems [Clancey-84a], Chandrasekaran's framework for generic problem solving tasks [Chandrasekaran-86], Gentner's framework for analogical reasoning [Gentner-83], the KADS framework for expertise modelling [Breuker-89], Minsky's representational framework [Minsky-75], Steel's componential framework of expertise [Steels-89], and Michalski's inductive learning methodology [Michalski-83]. These frameworks tend to be - to varying degrees - biased towards the modelling approach role. There are also frameworks that are developed for purely analytical purposes - intended to provide a formalism (as neutral as possible) for comparing and evaluating existing approaches according to some criteria. Mitchell's framework for incremental concept learning [Mitchell-82], Holte's framework for analyzing concept learning methods in general [Holte-88], and Bundy's framework for comparing rule-learning methods [Bundy-85] are examples of analytical frameworks.

This chapter defines a framework for specifying requirements and describing properties for highly competent systems that learn from experience. The role of the framework is to serve both as a descriptive and comparative framework, and as a modelling approach. Essential requirements of the kind of systems aimed at are inherent in the framework, in the sense that the framework itself represents a particular perspective or view of representation, reasoning and
learning. The perspective is kept at a high level, however, enabling the framework to describe various specific methodologies. The objective of the framework is

• first, to serve as a system of reference for describing and comparing contemporary research results relevant to the problem of knowledge-intensive, sustained learning through problem solving

• second, to serve as a basis for design of a methodology that attacks some significant deficiencies of current approaches

The description of the framework in this chapter focuses on its descriptive and analytical role. When modelling approach issues are discussed, it will be explicitly stated unless this is obvious from the context.

3.1. System Requirements and Framework Perspective

Since the framework will be used to describe and analyze existing methods, it should be general enough to express several representation formalisms, problem solving control methods, multi-paradigm reasoning, multi-paradigm learning, and integration aspects of combined reasoning and learning. Any framework imposes a certain view or perspective to the objects that are to be described. A purpose of the framework described here is to capture the significant properties of systems that are intended to continually learn through problem solving experience in knowledge-rich environments. Based on the background discussion in chapter 1, the following three requirements will have to be satisfied by these systems:

------------------------------------------------------------------------------------------------------------------

R1. Expressive and extendible knowledge representation formalism, enabling an explicit and thorough modelling of relevant knowledge types.

R2. Problem solving and reasoning methods that are able to effectively combine reasoning from past cases with reasoning within a competent and robust model of more general domain knowledge.

R3. A learning method that is able to retain concrete problem solving experiences, and integrate each experience into the knowledge model in a way that gradually improves knowledge quality and problem solving performance.

------------------------------------------------------------------------------------------------------------------
Expressiveness (see also chapter 1.2.2), referred to in requirement 1, is a vague term. It forms a continuum in which there are no objective evaluation criteria. The point is that the knowledge content which is needed in order to develop a thorough knowledge model, as judged by the expert/knowledge engineer team, should be explicitly expressible in a meaningful way, enabling a problem solving and learning behavior of the system that satisfies requirements 2 and 3. Expressiveness may be viewed both at the knowledge level as well as the symbol level (as defined in [Newell-82]). At the knowledge level, expressiveness is related to what type of knowledge content that may be expressed, not to how knowledge may be coded in some representation language. This notion of expressiveness therefore corresponds to the notion of epistemological adequacy of a representation formalism, as defined in [McCarthy-69]. At the symbol level, the concern is how to build computational representations which are able to capture the meaning content defined at the knowledge level. Knowledge representation languages differ in what type of knowledge they easily express. For example, disjunctive concepts are not readily expressible in some object-oriented knowledge modelling schemes\(^1\), while prototypical default properties are not expressible in first order logic\(^2\). It may be very difficult to confirm that all the relevant knowledge types are expressible before serious attempts are made to build thorough knowledge models in a particular representation formalism. An additional criteria to expressiveness, that of extendability, is therefore regarded to be important. An extendible representation formalism allows new representational constructs - and related inference methods - to be defined and explicitly described within the language itself.

Requirement 2 reflects the approach of combining concrete past experiences with more general domain knowledge, as a means to increase both efficiency and robustness in problem solving. General domain knowledge includes the (deep) conceptual model, as well as heuristic rules\(^3\). The role of the basic conceptual model within the framework is threefold:

- To guide learning of new knowledge by selecting which part of a case to store, to justify generalization steps, and to index a case for effective retrieval.

- To support reasoning based on surface level knowledge by generating explanations to justify reasoning steps. For example, to use deeper knowledge to filter relevant problem

\(^1\)Disjunctive concepts are either internal disjunctions, like "John’s car is either green or red", or external disjunctions, like "Either John’s car is green, or Mary’s bicycle is a tandem". External disjunctions are the hardest to express in standard frame-based systems. It may be done, however, e.g. by defining particular types of frames as holding disjunctive knowledge and give them special treatment by the interpreter.

\(^2\)In general, a view of knowledge as consisting of prototypical concepts, with defaults and typical values that may be overwritten by specific property values of instances, quickly leads to contradictions within a logic based system [Touretsky-87]. Developing higher order logics to cope with this problem is an active research area, with a lot of difficult problems still to be resolved [Nutter-87], [McDermott-87].

\(^3\)A deep knowledge model is the basic, definitional part of a knowledge model. A deep model may be theories or first principles, but more generally it refers to a structure of concepts thoroughly defined by their network of relations to other concepts - at the depth level appropriate for the application domain. Hence, a deep knowledge model is also referred to as a conceptual model of the domain.
PART II - Framework and comparative analysis

descriptors from the total problem description, or to justify a proposed conclusion. Surface level knowledge is typically past cases, but may also be heuristic rules.

- To be the knowledge to fall back on when problem solving based on surface level knowledge (cases and heuristic rules) fails.

The conceptual model's function in problem solving and reasoning is an extension of the role deep knowledge plays in many "second generation expert systems" architectures (e.g. [Steels-85]). Such architectures - or systems - typically combine deep knowledge with surface level rules, and the role of the deep knowledge corresponds to the latter point in the above list. A main theme in the framework presented here, however, is the additional role of deeper model knowledge in actively supporting shallow reasoning, as well as learning.

The analytic role of this framework should be viewed pragmatically: To describe existing approaches to knowledge-intensive sustained learning, and discuss their strengths and weaknesses along a set of comparative dimensions based on requirements R1-R3. The objective of the analysis is to identify strengths and weaknesses of current systems that will serve as an input to proposing an improved methodology.

3.2. Reasoning and Inference in Problem Solving and Learning

The terms problem solving, learning, reasoning, and inference have been used several times in this dissertation, but basically in an intuitive sense. Before describing the framework, a more specific definition of these terms is needed. The way the terms are used in this report implies a process structure as shown below:

![Figure 3.1: Structure of knowledge based processes](image)

The most fundamental process is Inference. Inferences are subprocesses of Reasoning and Learning methods. Reasoning - as interpreted here - is a subprocess of Problem solving (see comment in the text), while Learning method is the corresponding subprocess of Learning.
Problem solving is a process that takes a problem description, a goal, and a knowledge base as input, and derives a solution that satisfies the goal. The goal contains a specification of the requirements that must be fulfilled in order for a result to be a solution to the problem. A problem may be structured as sub-problems, in which case the problem solving process may be correspondingly split into sub-processes. The ability to solve problems is part of an agent's expertise, incorporating strategies (plans) as well as lower level reasoning methods.

Learning is a process that takes an existing problem solving system and new information as input, and derives a system where the quality and/or efficiency of problem solving is improved. To learn may, of course, be regarded as a problem in its own right, and problem solving could be said to subsume learning. To avoid confusion, problem solving will here solely refer to solving application problems.

Reasoning is a process that take some facts and a goal as input, and derives a result by applying one or more inference methods to a body of knowledge. Reasoning may be regarded as a more general term than problem solving, in the sense that its input and output may be any kind of information, not necessarily a problem description and a solution. Reasoning is also a sub-process of problem solving, since it is used to derive results that contribute to solving a problem. The term reasoning is most often used in connection with problem solving, and characterizes parts of the problem solving process, as in rule-based reasoning, model-based reasoning, case-based reasoning. Being a general process of deriving a result through inference, reasoning is also involved in some learning processes - and particularly in knowledge-intensive learning. In the machine learning literature, however, terms like learning method or learning algorithm are more frequently used when talking about particular sub-processes of learning, reserving the term reasoning to the problem solving process. In this dissertation, it should in general be assumed that reasoning refers to the problem solving process.

Reasoning methods, based on the three reasoning types listed above, may be put together to form more complex structures for deriving particular types of results, i.e. for performing a task. A reasoning model relates reasoning types to their roles in achieving tasks.

Inference denotes the lowest level of processes in the hierarchy, i.e. the building blocks for reasoning and learning. The inference methods define the primitive operations on the knowledge, and hence: the basis for its semantic interpretation. There are three basic types of inference processes:
• **Deduction**, which is a truth preserving inference process where an inference step typically is a logical derivation of a theorem from a theory that is assumed to be both consistent and complete. Deduction is based on a universal inference rule, like *modus ponens* - which may be expressed in the following form:

\[
P(a) \& (P(x) \rightarrow Q(x)) \rightarrow Q(a)
\]

A classical example:

\[
\text{Isa-man}(Socrates) \& (\text{Isa-man}(x) \rightarrow \text{Is-mortal}(x)) \rightarrow \text{Is-mortal}(Socrates)
\]

As an inference rule of first order predicate logic, deduction guarantees the truth of the conclusion given the truth of the premisses. But the requirements on the knowledge as a non-contradictory set of theorems are often hard to meet in real world contexts.

• **Abduction**, also called *inference to the best explanation*, is a kind of reverse deduction. A simplified way to express abductive inference is:

\[
Q(a) \& (P(x) \rightarrow Q(x)) \Rightarrow P(a)
\]

Example:

\[
\text{Has-abdominal-pain}(Socrates) \& (\text{Has-appendicitis}(x) \rightarrow \text{Has-abdominal-pain}(x)) \Rightarrow \text{Has-appendicitis}(Socrates)
\]

An abductive inference does not guarantee the truth of the outcome given the truth of inputs. There may be a lot of reasons for abdominal pain, appendicitis is only one possibility. An abductive conclusion therefore has to be supported by an explanation that leaves the particular conclusion as the best choice. Abduction is also called *plausible inference* (e.g. in [Charniack-84]), and is closely tied to generation and evaluation of explanations.

Abduction is considered to be the kind of inference typically made in diagnosis (at a top level, at least) where a set of symptoms infers a fault by being explained by the fault, i.e. by generating a plausible argument that supports a particular fault and weakens competitive fault hypotheses.

---

*The => operator denotes a non-deductive reasoning step, while a -> symbol denotes a logical implication.*
• **Induction** is similar to abduction in that the truth of the premisses does not guarantee the truth of the conclusion. An inductive conclusion should therefore also be supported by a justification. An inductive inference step may be expressed as:

$$ (P(a) \rightarrow Q(a)) \land (P(b) \rightarrow Q(b)) \land \ldots \Rightarrow P(x) \rightarrow Q(x) \quad \text{for all } x $$

**Example:**

$$(\text{Isa-bird(Robin)} \rightarrow \text{Can-fly(Robin)})$$

$$\land (\text{Isa-bird(Eagle)} \rightarrow \text{Can-fly(Eagle)})$$

$$\Rightarrow (\text{Isa-bird(x)} \rightarrow \text{Can-fly(x)})$$

Inductive inference is particularly related to learning of general knowledge from observing examples (instances) and counter-examples.

Deductive inference is a strong method, but its applicability for reasoning is limited by assumptions that do not generally hold for real world situations: A world - typically assumed to be closed - where all phenomena (all properties of concepts) are either true or false, and a consistent domain theory by which any proposed fact (i.e. proposition) may be proved to be true or false. However, even if knowledge in general does not fit the requirements of deductive reasoning, some parts or types of knowledge may do so, in which case much is gained if deductive methods can be applied to that subset.

Abduction and induction are fundamentally different from deduction, since they do not guarantee that the inferred result is true. The results therefore have to be justified or supported by other means, for example by 'lookup' in other parts of the knowledge base or via other inference processes leading to the same result. The results are to be regarded as plausible, rather than absolutely true. The degree of plausibility may be determined by calculating probabilities, or by more knowledge-rich and context-dependent methods that support a result by generating explanations. The advantage of inductive and abductive inference methods is that they are not necessarily subject to the limitations of first order logic: The world may be viewed more realistically as an open, dynamically changing environment. Concepts may be described in terms of typical features or default values which may be inherited to more specific concepts, and replaced by local values if these later become available. A disadvantage is that the particular inference methods needed has to be defined by the system developer, leading to a complex and often unclear definition of knowledge semantics. (The issue of deductive vs. non-deductive inferences in reasoning is given a deep and thorough discussion by the collection of papers in [McDermott-87]).

Based upon the three **inference types** described, a range of **inference methods** have been developed in AI. An inference method specifies a way to derive new information from existing
knowledge in an actual representation system. Examples are specific methods for chaining of rules, for inheritance of properties in network hierarchies, for matching of concept schemas, etc. Inference methods may be put together to form inference structures. A goal-oriented, complex inference structure (like Clancey’s inference structure for heuristic classification [Clancey-85]) will be here be referred to as a reasoning model.

3.3. Descriptive dimensions

The basic structure and components of the framework is reflected in a set of major dimensions, or aspects, that cut the space of methods into characteristic subspaces, suitable for description and analysis. Based on the requirements R1-R3 (section 3.1), the following five dimensions - with data points as indicated - will be used to focus the description of selected approaches:

1. The problems addressed
   • Type of problem to be solved
   • Type of learning

2. The knowledge model
   • Knowledge types
   • Representational terms
   • Representation of general knowledge
   • Structure of explanations
   • Representation of cases
   • Integration aspects

3. The problem solving process
   • Problem solving strategy
   • Problem solving tasks

4. The reasoning
   • Reasoning methods
   • Reasoning structure
   • Control of multi-paradigm reasoning

5. The learning
   • Learning process structure
   • Case learning
   • General knowledge learning
   • The user’s role
   • Control of multi-paradigm learning
First, the problem solving and learning tasks addressed by the approach needs to be identified. What kind of problem is to be solved, and what are the problem solver's external constraints - like user interaction and coupling to other system parts? What kind of questions is the problem solver supposed to answer? And for the learning task: What is to be learned - concept definitions, heuristics, or strategies? What are the external constraints on learning?

As pointed out before, the work presented in this dissertation is motivated by a "very knowledge-intensive" view to the role of general domain knowledge in problem solving and learning. An important dimension for characterizing a particular approach is the structure and contents of the knowledge model, including the general domain knowledge as well as experienced cases. The ability to build a strong knowledge model depends not only on the properties of these two components, but also on how these two submodels are interconnected and integrated. Important characteristics of the knowledge model are its representational uniformity, expressiveness, flexibility and extendability, transparency (user understandability), and the underlying inference methods assumed.

The third dimension characterizes the problem solving process - at a high level. What strategy - or strategies - exist for attacking a problem presented to the system? What type of tasks (subproblems) exist and how are they related?

The next dimension describes the reasoning model. What reasoning methods exist, and how are they combined and related to problem solving tasks? If both rule-based and deep model-based reasoning is combined with case-based reasoning, their individual roles and how the combination of several reasoning paradigms is controlled, are significant characteristics. Since general domain knowledge will be used to justify and support reasoning steps based on specific case knowledge, the structure and actual role of explanations is also an important system descriptor.

Finally: The dimension characterizing the learning model. Even though case-based reasoning is a recent research direction of machine learning, several methods have been developed. They vary in the type of knowledge that is contained in a case, how cases are indexed for later retrieval, whether and how case knowledge is generalized before saving, how general domain knowledge is used to support the learning process, etc. Another important matter is to what degree the system user is allowed to - or is required to - interact with the learning process.

To enable a unified description of different systems and methodologies along these dimensions, the framework must define a basic set of descriptive terms. The representational platform

---

1Uniformity in representation refers to whether one single, or several different, representation formalisms are used.
described in the following section describes a general component model of expertise, and a
model of primitive terms for describing knowledge representation schemes. The
representational platform is primarily a basis for expressing properties of knowledge models
(dimension 2). The framework’s models for describing and analyzing problem solving,
reasoning and learning methods are presented in the subsequent sections.

3.4. Representational Platform

A requirement of the framework is to enable the combination of different types of reasoning
and learning within a rich knowledge model. Knowledge representation systems for complex
representation tasks are still a heavy research area in AI. Theoretical work on knowledge
representation has to a large extent been focusing on particular representation problems, or
constrained by a restricted view of knowledge level reasoning and representation. Examples of
the former are spatial representation models [Waltz-75], and representation of time relations
[Allen-83]. Examples of the latter are work based on first order logics and its extensions
[Brachman-83, Genesereth-87]. More pragmatic and expressive approaches to knowledge
representation are represented by the KRL [Bobrow-77], RLL [Greiner-80], KODIAK
[Wilensky-86], KRS [Van Marcke-88] and CYC [Lenat-86, Guha-90] systems.

Our representation needs to be able to integrate different knowledge types and multiple
interpretation methods, while ensuring a common consensus. This suggests that an approach
based on first-order logics is too limited. Knowledge in real world, natural domains is
characterized by various degrees of typicality - from mere defaults to necessary values.
Concepts are assigned properties with values based on the information available at a certain
point in time, and these values may be overridden when more information becomes available.
Viewing the world as consisting of truths and falsities only, and viewing reasoning as theorem
proving, is a description at a level far from the level of knowledge content we want to
represent. Hopefully, research efforts into higher order logics will eventually lead to
theoretically sound representation formalisms at the level of expressiveness and knowledge
content needed to model complex real world domains.

An interesting theoretical framework for merging formal and informal methods is the
conceptual graphs formalism of Sowa [Sowa-84]. Unfortunately, this formalism is well
developed only at basic, detailed levels of representation, reasoning and inference, while our
framework needs a unified model of knowledge, reasoning and learning methods at higher
levels - including conceptual models, problem solving strategies and reasoning tasks.
Existing models and on-going research relevant to specific parts of the framework will be discussed along with the respective framework parts. Two models define the scope of our representational framework: a top-level model of terms to describe expertise - including a submodel of knowledge types - and a bottom-level model of basic representational primitives:

1. **A model of expertise**

   Expertise is knowledge in the broadest interpretation of the term, and includes factual knowledge of a domain, problem solving strategies and methods, and learning strategies. In addition to a set of primitives for describing various types of knowledge, an expertise model needs to define a set of strategies and reasoning processes that operate upon and utilize conceptual knowledge, past cases, and heuristic rules. Expertise is knowledge, and knowledge about how to use and maintain it. In this interpretation, expertise may be regarded as a combination of problem solving competence and learning ability.

2. **A model of representational primitives**

   Every knowledge representation system has a set of terms that form the foundation for the representational vocabulary. They define representational terms like "symbol", "concept", "entity", and "relation". Each term has its own semantical interpretation which is defined in the program code that implements the representation system. Such a primitive model is not always explicitly defined, but it is important that this be done in order to achieve the degree of explicitness in knowledge modelling required of the framework described here.

In the following sections, a representational platform for thorough knowledge modelling, for integrating problem solving and learning, and for combining deep model and rule-based reasoning with reasoning from past experiences, is presented.

3.4.1. Modelling of Expertise

A competent and robust expert system, which is able to learn from experience, must contain a thorough and explicit knowledge model of its domain of application. The quality of this knowledge model is crucial to the overall performance, user friendliness and learning ability of the system. This is consistent with the symbolic AI principle, where reasoning is a process of manipulating symbols that explicitly represent phenomena in the world. The knowledge modelling perspective presented is, thus, fundamentally different from approaches such as, e.g., neural networks, where a concept’s description is distributed among several network nodes connected by weighted exhibitory and inhibitory links [Rumelhart-87]. Within that approach
knowledge is represented implicitly as an activation pattern of nodes connected by links with
different strengths, i.e. as a non-symbolic, distributed concept definition1.

Every knowledge representation system will necessarily impose some constraints on the kind of
knowledge that is expressible. A high degree of expressiveness generally also requires more
complex reasoning and inferencing schemes, with decreased run-time speed as a consequence.
The stance taken in the research reported here is that expressiveness in representation should
not be sacrificed for optimization of performance speed, as long as the resulting systems run
within acceptable speed limits2. This is motivated by the current state of knowledge based
systems research, where much more research is needed on modelling of knowledge and on the
utilization of thorough knowledge models in AI systems (see, for example, [Lenat-87, Clancey-
89]). The AI community need to show the feasibility of computational models of problem
solving and learning which are qualitatively acceptable. Once this is achieved, optimizing
performance speed may be done in alternative ways, where constraining the representation
language is one option. Another is to use learning methods to optimize performance - either by
working on the knowledge base as a whole, or by incremental methods. Improvement of
performance efficiency is a heavy research area in machine learning [Diettrich-86], and a
particular focus of some similarity-based as well as some explanation-based and case-based
learning systems (e.g. [Mitchell-86, Laird-86, Koton-88]).

3.4.1.1. Knowledge Types

A given piece of knowledge may be classified by its position on a set of knowledge
dimensions. The choice of knowledge dimensions for the framework described here reflects
important aspects of knowledge, from the perspective of knowledge-intensive problem solving
and learning. Five significant dimensions are defined (figure 3.2):

1. Knowledge level refers to whether the knowledge describes concepts of the application
domain - called object knowledge - or the use and control of the object knowledge. The
latter type is called control knowledge. The control level is a meta level with its own
complex substructure, incorporating domain dependent problem solving strategies as well
as more basic inferencing and reasoning control. Search control refers to controlling the

---

1 Neural nets, in this sense, should be regarded as an extreme type of what is generally referred to as connectionism. All
connectionist methods share the basic computational model of nodes with threshold values that may or may not 'fire' other
nodes, depending on the type and strength of their connections. Learning in connectionist networks is a process of automatically
reducing or increasing strengths of connections and/or thresholds of nodes. However, some connectionist approaches let the
nodes represent explicit domain concepts; such approaches are often called sub-symbolic approaches (e.g. [Smolensky-87],
[Thagaard-89]).

2 What is acceptable in terms of performance speed will be application dependent. The principle of representational
expressiveness suggests that performance speed issues should be set aside for a while, putting more effort into the modelling of
knowledge content. A minimal requirement on performance speed is, of course, that the reasoning and inference methods
performed within the knowledge model are computationally tractable.
low level system-internal search procedures like depth-first or best-first search, the propagation of activation through rules or along specific relations, etc. Strategies should be interpreted in a general sense as controlling the sequence of actions taken to solve a problem (as described in [Gruber-88]). Strategic knowledge of medical diagnosis, for example, includes knowing when to do examinations and what symptoms to look for, when to generate hypotheses and when and how to evaluate them, etc.

Figure 3.2: Dimensions of Knowledge
The five dimensions for characterizing knowledge, described by specialising into subclasses. The partitioning of level and role are complete, while only examples of subtypes are given for form. The subtyping of depth should be viewed as a continuum, ranging from deep conceptual model, through less deep models that may involve 'compiled' heuristic knowledge and cases, up to shallow associations that link input features to solutions. Generality is also a continuum, with general class definitions at one end, and a particular solved case at the other end of the spectrum.

2. Knowledge depth refers to a scale from fundamental, theoretical, deep model or 'first principles' knowledge, up to readily applicable rules of thumb or concrete past experiences that directly associate a solution with a set of problem descriptors. What is often called "model knowledge" in the literature typically refers to deep knowledge. Davies' electrical circuit model [Davis-82a], Kuiper's medical diagnosis model [Kuipers-87] and Hayes' liquid model [Hayes-85] are typical examples, although they are quite different approaches to deep modelling.
3. Knowledge generality classifies knowledge according to how large a space of the real world it describes. Conceptual model knowledge is typically general knowledge, since it contains class definitions, described by general relationships with other classes. A more shallow type of general knowledge is typically held within heuristic rules. Conceptual models may also describe more specific knowledge, however, as instances of definitional classes. An even more specific type of knowledge is the knowledge contained in past episodes and concrete experiences, for example previously solved cases. Case knowledge describes a particular instance of an association, or an action sequence, performed within a particular range of time.

4. Knowledge role is a dimension related to the use of knowledge. It splits into two subclasses: Descriptive knowledge - defining conceptual structures, stating established facts and assumptions - and operational knowledge, like a direct association between an input descriptor and a goal concept, or an action sequence that may be activated in order to reach a particular goal. Descriptive knowledge may be deep knowledge contained in an underlying conceptual model, or more shallow associations. For example, a relationship expressed by the relation 'is-implied-by' is considered more shallow than one using the relation 'is-caused-by'. Knowledge as heuristic rules and previously solved cases may be viewed as descriptive as well as operational. This type of knowledge is here classified as operational, however, since its role typically is to directly associate problem descriptors with (sub)goals and solutions. Operational knowledge may also be plans, methods or action sequences in the form of procedures.

5. Knowledge form characterizes the syntactic form of the knowledge. This dimension is close to the actual representation system used to capture the knowledge, but it does not describe the representation language per se. A concept network, for example, may be represented as Horn clauses in Prolog or as frames in a Lisp based frame system. Rule as a knowledge form is different from a heuristic rule, since the latter says something about knowledge content, not necessarily about form. The same argument applies for case and solved case, although this distinction is more subtle.

The first four dimensions describe the meaning content of knowledge. The distinctions between control level and object level knowledge, between shallow and deep, general and specific knowledge, and between descriptive and operational knowledge, are aspects of knowledge semantics and pragmatics, not of representational formalism. Explicating these dimensions enables a knowledge modelling approach that attacks major problems of today’s knowledge based systems, by:

- Explicitly modelling problem solving and learning strategies, which enables a system to
reason about strategies as well as object knowledge
• Combining shallow and deep knowledge, which may be present both at the control level and at the object level (although control level knowledge typically is shallow)
• Problem solving by combining previous experiences with more generally applicable knowledge
• Relating knowledge to its role in problem solving and learning, extending the notion of knowledge to include methods and procedures as well as concept schemas and heuristic rules

Related work on splitting the knowledge space into dimensions, according to some purpose, is abundant in the knowledge representation literature (e.g. several articles in [Brachman-85]). However, this typing of knowledge is often limited to one or two dimensions (e.g. object/control or deep/shallow), or tends to be implicit in the description of a representation language or an application. Some review studies, however, present more comprehensive descriptions, for example Jansson's work [Jansson-86], from which the figure below (figure 3.3) is reproduced.

\[\text{Figure 3.3: Some Representational Dimensions}\]

Nine significant dimensions of concept representations, based on the discussion of important aspects of representation in [Jansson-86]. In the figure, an arrow points to a dimension name, while a name crossing an arrow indicates a data point on that dimension.

Comparing the dimensions in figure 3.3 to our model in figure 3.2, the major contrast is the specificity (detail level) of the dimensions. Some dimensions shown in figure 3.3 are in our framework regarded as aspects of single concepts, not characterizations of knowledge in general. Some of these aspects are included in the model of representational primitives, described in section 3.4.2 (e.g. 'relational type', 'meaning type'). What is called 'concept generality' in figure 3.3 has a different meaning than our generality dimension, and is regarded as types of concept definitions in our framework. Certainty is regarded as a property of a
concept’s value, while problems involving the type of modality indicated in the figure (i.e. knowing vs. believing) are not explicitly addressed by our framework.

The distinction between declarative and procedural knowledge is deliberately avoided in the present discussion, since these terms are used ambiguously in the literature, leading to a distinction which is often vague and unclear: The distinction between "knowing what" (declarative knowledge), and "knowing how" (procedural knowledge)\(^1\) - may be clear at some level of interpretation of knowledge, as exemplified by the following two statements:

- my car is green
- painting a car is a sequence of cleaning the car, covering parts not to be painted, ...

The first sentence is stating a fact, a description of a situation, while the other sentence is describing a procedure, a sequence of actions to achieve something. However, at a more detailed level of knowledge representation and interpretation, like the level of inference methods that an intelligent agent applies in order to interpret represented knowledge, the declarative-procedural distinction is unclear and still a matter of dispute within the AI community\(^2\) (see the collection of articles in [McDermott-87]). Within this perspective, the notion of declarative knowledge is particularly related to logical based representation formalisms, where a declarative representation (in the form of logical expressions) is viewed as identical to an explicit representation of knowledge, and procedural knowledge is defined as the knowledge implicit in procedures\(^3\). The notion of declarative semantics is often used synonymously to model theoretic semantics of first order logics (e.g. in [Genesereth-87]). This is based on a view that all knowledge in an intelligent computer system should be represented declaratively (in the logical sense) and interpreted by logical proof-procedures (e.g. resolution) based on deduction. Of course, all explicitly represented knowledge needs to be interpreted by an underlying procedure. An argument from a non-logicist point of view, then, is that proof methods based on logical deduction are just one type of knowledge interpretation procedure (others are, e.g., abductive inheritance methods, probabilistic inference, analogy matching). A declarative semantic is just one type of semantic, tied to one type of interpreter, with some strengths and some limitations. Thus, logical deduction - as well as any other interpretation procedure - is suitable for interpreting certain kinds of knowledge, but unsuitable for other kinds.

---

\(^1\)This distinction between procedural and declarative knowledge is central to some psychological models of problem solving and learning, e.g. Andersson’s ACT system [Andersson-83].

\(^2\)What has been called the declarative-procedural controversy in AI concerns whether the knowledge people have and use for problem solving is basically “what” or “how” knowledge. Winograd [in Winograd-75] proposed a kind of frame representation system that synthesizes the two into a single representation system.

\(^3\)This is obviously a different interpretation of the terms than the “knowing what - knowing how” distinction illustrated by the car example.
Within the framework described here, the distinction between declarative and procedural domain knowledge - interpreted as in the car example above, is captured through semantic primitives of the domain, defining concepts like process, event, state, physical-object, etc. The role dimension (see figure 3.2) - distinguishing between descriptive and operational knowledge - captures the distinction between definitional, factual knowledge on one hand, and methods, procedures, and strategies for interpreting and using the descriptive knowledge, on the other.

3.4.1.2. From Knowledge to Expertise

One of the current trends in knowledge modelling is to extend the traditional notion of knowledge to incorporate reasoning methods and problem solving strategies as well. Different subtypes of problems in a domain often need different strategies and reasoning methods to be successfully and efficiently solved. Strategies for how to solve different types of problems, as well as characteristics of different reasoning methods, then become a part of the explicitly represented model. To distinguish this broader interpretation of knowledge from the more traditional one, it is sometimes referred to as expertise.

As argued in chapter 1, the ability to learn from problem solving experience is regarded as an essential property of future knowledge based systems. Human learning is to a large extent an automatic mechanism, but experts also acquire knowledge about how to learn, which may also be used deliberately to improve learning. This knowledge is used together with domain knowledge to decide, for example, which parts of an experience are worth remembering, and whether some feasible generalizations should be made. Competent and robust expert systems need a computational model of expertise that incorporates conceptual domain knowledge, problem solving strategies and learning strategies.

Figure 3.4 shows the three main components of an expertise model: A knowledge fundament describing the domain, a control level model of problem solving strategies (diagnosis and repair), a structure of tasks and subtasks, a reasoning model containing a set of reasoning methods and a reasoning structure (which describes the sequencing of tasks), and a learning model that integrates explanation-based and case-based methods. The definitional knowledge model contains object-level as well as control level concepts. How the different types of definitional knowledge are used is specified at the control levels (problem solving and learning). The definitional knowledge is associated with several inference and reasoning methods for interpretation of its semantical content. For example, a hierarchical conceptual model may have concepts defined by typical properties, and base its interpretation of the knowledge upon default inheritance along class/subclass-instance relations. A case model is typically interpreted by a certain case-based reasoning method, etc. The framework enables an
explicit representation of the submodels shown in the figure. Given the task of developing a particular expert system, it

![Component Model of Expertise](image)

**Figure 3.4: A Component Model of Expertise**

The figure shows components of the expertise model for diagnosis-and-repair type problems that underlies the framework. The definitional model contains general conceptual knowledge and object level cases and rules. The problem solving and the learning models are control level models (see fig. 3.1).

is not always the case that all the submodels need to be represented explicitly. The guiding principle for decision should be the following:

*all knowledge to be reasoned about is to be explicitly represented.*

A strategy is a plan for how to achieve a particular goal. A task is a type of subproblem, which may be specified at different levels and scope of generality - from high level tasks like diagnosis, to low level tasks like constructing a generalization from a set of instances. A diagnosis-and-repair strategy, together with the corresponding task structures and reasoning types, defines what will be called a *diagnosis-and-repair model*. This is a generic model of problem solving which guides the knowledge modelling, problem solving and learning within this type of domain. Such a model typically contains control level knowledge about choosing
the right action at the right time, as well as an underlying concept model (part of the definitional knowledge model) that defines concepts like "symptom", "hypothesis", "measurement", "diagnosis", etc.

The extension of the knowledge concept towards expertise, in the sense sketched here, is a consequence of recognizing problem solving and learning methods as knowledge to be modelled and represented, just as factual, object level domain knowledge. In addition to the structural component model of expertise shown in figure 3.4, a functional model - a methodology for integrating and controlling the submodels - is needed. In the KADS project [Breuker-89, Wielinga-86] an approach to modelling of problem solving expertise has been developed, based on the four layers of expert knowledge shown in figure 3.5. This model is adopted - with a few modifications - in our framework:

The *domain layer* in the KADS model contains definitional knowledge of the domain. This corresponds to what is called *object level* in figure 3.2. The term object level (or layer) is preferred to domain layer since the three other layers typically also contain domain dependent knowledge. In KADS, most of the object level knowledge is task independent (general concept definitions, general facts, general constraints), while in our model parts of this knowledge will typically be task dependent (e.g. context dependent concept definitions, procedures, cases and rules with particular roles).

<table>
<thead>
<tr>
<th>layer</th>
<th>relation</th>
<th>objects</th>
<th>organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>strategic layer</td>
<td>controls</td>
<td>plans, meta-rules, repairs, impasses</td>
<td>process structure</td>
</tr>
<tr>
<td>task layer</td>
<td>applies</td>
<td>goals, tasks</td>
<td>task structure</td>
</tr>
<tr>
<td>inference layer</td>
<td>describes</td>
<td>meta-classes, knowledge sources</td>
<td>inference structure</td>
</tr>
<tr>
<td>domain layer</td>
<td></td>
<td>concepts, relations and structures</td>
<td>axiomatic structure</td>
</tr>
</tbody>
</table>

*Figure 3.5: Layers of Expertise in KADS*

The figure is reproduced from [Breuker-89]. See surrounding text for explanation.

The inference layer describes the structure of reasoning and inference mechanisms that operate upon the domain layer, and covers both what is called *reasoning* and *inference* in our framework. The inferencing expertise is contained in a set of inference making functions, called
knowledge sources, like abstraction, specialization, and matching. Inference functions are combined to form inference structures (i.e. reasoning models), which are controlled by task structures in order to accomplish particular tasks (i.e. achieve particular goals). The role of domain knowledge in inferencing is described in KADS in what are called meta-classes. Meta-classes correspond to problem solving concepts such as "hypothesis", "evidence", "conclusion", etc., i.e. concepts that are part of the control level diagnosis-and-repair model in our framework.

While the inference layer describes which inferencing functions are available, the task layer explicates knowledge about when a certain inference is to be made. The inference methods are related to what task or goal (subtask or subgoal) they may contribute to solving. As an example, a high level inference structure for the task called systematic diagnosis - with a corresponding task structure - is shown in figure 3.6.

![Figure 3.6: Inference- and Task-structures for Systematic Diagnosis](image)

An example that illustrates the difference and interconnections between an inference structure and a task structure, according to the KADS methodology. The figure is reproduced from [Breuker-89].

The strategic layer sets up the plan for achieving a goal, thereby controlling the reasoning in the task layer.

---

Generic inference structures related to classes of generic tasks are called 'interpretation models' in KADS (their role in knowledge modelling is thoroughly discussed in [Breuker-87]).
The four-layer model in KADS is developed as a model of problem solving expertise. Hence, it differs from our expertise model in that learning competence is not incorporated. This is the traditional view of expert knowledge, but viewing learning competence as a kind of knowledge seems the right approach when integrated methods for problem solving and learning are aimed at. However, the generic nature of the four-layer model makes it suitable to describe learning competence as well.

![Four-Layered Expertise Model](image)

**Figure 3.7: Integrated Four-Layered Expertise Model**

The figure shows the four-layered expertise model of our framework, described within a slightly modified version of the KADS model.

As illustrated in figure 3.7, the KADS four-layer model may be used as a basis for modelling learning competence as well as problem solving competence - by integrating separate problem solving and learning knowledge into a common knowledge model. This model may be regarded as the top level of the expertise component model in figure 3.4, viewed from a different perspective. The object layer in our model basically consists of knowledge in the form of conceptual models, past cases and heuristic rules. Task independent knowledge is a basis for problem solving as well as learning. At the inference level, most methods will be general - applicable to problem solving and learning alike, some will be specific for certain kinds of problem solving and learning methods. Problem solving tasks and learning tasks are different, and will have different task structures. The same is true for strategies, but a top level strategy is also needed, describing how to combine problem solving and learning - e.g. when to solve the problem (a subproblem), and when to (partially) learn. By sharing some knowledge structures between the learning and the problem solving model, both learning and problem solving
competence can be described by the four-layer model. One may view this as two separate models integrated into a common knowledge (expertise) model, as shown in figure 3.7.

The figure illustrates a merge between the expertise component model (figure 3.4) and the four-layered knowledge model (figure 3.5). The perspective is changed, since the definitional knowledge model (left branch of the expertise model) may define knowledge at all four layers (object concepts, inference structure concepts, etc.). The problem solving and learning models of figure 3.4 corresponds to control level knowledge in the model shown above. As pointed out in [Breuker-89], the inference and task layers may be regarded as intermediate layers linking object level knowledge to strategies.

3.4.2. A Model of Representational Primitives

The expertise model described in the previous section represents the top level components of the framework. In this section, the framework's conceptual model of low level terms for describing knowledge is presented.

The model described here is a model of representational concepts, not a model of real world objects. A representation system may be broadly viewed as modelling concepts in two planes:

1. Plane of application domain, where the knowledge objects are domain-related concepts (e.g. viscosity, drilling-speed, temperature, measurement, possible-fault)

2. Plane of representations, a meta-plane where the knowledge objects are representational concepts (e.g. symbol, concept, entity, relation, value)

The model of actual objects of the real world\(^1\) resides in plane 1, while the model of representational primitives is a meta-plane model that lies in plane 2, as illustrated in figure 3.8. Plane 1 in general comprises two different views of world knowledge - an intensional and an extensional view - corresponding to the two subplanes in the figure\(^2\) (the extensional subplane is gray). The upper plane corresponds to what Brachman calls the epistemological level of semantic network representations (in [Brachman-79]), i.e. a "structure of conceptual units and their relationships as conceptual units independent of any knowledge expressed therein". The purpose of identifying plane 2 as a separate type of knowledge, is to be able to develop an explicit model of representational concepts.

\(^1\)More precisely, what is being represented is the modeller's view of the real world. Hence, objects and concepts refer to real world objects as perceived and understood by the person(s) doing the modelling.

\(^2\)The concept planes may be further divided into subplanes and superplanes depending on modelling perspective. see [Bergheim-88] for a thorough discussion of different aspects of concept planes and meta-planes.
Conceptual primitives for describing knowledge at level 1 are often referred to as semantic primitives. A model of semantic primitives constitutes the foundation of an ontology of the world. Aristoteles, for example, identified the following semantical primitives: Substance, Quantity, Quality, Relation, Time, Position, State, Activity, Passivity. While the representational primitives describe types of representational constructs, the semantical primitives are building blocks for modelling the real world. By linking domain concepts to representational concepts, a system is able to use the explicit model of representational concepts in its understanding of the domain.

Figure 3.8: Representations vs. World Objects
A pictorial view of the relation between representational concepts and concepts of the real world. A concept in the representation plane is a type of descriptor for concepts of the real world. Real world concepts may be defined intensionally (as classes that describe properties of their instances), or extensionally (as sets described by the collection of properties of their members). Our model of representational primitives lies entirely within the upper plane.

All representational theories and formalisms impose a certain view on what a concept is, what a relation is, etc., but this is often not stated as explicit models of representational concepts. In the work of Jansson on representation of taxonomies [Jansson-86], several explicit submodels of representational concepts are described. Parts of this work will be briefly described, and subsequently compared and contrasted with the representational model of our framework.
Jansson’s work is a thorough analysis of relations for representation of generalization/specialization hierarchies. Its purpose was to study issues of taxonomical representations related to information systems design, but the scope of the work is of a more general nature. The work is a presentation and discussion - within a gradually evolving descriptive framework - of different approaches to representational issues that have been reported in the literature. This summary highlights the parts of the descriptive framework that address definitions of basic representational terms and structures\(^1\). Representational constructs of particular relevance to information system modelling languages, and programming languages, are left out.

Taxonomical relations are defined to be of two types:

- those connecting two intensional concepts, or
  - a particular object to an intensional concept: \textit{specialization-of} and \textit{instance-of}
- those connecting extensional concepts: \textit{subset-of} and \textit{element-of}

Specialization-of is identical to what is called \textit{subclass-of}\(^2\) in this report, and the inverse relation, \textit{generalization-of}, is identical to \textit{has-subclass}. Subclass-of/has-subclass is used here, since they are regarded as being more precise (specialization is often taken to subsume subclass as well as the other three relations in the list above).

Jansson gradually introduces a language - a structure of representational terms - for discussing issues related to taxonomies:

The basic term for talking about phenomena in the world is the \textit{category}. Following common practise in cognitive sciences, a category is defined extensionally as a collection of objects that share significant similarities. A \textit{concept} is the corresponding intensional interpretation of an object class. The basic representational unit is the \textit{entity}. An entity has the subclasses \textit{event}, \textit{event structure}, and \textit{relation}.

Relations are typed according to the subclass-hierarchy shown in figure 3.9. The \textit{decomposition relation} has subtypes with different representational roles: \textit{Part} relates a concept to its structural parts (the cat has three legs). \textit{Property} has the role of an adjective (the cat is grey). \textit{Relationship} expresses a relation between instances of two concepts (the cat hates the dog). The distinction between property and relationship is based on a distinction between \textit{primary} and

\(^1\)Structure in Jansson’s report corresponds to what is called \textit{model} in this dissertation.

\(^2\)The interpretation of subclass-of is based on the notion of a \textit{class} as representing an intensional concept. The corresponding extensional term is often called \textit{category}, described by a collection (set) of individuals (see for example Smith-81).
secondary objects. Primary objects are explicitly defined as part of the domain description, while secondary objects (e.g. grey) are labels with no corresponding concept definition.

The specialization of event relation is influenced by linguistic approaches to knowledge representation, where the notion of a case relation is central to the structure of events expressed in a sentence. Jansson proposes the cases of Sanskrit as a basis for representing events:

 NOMINATIVE active agent ABLATIVE source of action
 ACCUSATIVE object for action LOCATIVE place for action
 INSTRUMENTALIS tool, collaborator GENITIVE ownership
 DATIVE target for action VOCATIV address

(These particular cases are semantic primitives rather than representational primitives, but are included to illustrate the notion of a case in this sense.)

A relation is expressed in terms of a set of facets, where each facet has one - or a set of - values. A value may be a single symbol, or a complex symbolic structure. Symbols are pointers to other descriptions. The facets of a relation describe its semantics by characterizing the type of values that define the relation. The figure below shows the classification of what is referred to as a standard set of facets:

![Figure 3.9: A Taxonomy of Conceptual Relations](image)
The value facet represents the actual value (current value) expressed by a relation. The procedural facets correspond to procedures to be invoked, depending on the type of operation performed on the relation. Restriction facets are elsewhere also called value-class (type), and element-of\(^3\) (enumeration), while the constraint facet symbolizes any other restriction one may want to define (e.g. ranges, complex expressions). Default represents a probable value for use if no actual value exists, while belief represents assumptions regarding the actual value. Dependency is a derivational trace of the value, and authorization specifies which other categories may refer to the value.

![Figure 3.10: A Subclass Hierarchy of Facet Types](image)

A standard set of facets to describe and constrain relations (adapted from [Jansson-86]).

Jansson adopts a declarative representational view\(^2\), but keeps the possibility for procedural extensions open: All static knowledge (time independent knowledge) is interpreted declaratively, independent of its use and role in problem solving and learning tasks. A general, logic-based interpreter is assumed. Some types of dynamic knowledge (state transitions, processes) may be represented procedurally, e.g. by procedural attachments. Here, the inference methods are not solely described by a general interpreter, but also represented as domain specific knowledge defined in pieces of program code.

Getting back to the framework requirements of section 3.1, a model of representational terms should be biased towards explicit representation of all relevant knowledge. Jansson’s model is biased towards modelling of knowledge as part of a more general information system analysis task, where large parts of the domain term set (domain symbols) will be syntax strings with no

---

\(^1\)In a way similar to the taxonomic relation with the same name, although its representational role is different
\(^2\)Declarative, in the logical sense (see section 3.4.1.1).
semantic content within the system. That is, their semantic content will be interpreted by a human user only. Compared to Jansson's representational term structure, the modelling approach of our framework encourages a thorough, explicit, representation of all symbols. Further, to facilitate reasoning within real world, open domains, a concept is viewed as a prototype, i.e. a structure with typical properties, where any property (unless otherwise stated) may or may not be an actual property of an instance. This has lead to a stronger role of intensional concepts than in Jansson's model, where extensional categories is the basic representational term. It has also resulted in a more flexible approach to semantic interpretation than represented by a declarative approach: Particular types of knowledge may need interpretation procedures that do not exist at the time they are needed. Hence, the framework should allow for definition of new and different semantic interpretation procedures as the need arises, and not be limited to a logical, model theoretic one. Such extensions of the representation must be carefully done, in order to preserve global coherence of the knowledge model. This may be achieved by representing the interpretation procedures as explicit concepts with certain preconditions, roles, constraints, input- and output types, etc1.

Representational explicitness is also facilitated in our framework by a unified interpretation of concepts and relations, which is achieved by viewing relations as a type of concept. Facets then become properties of a value pointed to by a relation, rather than a way of expressing relations. For example, in

\[
\text{car has-number-of-wheels (default 4)}
\]

the facet "default" is a characterization of the value 4. Important parts of the framework's model of representational primitives are illustrated in figure 3.11.

The top level term2 is called object. A representation object has a syntax structure and a semantic interpretation. The syntax structure is composed of numbers, strings and symbols. Numbers and strings represent themselves, while a symbol refers to a semantic structure, called a concept. A symbol that refers to a concept is called the concept's name. A concept may be defined extensionally or intensionally, but within the modelling approach represented by the framework, a generic concept is viewed as an intensional description, a scheme containing the expected properties and values of its instances. Concepts and semantic classes are synonyms,

---

1 This approach is taken in the CYC-project [Lenat-86].
2 'Top level' refers to the most general primitive terms. At first sight this may seem strange, since the top level terms are also the most basic terms, being used to compose the more complex structures further 'down' the hierarchy. If the view of the model as describing properties of representational terms is kept in mind, hopefully this will not lead to confusion.
Figure 3.11: Representational Primitives

A semantic model of knowledge representation terms. Some characteristic properties of the model are the distinction between object and concept, and the fact that both an entity and a relation is a concept. Essentially, the model represents a view to the meaning of the term concept. The objects that the model describes are representational primitives. Links to semantical primitives of the real world is illustrated by the broken arrows from primitive-entity. Most relations have inverses with different names, as exemplified by the causes and function-of relations.
while extensionally defined concepts are called categories\(^1\). A concept is identified by a name, and has a value that describes its properties. The value of a concept represents the meaning content expressed by the concept.

A concept’s value may range from a single attribute-entity pair (e.g. the concept color-of-my-house has attribute brown), to a complex structure of entities and relations forming a model of the concept (e.g. the concept drilling-fluid).

A concept is either an entity or a relation. A relation is a link between two or more concepts. An attribute-relation is regarded as a unary relation, describing an intrinsic property. An attribute relation has no explicit semantical content. For example,

\[
\text{my-car: red, big, old}
\]

expresses that my-car has the three attributes red, big and old\(^2\). This way of expressing concept properties (features\(^3\)) is common in classical, syntax oriented machine learning methods. Attribute expressions involve implicit relations. A relation like is, has-attribute, has-feature, or has-property may be considered as a hidden, un-named attribute-relation.

Knowledge-intensive problem solving and learning methods are able to make use of explicit relationships in their understanding and reasoning processes. Attributes may then be replaced by semantic relations whenever such relations are known:

\[
\text{my-car: has-color red, has-size big, has-age old.}
\]

Has-color, etc. are concepts with their own definition within the knowledge model, describing, e.g., what it means to have color. Being able to represent relations as concepts is a requirement for thorough knowledge modelling, and an important characteristic of our framework.

Entities are split into primitive and complex entities. Primitive entities correspond to semantical primitives of the domain. Some examples of such primitives, which will be referred to later, are indicated by dashed arrows indicating a penetration into the domain concept plane. A physical object is an object that occupies a part of physical space, characterized by a particular volume, shape, mass, etc. An abstract object is a non-physical object, like a mathematical point or line, a measurement, or a hypothesis. A state is a coherent characterization of a part of the world with particular properties, an event is a transition between states, and a process is a

---

\(^1\)This is consistent with the terminology of Smith and Medin [Smith-81], which gives a comprehensive treatment of ways to define concepts.

\(^2\)To be correct, the implicit feature names are colour, size and age, with the explicit feature values as shown.

\(^3\)A feature is synonymous to a concept property, including attribute type as well as semantic type properties.
sequence of events. Complex entities range from simple facts, or statements, to large conceptual structures (models).

Relations are also grouped into subclasses with characteristic properties. Properties of relations are important for generating and evaluating explanations (chapter 3.3). The most significant subtypes in this respect are shown in the figure.

Composite structures are built from the representational primitives. A concept’s value may be any meaningful complex structure that is expressible in the representation language being used. A fact is a structure that is defined recursively as:

Definition: 

\[ \text{fact} \]

\[ \text{entity} \]

\[ \text{OR} \]

\[ \text{fact-relation-fact} \]

triplet.

A relationship is a structure that relates entities or facts to each other via a relation, i.e. a relationship is a fact-relation-fact triplet. For example, the fact:

\[ \text{car has-part steering-wheel} \]

is a relationship described by the relation has-part between the entities car and steering-wheel, while

\[ (\text{car has-fault fuel-tank-empty}) \text{ causes} (\text{engine-does-not-turn}) \]

is a fact expressed by a causal relationship between a composite fact and an entity.

The specialization of structural relation as shown in the figure - having member-of, part-of, etc. as instances instead of subclasses - needs an explanation: One might view an instance of a relation as a concrete application of the relation in a relationship, for example the application of has-part in the example just mentioned. According to this view, the relation between structural relation and part-of in our model should be has-subclass instead of has-instance. The has-instance relation between them reflects that our model is a model of representational primitives, contained in the upper plane of figure 3.8, as previously stated. Hence, the 'bottom level' relational concepts are the atomic relations, like has-part. Instances of actual relationships are part of the lower plane in figure 3.8.

\[ ^1 \text{In the literature, the term proposition is also frequently used to denote what here is meant by the term fact.} \]
3.4.3 A Network View of Knowledge

Being able to utilize knowledge of different types intensively and effectively, requires the different types of knowledge to be represented in ways that ensure a common understanding. From a given piece of knowledge, new knowledge may be derived by some inference procedure. An inference procedure - guided and constrained by some operational goal - enables reasoning, i.e. deriving a conclusion to a problem or answer to a question. If more than one representation formalism is used, the interpreters and reasoning procedures must preserve the meaning content of a piece of knowledge across representation formalisms.

![Diagram of a tangled network model]

**Figure 3.12: Integrating Cases and Rules into the Conceptual Network**
A simplified visualization of how associational knowledge is integrated into a deeper knowledge model of entities and relations represented by the semantic network.

The area of knowledge transformation - aimed at moving knowledge between different formalisms, each with its own 'view' of knowledge - is an active research field in AI (a discussion of some of the difficulties involved is given in [Rothman-88]). Different representation formalisms have different strengths and limitations, and therefore impose different views or scopes of the knowledge. Automatic transformation of knowledge without changing the knowledge content is therefore extremely difficult, except for formalisms which are very close in type and form of expressiveness and underlying inference paradigms. It takes
a lot of knowledge to transform knowledge while preserving its meaning. In order to ensure a common understanding of basic, definitional knowledge among various reasoning methods, while avoiding the complications of knowledge transformation, knowledge should be represented within a single, unified representational system that is expressive enough to capture all relevant knowledge. This unified model is based on viewing a knowledge model as a tightly coupled network of concepts.

The term knowledge model, according to this view, refers to the total network of concepts and relations of different types - including heuristic rules - as well as the collection of past cases. The nodes in the network are concepts (entities as well as relation type concepts), and the links are relations. A relation concept defines the property of a link in the network at the conceptual level. Within a total knowledge model, surface level, associational knowledge is connected to the deep, conceptual model by having their concepts defined within the model. Associational knowledge is typically experiential knowledge in the form of heuristic rules or concrete, experienced cases. Figure 3.12 illustrates how cases and rules may be viewed as integrated into the conceptual network model by having all their descriptors explicitly defined in the model.

A knowledge model should be viewed as a single, tightly coupled network of concept definitions, heuristic rules, past cases, procedures, methods, and strategies. The basis for the system's understanding is the fundament of descriptive knowledge formed by explicit definitions of all terms used in the various submodels.

3.5. Problem Solving and Learning - A Knowledge Intensive View

To take advantage of a thorough knowledge model, powerful methods for utilizing the rich body of represented knowledge are required. The framework needs a general model of problem solving and learning that fits its "extensive, explicit and tightly coupled" view of knowledge representation.

This subchapter presents a broad, general model of problem solving, reasoning and learning where the generation and evaluation of explanations plays a core part. The approach described partly draws upon results from theories and research on human cognitive processes. This is founded on the belief that a computational model for robust problem solving in real world complex domains should to some extent take advantage of existing theories of human problem solving. After all, people have shown to be the most successful problem solvers to date. This is not taking the stance that a goal of AI is to emulate human reasoning per se, the rationale is a pragmatic one: In order to build computer-based problem solvers at an expert level, look for justified and plausible models of human problem solving to start out from.
3.5.1. The Concept of Explanation

The concept of an explanation has two interpretations in AI. One is as a means of increasing a system user’s understanding of what is or has been going on inside a system. This is the interpretation commonly used in connection with expert systems, and has been the focus of work done by Clancey [Clancey-84] and Swartout [Swartout-87], for example. The objective of an explanation is to make parts of a system's reasoning process transparent to the user. Typically, the system explains why a certain question was asked or how a certain result was reached. An explanation may be a trace of a rule-chain leading from some observations to a conclusion (like in EMYCIN [VanMelle-80]), or a more complex justification based on a deeper model of the knowledge [Steels-85] and/or an explicit model of the problem solving strategies [Clancey-86].

The other interpretation of an explanation in AI is as a method for inferencing and reasoning. Classifying an example as an instance of a concept, selecting relevant symptoms from a set of observations, choosing the best diagnostic hypothesis on the basis of relevant findings, assessing the plausibility of a derived result, etc., may be viewed as processes of producing the necessary explanations to support or justify a hypothesis. When receiving a patient with the problem “I have this pain in the lower right region of my abdomen, what is the diagnosis?”, a physician will typically use his knowledge to generate a set of possible hypotheses, and explain to himself whether it is likely to be, e.g., an appendicitis, a strained muscle, a sub-dermal inflammation, or whatever fault he regards as reasonable given the observations and the current situation context. Generating and evaluating explanations then becomes an essential part of the reasoning process itself. This role of explanations gets to the heart of what AI is about, as argued by Kodratoff in an article entitled: “Is AI a Sub-field of Computer Science - or is AI the Science of Explanations?” [Kodratoff-87a]. Schank (in [Schank-86a]) argues that the capability of an agent to explain to itself why something is the case, or how a result was derived, is a requirement for understanding, and hence for intelligent behaviour. He further argues that the most important driving force behind learning is an expectation failure, where the the learning results from the attempt to explain why an expected result was wrong.

This (second) perspective on explanation is of particular relevance to the work presented in this dissertation, as it has been to a lot of other problem solving and machine learning approaches - of which an overview is given in [DeJong-88]. This is not saying that user explanations are not important. It rather expresses the view that developing a model of internal explanations in an AI system is the right place to start: In order for an intelligent agent to give a good explanation of something to a person, it is of great help being able to explain the same thing to itself.
3.5.2. Associational and Elaborative Reasoning

The following scenario will be used as a basis for the further discussion:

It is a cold morning. A driver opens the door of his car, gets inside, puts the ignition-key in its slot and turns it to the starting position. He hears the engine turning, but it will not start. He releases the key and the engine stops. Repeating these actions does not help; it only makes the engine turn more slowly each time. Why will the engine not start? What is the fault?

Finding an answer to a question like this may be done in two apparently different ways: An agent attempting to solve the problem may know the answer, or it may infer it\(^1\). "Knowing", in this context, refers to a kind of look-up process where the observations just made are matched against a set of similar observations stored in a knowledge structure, and where this set of observations point directly to a solution to the problem. This knowledge may be a particular past experience that is being reminded of (e.g. the same problem was solved two days ago), or a general rule induced from a set of experiences or acquired from an external source. The kind of reasoning involved in this process is sometimes referred to as **associational reasoning**\(^2\) (e.g [Koton-89]), since knowledge is used to directly associate problem descriptors with solutions. The knowledge involved is typically operational knowledge at a shallow level (see fig. 3.1), capturing well established or experienced relationships among knowledge units.

The other type of reasoning infers a conclusion by going through a series of elaborate reasoning steps. In order to arrive at a conclusion, the problem solver generally has to traverse different types of relationships with various types of interdependencies, constraints, etc. The aim of this process is to produce a reasoning chain that is a good explanation of the observations (and other problem descriptors). This type of reasoning is here referred to as **elaborative reasoning**. The knowledge used is typically descriptive, deeper knowledge that is part of a model for describing the domain\(^3\).

The distinction between an association and a more elaborative interpretation is not sharp, however. Even if an answer to a problem is 'known', some pre- or post-processing is always

---

\(^1\)A 'third way' would be to get the answer - or part of it - from an external source. This new knowledge would have to be used in one of the two ways listed, so whether the knowledge already exists in the agent, or is acquired before (or during) problem solving, is irrelevant in this respect.

\(^2\)This kind of knowledge is also referred to as "compiled knowledge" and "chunked knowledge".

\(^3\)The typing of knowledge into associational and inferential is concerned with the use of knowledge, rather than characteristic properties of the knowledge as such. For example, knowledge which is part of a conceptual model may be used for associational reasoning as well - for example a direct association along an implies relation.
involved to ensure that the associated solution holds. Taking the car-starting problem, even if there is a reminding to a previous similar starting problem, some crucial questions have to be answered: Are external factors (such as temperature, humidity, geographical site) the same as for the previous case? Is it the same car? In other words: Is the problem context the same? Two problem situations may be very similar, but they are never exactly the same (at least the point of time is different). No matter how directly a solution to a problem is retrieved, it will always have to be checked against some aspects of the current problem context before it can be applied (or rejected): Is there an explanation that asserts the validity of the past solution in the present situation? Or, is there an explanation for why the previous case does not match closely enough? Producing and evaluating explanations like these are important parts of expert level problem solving. The processes involved are highly knowledge-intensive processes, that need to have a system’s entire knowledge resources at its disposal. Deeper model knowledge obviously has an important function in the explanation process, whether it supports associational or drives elaborative reasoning.

Research results from philosophy and psychology strongly indicate that expert level human problem solving to a large degree is associational [Anderson-85, Schank-82, Johnson-Laird-83]. Highly competent people typically solve problems by being reminded of a similar or analogous situation, or a principle or rule that applies in the current situation. They arrive at a conclusion by modifying the solution to a previously experienced problem, or by instantiating a principle or rule. Students and other novices tend to use basic, theoretical knowledge much more frequently than experts. Experts, too, sometimes have to fall back on more general, basic knowledge, but they are able to focus the use of this knowledge according to their experience about ‘what may work’. They are therefore able to arrive at a conclusion more quickly than inexperienced students. Experts are also able to combine the two types of knowledge by using the deeper model knowledge to select the right piece of associational knowledge to apply. This often involves creating an explanation for why that piece of knowledge is appropriate - and why other chunks of knowledge are not - as well as explaining the plausibility of the derived result as a solution to the given problem.

Influenential theories on learning at a competent level - the making of a skilled novice into an expert - view the two knowledge types as a continuum between two extremes: On the one side fundamental, first-principle, declarative, deep knowledge, and on the other side procedurialized, compiled, automated, chunked knowledge (e.g. [Laird-86, Anderson-85, Dreyfuss-86]). Learning through experience is seen as a transition between the two states.
3.5.3. What Constitutes a Good Explanation?

This is a fundamental philosophical question. A lot of research has been - and is being - done in order to get closer to an answer, particularly within the philosophy of science. Peter Achinstein [Achinstein-83] and Paul Thagaard [Thagaard-88] both give an overview of different theories and models of explanation within philosophy, and the role explanations of scientific hypotheses play for the development and application of scientific knowledge.

In order to analyze explanations, an explanation is often regarded as consisting of two parts: The part that is to be explained, called the *explanandum*, and the actual explanation, called the *explanans*. For example:

```
Explanandum: The patient died
Explanans: The patient had cancer
Explanation: The reason the patient died is that he had cancer
```

Achinstein emphasizes the role of the explanation process - the *explaining act* in which someone writes or utters something to someone else. What is written or uttered in this process is called the *explanation product*. Achinstein’s view is that an explanation (product) can not be understood or evaluated without reference to the explaining act. The explaining act defines some aspect of the context and purpose behind the explanation which is needed for a correct and meaningful interpretation of the explanation product.

Thagaard, as Achinstein, is concerned about what he calls the *pragmatics* of an explanation. Explanation is viewed as a process of providing understanding, and understanding is to a large extent achieved through *locating and matching*. This is a view of reasoning based on matching, retrieval and adaption of knowledge structures like schemas or scripts - and more general ‘mops’ (memory organization packages, [Schank-82]). In Thagaard’s model, these knowledge structures - based on concrete or generalized situations or episodes - are supplemented with more general knowledge in the form of rules. In order for an explanation to be understood, it must activate this ‘mental model’ in a meaningful way - that is in a way that enables the existing knowledge structure to confirm the explanation without seriously contradicting with other parts of the knowledge structure.\(^1\)

---

\(^1\)Contradictions are not necessarily a problem, given that the contradiction can be resolved by another explanation. Such an explanation may lead to a weakening of the importance or relevance of one of the contradictands, for example by taking into account contextual constraints that were not regarded as important in the first place.
Both these views are in contrast with C. Hempel’s classical deductive-nomological (“deduction from laws”) theory of explanation [Hempel-65], where an explanation is a deductive proof from a set of laws (a theory, axiomatized in a logical formal system) and a set of premises. There is no mentioning of contextual, intentional or other influencing factors from the explanation act in Hempel’s model. A good explanation is - according to Hempel - a logically valid argument from a set of premisses. So, assuming the premisses is true, a good explanation of the premisses is their proof. This ensures that an explanation will always be correct. The question is whether correctness is a sufficient criteria for a good explanation. Both Achinstein and Thagaard give examples of some of the problems with this approach. For example:

The problem is explaining why a certain flagpole is 15 metres high.
The length of the flagpole’s shadow, the position of the sun and some trigonometric laws are given (among other facts about flagpoles and other objects).

A calculation of the flagpole’s height on the basis of the given facts gives 15 meters. This is a true (valid) deduction on the basis of laws and premisses, but is hardly any (good) explanation of why the flagpole is 15 metres high. A good explanation would need to focus on the purpose and use of flagpoles, knowledge about manufactured objects, limitations of manufacturing techniques, typical height of neighbouring objects to the flagpole (houses, trees), etc. Another example:

The reason the patient died is that he had bacterial infection.

Assuming this is true, is it a good explanation? If it is, it should increase the understanding of the patient’s disease. In this case it is unlikely, since - although it is correct - the explanation is too general. Thus, some evaluation criteria for a good explanation in addition to correctness is needed.

There seems to be consensus among various schools that correctness in not a sufficient criteria for an explanation (see the discussion following McDermott’s critique of logisism [McDermott-87]). What about necessity? Is correctness - or truth in a deductive sense - a necessary criteria for a good explanation? Hempel and most other researchers taking a logisist view say ‘yes’. One of the counter-arguments to this view is related to the notion of truth: McDermott [McDermott-87] argues that an explanation may be good merely by making the observed facts probable, not necessarily proving their truth. Another argument is related to the role of pragmatics: An explanation may be a good one (in a certain context) even if it hides facts that under other circumstances would be regarded as important for understanding a phenomenon. An example is deliberately hiding or twisting the truth for pedagogical purposes. In the history of science there are lots of examples of explanations that served perfectly as explanations of phenomena and contributed to the advancement of science, although being false: The earth as the centre of the solar system, an ‘ether’ substance for propagation of electromagnetic waves,
Achinstein (p. 107) cites Ptolemy’s explanation of the motion of planets as an incorrect, but good explanation. He continues (p. 108):

“The goodness or worth of an explanation is multidimensional; correctness is only one dimension in an evaluation. An explanation is evaluated by considering whether, or to what extent, certain ends are served. The ends may be quite varied. They may concern what are regarded as universal ideals to be achieved, particularly in science, e.g. truth, simplicity, unification, precision. Other ends are more "pragmatic".”

Good explanations, and useful results, may come out of inherently false premises, as long as the premises have some aspects or properties that are true and meaningful within the actual context. A false premise may have a generalization or a part that is true and relevant, or it may point to an analogy that is useful for producing an explanation.

Since the deductive-nomological theory of explanation does not analyze the actual meaning within the context of the explanandum, often leading to an explanans that by no means serves as a good explanation, it is regarded as being based on a rather syntactical view of knowledge. Its underlying model theoretic (Tarskian) semantics is well defined in terms of the rules of logical deduction, but too weak to relate semantics to more pragmatical issues, like the context, perspective¹, and intended use of the semantic interpretation. Thagaard expresses it like this ([Thagaard-88], p. 35):

“Of course, formalization of some sort will be necessary for any computational implementation of scientific knowledge, but it will have to be directed toward procedural issues rather than logical rigour. The emphasis on syntax that is endemic to the view of theories as axiom systems leads to the neglect of semantic considerations crucial to the understanding of conceptual development, and to the neglect of pragmatic considerations that are crucial to justification and explanation.”

3.5.4. A Computational Model of Explanation

What is needed in the framework discussed here, is a computational model of explanation. This model should be based on relevant general theories of what explanations are, and how they function in problem solving and learning. Essential parts of such a theoretical basis were described in the preceding section. The model of explanation within our framework is based upon the following fundamental principles:

• Correctness - in the broad sense that includes plausibility - is regarded as a necessary requirement for a good and useful explanation.

• A plausible result is viewed as a product of an abductive reasoning chain, i.e. as a product of an inference to the best explanation. This view of abductive reasoning and explanation

¹For example, the meaning of a car is different when viewed as a means of transportation than as a mechanical device built out of certain components. Two different perspectives may even lead to contradictory statements, like viewing light radiation as wave propagation or as particle movement, although they are both considered true in the real world.
is close to the views expressed by several philosophers with a computational view of epistemological problems (as briefly reviewed in [Thagaard-89]).

- Multiple explanations produced from a body of knowledge need to be coherent. The notion of coherence in our model has the same role as consistency in a model which assumes complete domain theories. Coherence is a weaker term than consistence, and expresses a context dependent, global evaluation of locally consistent bodies of knowledge.

3.5.4.1. Explanations and Coherence of Knowledge

The notion of knowledge coherence - as a relaxation of the formal notion of consistency - has been adopted by several authors (e.g. [Davis-83a, Clancey-85, Schank-86c, Lenat-87]), but very few attempts have been made to develop models or theories of knowledge coherence relevant to our framework. Thagaard, however, (in [Thagaard-89]) proposes an interesting theory of explanatory coherence. Coherence, in this theory, is primarily related to a set of propositions. It only makes sense to talk about coherence of a single proposition if viewed with respect to another set of propositions. The notion of acceptability is introduced to characterize this property of single propositions. The theory is based on seven principles for evaluation of acceptability and coherence, summarized below:

1. **Symmetry**
   - If P and Q cohere, then Q and P cohere.
2. **Explanation** (in general)
   - If hypothesis P is part of the explanation of evidence Q, then P and Q cohere.
   - If hypothesis P₁ is explained by hypothesis P₂ then P₁ and P₂ cohere.
   - The degree of coherence of a proposition in an explanation is inversely proportional to the number of propositions constituting the explanation.
3. **Analogy**
   - If P₁ explains Q₁, P₂ explains Q₂, P₁ is analogous to P₂, and Q₁ is analogous to Q₂, then P₁ and P₂ cohere, and Q₁ and Q₂ cohere.
4. **Data priority**
   - A proposition that describes the result of an observation has a degree of acceptability on its own.
5. **Acceptability**
   - The acceptability of a proposition P in a system S depends on its coherence with the propositions in S.
   - The acceptability of a proposition that explains some evidence is reduced if a large portion of the relevant evidence is unexplained by any hypothesis.
6. **System coherence**
   - The global explanatory coherence of a system is a function of the pairwise local coherence of those propositions.

These principles represent a refinement of the model of abductive reasoning described in [Thagard-88], where consilience (favouring explanatory breadth), simplicity (favouring explanations with few propositions), and analogy (favouring explanations based on analogies) are claimed to be the most important criteria for selecting the best explanation.
These principles are suitable as a basis for a model of explanation in our framework. However, it needs an extension: Thagard's model provides no means to describe the varying importance and relevance of different relations which constitutes an explanation chain. A notion of explanation strength is therefore included.

The strength of an explanation as an argument for justifying a result, is heavily dependent on the relations involved in the explanation. Getting back to the car-starting problem, the explanation:

```
The engine does not start because
  the battery voltage has voltage very low,
  a very low battery voltage causes the starter-motor not to turn
  starter-motor has function of turning engine at ignition speed
  engine at ignition speed is a necessary requirement for engine to start
```

is an explanation chain involving several relations (has-voltage, causes, has-function, necessary-requirement-for). The explanation supports an abduction from engine-does-not-start to very-weak-battery. Two important factors for determining the strength of this explanation are the relevance - within the current context - of the entities referred to, and the explanation strengths of the relations. A causes relation, for example, generally has stronger explanation strength than occurs-together-with; always-causes is stronger than may-cause, etc. A system using explanations to support derivational steps needs to have a model of evaluation and combination of explanations. A specification of how explanation strength is modelled and evaluated will have to be system dependent, and not specified by the framework. As a basis for such a model, a generic structure of explanations is outlined in the following section.

### 3.5.4.2. Explanation Structures

Within our framework, an explanation-relation is referred to as a type of relation with some explanatory strength. Within the representational framework described here, an explanation has the following syntactic structure:

An explanation is a chain of facts and explanation units

- An explanation unit is a fact - explanation-relation - fact triplet

The explanation shown in the preceding section would be expressed by the following explanation chain, containing both facts and explanation units:

```
battery has-voltage very-low  (fact)
(battery has-voltage very-low) causes (not turning-of-starter-motor)  (e.u)
starter-motor has-function (turning-of-engine (at-speed ignition-speed))  (e.u)
(turning-of-engine (at-speed ignition-speed)) necessary-requirement-for engine-start  (e.u)
```
A simpler explanation - containing only explanation relations and entities:

car subclass-of vehicle, vehicle has-function transportation-of-people, transportation-of-people enables moving-within-a-population

is a partial explanation of why the car leads to more frequent moving within a population.

In the following three sections a generic model of problem solving, reasoning and learning is presented, in which generation and evaluation of explanations play an important role. The model is described as submodels - in the form of iterating processes - according to the general scheme:

1. Initialize internal knowledge structures
2. Derive intermediate results, focused by the goal of the process and supported by explanations within the knowledge model
3. Evaluate competing candidates for the final result, and decide on final action

3.5.5. Phases in Problem Solving

Much research in cognitive psychology and artificial intelligence have addressed the issue of breaking the problem solving process into smaller phases. A basic principle in AI is to view problem solving as search in a problem space. The problem space consists of an ultimate goal state, an initial state, and intermediate states representing subgoals and states derived from the initial state. Problem solving, then, involves finding a sequence of operators that transforms the initial state into the goal state, thereby achieving the goal. Many problem solving models have been developed to describe how goals may be split into subgoals, how operators are formed or selected, etc. (e.g. the Soar model [Laird-86]). A phase model of problem solving - as described here - is a more abstract model of the problem solving process at a strategic level, specifying when to do what, rather than how to do it.

Some psychological models of problem solving phases are reviewed in [Mayer-83], (pp 42-52); G. Wallas, in his 1923 book called The art of thought, suggested four problem solving phases:

1. Preparation - gathering of information and proposing preliminary solutions
2. Incubation - putting the problem aside for a while

1 A large set of search methods have been developed in order to find the optimal path in the search space, from simple tree search methods (depth-first, breadth-first), via heuristic search methods like best-first search or hill-climbing, and up to more complex search regimes involving control strategies, such as means-end analysis [Newell-72].
3. Illumination - finding the key to the solution
4. Verification - checking the solution to make sure it works

This model was based on introspection reports from several scientists within science and mathematics. It is flavoured by solving scientific research problems, where sudden moments of insight ("aha") play a significant role in arriving at a solution (step 2 and 3). Another model is given by G. Polya in How to solve it (1957), based on his experience as a teacher of mathematics:

1. Understanding the problem - gathering information about the problem
2. Devising a plan - using past experience to find a method of solution
3. Carrying out the plan - trying out the plan, checking each step
4. Looking back - checking the result, trying another method, checking if result or solution method applies to other problems

As pointed out in [Mayer-83] the two schemes have strong similarities. Polya’s model is particularly interesting, since it emphasizes solving new problems by reminders to previous cases (step 2), as well as a final step which may be extended towards a learning method that retains knowledge useful for future problem solving. As steps 2 and 3 indicate, the model describes problem solving as a search for a problem solving method rather than a particular solution.

Computational models of problem solving phases are, in general, described at a more detailed level than the psychological models, specifying the reasoning and inference methods rather than a top down problem solving scheme. Some of these models will be discussed in the next section. Clancey, for example, (in [Clancey-85]) presents heuristic classification as a problem solving model, but since it describes a high level inference structure, rather than a top level phase model, it is treated here as a reasoning model.

Below, our framework’s three-phase computational model of problem solving within a knowledge rich environment is outlined. The phases describe the problem solving process in general, and irrespective of whether a problem is solved by direct association, by a more elaborative inferencing process, or both. Compared to Polya’s model, the perspective of this phase model is to find a particular solution, rather than a solution method or plan. While the search for methods in a multi-paradigm reasoning framework is important as well, the primary goal of the process - the solution - is regarded as the natural focus of a top level model. Another distinction with Polya’s model is that the verification and looking back phase is not explicitly
included, since it is regarded as a phase of another type - and at another level - than the other three phases: It is a kind of re-running of the first three phases on another task (verification).

The problem solving model of the framework has the following three phases:

A. Understanding the problem
B. Generating plausible solutions
C. Selecting a good solution

Although the process starts with A and ends with C, problem solving is generally not a strictly sequential process. It is rather a cyclic activity of A-B-C-A, A-B-A, and B-C-B cycles, which may be recursively called on subproblems.

A. Understanding the problem

The first step is an attempt to understand the problem. An agent - human or artifact - competent of solving problems in a domain, is viewed as having a cognitive model within which understanding, learning and problem solving takes place (different cognitive models are described in [Schank-82, Johnson-Laird-83, Minsky-88]). The cognitive model holds the structures and methods of an agent’s expertise (fig. 3.2 and 3.4). It contains domain knowledge integrated into more general knowledge of the world, and corresponds to what has been referred to as the conceptual model - or general knowledge model - of a knowledge based system. In order to understand a problem, the input description has to be interpreted by this model. The interpretation process may be viewed as a series of coherent explanation processes that select relevant descriptors from noisy ones, identify inter-relationships among them, and test whether the given and inferred facts make sense.

Understanding then becomes a process of elaborating upon the problem description using the cognitive model: When an agent is presented with a new piece of information, an attempt is made to integrate the new fact into the existing knowledge. This may be viewed as explaining the new fact within the existing cognitive model, by generating confirmations or plausible justifications to support - or reject - the new information. The integration process checks for immediate contradictions and infers implications (associated facts) recursively. This may be done by following particular explanation-relations in the deep model (e.g. causal or functional relations), deriving implications through rule chaining, or generating expectations by retrieving similar cases. If the agent is unable to understand a given fact, there are three optional ways to proceed: The agent may try to solve the conflict by itself. This would involve a further elaboration by looking for explanations in more ‘distant’ parts of the knowledge. The second
option is for the agent to interact with the external world - for example by asking for more information, for help to resolve contradictions, or for confirmation of weak explanations. The third option is to accept the lack of understanding at this moment. This implies that a fact given or a contradiction derived is accepted, and saved, as an unexplained piece of information. Resolving the conflict is then postponed until later - when further processing of the problem may have produced additional knowledge and information.

Knowledge-based systems that do not allow representation of conceptual structures for the purpose of defining concept meaning in a deep sense, will have to reduce the importance of the Understanding phase. In a purely rule-based system with only shallow knowledge, an understanding as described here will hardly be possible. Given the requirements of section 3.1, a conceptual knowledge model that enables some degree of understanding is presumed in the framework.

B. Generating plausible solutions

The next major phase is to generate a set of plausible solutions, i.e. a set of possible solutions that has been justified in the sense that they achieve the goal without contradicting important constraints. A plausible solution may also have a trace attached to it, that explains (supports) the solution.

Having understood the problem implies that some knowledge structures have been activated. To understand the car-starting problem, for example, concepts like car-starting, starter, starting-fault, battery, etc. will have been activated, along with a lot of associated entities, relations, possible faults, repair actions, alternative ways to start a car, etc. Given the observations, the goal of getting the car started, and knowledge about car-starting problems, a competent problem solver will arrive at some candidate solutions to the problem. This may be done by adapting a solution from a previous case, by rule chaining, or by inferencing in a deeper model of knowledge. Further elaboration may take place immediately to strengthen or weaken some of the candidates, for example by creating multiple explanations for supporting an hypothesis or looking for contraindications. The eventual result is a set of hypotheses that are believed strongly enough to be subjects for further evaluation.

C. Selecting a good solution

The last phase is an evaluation of plausible solutions in order to select a good solution to the problem. This phase is characterized by the fact that a single solution candidate has to be chosen (only one solution can be the first one to try out), and a strategy and criteria for selection must be a part of the domain model. The user may play an active role in this phase.
If the selected solution is found unacceptable, a re-iteration of the problem solving process takes place (in principle), where the problem description may be stripped by removing irrelevant descriptors, and expanded by parts of the current system state, like derived consequences of the initial description, unacceptable solution types, and reasonable assumptions.

The figure below summarizes the roles of the three phases:

![Figure 3.13: Main Phases of Problem Solving](image)

The figure illustrates the UNDERSTAND - GENERATE - SELECT process of knowledge-intensive problem solving, transforming a problem description into a solution.

3.5.6. An Explanation Based Model of Reasoning

A reasoning model is a further specification of some parts of the problem solving process. Within the perspective of the problem solving model just described, a reasoning process is a successive combination of various inference methods (such as matching, property inheritance, constraint propagation, rule deduction), guided and supported by explanation methods that focus on the current goal and context of the process. The reasoning process is, in general, based on a combination of the three reasoning types: model-based reasoning, case-based reasoning and rule-based reasoning (see figure 3.4, and the accompanying text).

In the previous section problem solving was described as a three-phase process. In this section a generic model of reasoning is described. As described in section 3.2, reasoning is a subprocess of problem solving. The reasoning model to be described is generic, in the sense that it may be applied to each of the three problem solving phases.
As indicated in the previous chapter, Clancey’s *heuristic classification* model is a well known example of what is here called a reasoning model. It is reproduced from [Clancey-85] in figure 3.14, showing the three subprocesses Data Abstraction, Heuristic Match, and Refinement. The basic inference step is the heuristic match, characterized by an experienced relationship or a poorly understood correlation.

![Figure 3.14: Clancey’s Heuristic Classification Model](image)

The heuristic classification model is a reasoning method where the main inference step is a direct association between data abstractions and solution abstractions, i.e. an association between different types of concept structures. The association is *heuristic* - often empirical, corresponding to ‘rules of thumb’.

A heuristic relation is always uncertain, based on typicality. Clancey expresses it like this:

”The heuristic classification model characterizes a form of knowledge and reasoning - patterns of familiar problem situations and solutions, heuristically related. In capturing problem situations that tend to occur and solutions that tend to work, this knowledge is essentially experiential, with an overall form that is problem-area-independent.”

Clancey’s model is inadequate as a basic reasoning model within our framework, for the following two reasons:

First, the heuristic classification model is motivated by analyzing and describing rule-based systems (the model is based on an abstraction of MYCIN [Buchanan-84]). A heuristic rule is a direct association between input data generalizations and a generalized conclusion, e.g.:

\[
(\text{ignition-key turned}) \land (\text{lamps (all dark)}) \implies \text{electrical-system-fault}
\]

Clancey refers to heuristic classification both as a framework for problem solving and as an inference structure. As pointed out in section 3.2, this interpretation of the term inference structure makes it synonymous with our notion of a reasoning model.
Case-based reasoning also involves an associational match between input and conclusion, but at the instance level rather than at an abstract level:

```
car-starting-case-351
time early-morning
ignition-key turned
lamps (all dark)
radio (not working)
battery-voltage ok
fault starter-solenoid-defect
```

Second, even if a heuristic relation expresses an association that has shown to be useful, it may be explainable by a deeper knowledge model, if such a model exists in the system. The kind of knowledge rich systems aimed at in this dissertation will always have a deeper conceptual model of knowledge (requirement R1 in section 3.1), which may be used for justification of heuristic relations as well as deep model reasoning. The interaction between general heuristics and deeper knowledge must therefore be expressible within our reasoning model. Clancey's model does not capture the kind of reasoning and inference methods required to utilize a thorough domain model in this way.

What is needed, then, is a more general reasoning model, where Clancey's model is viewed as a submodel for one type of reasoning. Chandrasekaran's problem solving framework based on generic tasks [Chandrasekaran-87] provides a more general and flexible model of reasoning. A generic task is a unit of knowledge, with appropriate inference methods and control strategies for performing the task attached to it. Suggested generic tasks are¹:

- hierarchical classification - top-down refinement of hypotheses in a class/subclass structure
- structured matching - matching of data (problem descriptors) to hypotheses in a class/subclass hierarchy
- abductive assembly - finding a coherent collection of hypotheses that best explain the data
- database inference - deriving abstractions and other findings from input data

Using these building blocks, a problem solving process proceeds top-down in the hypotheses hierarchy, using the structured matcher to attempt matching hypotheses with data describing the

---

¹These are the generic tasks listed in [Chandrasekaran-87]. Other papers by Chandrasekaran or people from his group may present other 'task lists'. For instance, in [Josephson-87] the following list is presented as examples of generic tasks: hierarchical classification, plan selection and refinement, concept matching, knowledge-directed indirect inference, prediction by abstracting state changes, assembly and criticism of composite explanatory hypotheses. Hence, it is the idea of generic, problem solving specific tasks - and the level on which they are described - that is important, not the actual task list.
problem. The data are either given as input or inferred from the 'database'. Eventually, a set of plausible hypotheses is returned together with the data set that they explain. The abductive assembler then constructs a coherent explanation out of these partial justifications, which gives the conclusion of the process.

The notion of a generic task represents an interesting view of reasoning and problem solving. It differs from some other modular frameworks (e.g. KADS [Breuker-89]) in that domain knowledge is partitioned into task specific units, coupled to its use and associated with corresponding task specific control strategies and inference methods. The framework described in this dissertation is based on the assumption that task-specific knowledge is useful and relevant for modelling specific parts of a knowledge model, but also that large portions of knowledge and inferencing is of a general nature. Concrete experience from past cases, for example, is a typical example of task-specific knowledge. However, in order to achieve the high degree of understanding that is required by competent and robust knowledge based systems, a lot of knowledge will have to be of a general nature. The role of the more general knowledge is to (at the object level) generate supporting explanations from the general knowledge model when this is needed, to (at the control level) be the 'glue' necessary for the system to understand the inter-relationships between more specific, task-related knowledge (e.g. by explaining why a certain strategy is better than another in a certain situation), and finally, to be a knowledge fundament to fall back on when more specific expertise does not apply.

J. McDermott [McDermott-88] argues for a kind of generic task called role limiting methods, where the reasoning methods are linked to types of problems, but where control strategies resides outside the task-specific methods. These tasks are at a higher level than the ones suggested by Chandrasekaran. The following different role limiting methods - with the tools implementing them - are identified:

- cover-and-differentiate - MOLE [Eshelman-87]
- propose-and-revise - SALT [Marcus-87]
- acquire-and-present - KNACK [Klinker-87]
- extrapolate-from-a-similar-case - SIZZLE [Offut-88]

The extrapolate-from-a-similar-case represents a case-based reasoning method, but is too closely bound to the particular application (computer sizing) to provide a general case-based reasoning framework.

---

1 The database corresponds to a conceptual model, not necessarily deep or complex.
2 A motivation behind identifying the role-limiting methods at this level is to enable more powerful assistance in knowledge elicitation and modelling by building more specialized knowledge acquisition tools.
The model of reasoning within our framework is at a higher, more general level than the models just reviewed. Its purpose is to serve as a top level framework for describing different approaches to knowledge-intensive reasoning by combining general and case-specific knowledge. The model emphasizes the generating and evaluating explanations to support both abstract and concrete associations, as well as performing elaborative deep model reasoning. Given some findings and a goal, a reasoning process may, at a general level, be described by three sub-processes:

1. **Activating knowledge structures**
2. **Explaining candidate facts**
3. **Focusing on a conclusion**

---

**Figure 3.15: The Explanation-based Reasoning Model**

The framework’s generic model of reasoning, subsuming deep model reasoning as well as model-supported case-based and rule-based reasoning. The 'horse shoe' layout of the illustration is deliberately made to resemble Clancey’s model (figure 3.14). However, the interpretation of the arrows is different: While the arrows in Clancey’s model should be interpreted directly as abstraction to a higher - more general - level followed by an association at the same level, and finally a refinement to a lower - more specialized - level, the arrows in the figure above should be interpreted as more general processes. By substituting the processes in the figure with the more specific processes abstraction, heuristic match, and refinement - and renaming the two upper boxes, it is seen that Clancey’s model may be regarded as a specialization/simplification of this reasoning model.

A reasoning process is viewed as a process of *activating* a certain part of the existing knowledge - including triggering of hypotheses and goals\(^1\), *explaining* to what extent the activated parts form a coherent knowledge-structure, and *focusing* within the explained structure, returning an explicit answer or just the final state of the system, as illustrated in figure 3.15.

---

\(^1\)Goals are end-states to be achieved, while hypotheses are assertions to be confirmed or rejected. The notions of goal and hypothesis are to be interpreted in a general sense. They cover intermediate goals and hypotheses set up by the system during reasoning, as well as top level goals (like ‘find-fault’). A hypothesis may be an expected value of a problem descriptor as well as a possible solution to the problem.
The three phases of the model do not exactly match the phases of Clancey’s model: The middle phase - EXPLAIN - is missing as an explicit phase in the heuristic classification model. However, some means of justifying conclusions exist within that model as well, for example by checking for contradictions, or generating multiple heuristics leading to the same conclusion. In a type of heuristic classification called causal-process classification, the heuristics are replaced by causal relations (as in the CASNET [Weiss-78] and CADUCEUS [Pople-82] systems). Here, a problem description (a set of symptoms) is said to be explained by the conclusion (a diagnosis), but the causal association is interpreted as a direct 'heuristic', not as part of a thorough conceptual model involving many types of relations. In our model, this kind of propagation of activation along particular relations is regarded as part of the ACTIVATE phase. EXPLAIN represents a separate step that selects the activated concepts that are relevant for reaching a conclusion. This is done by generating support or justification from a deeper conceptual model according to some explanation generation strategy. The FOCUS phase is a kind of refinement towards a final conclusion.

Characteristic roles of each phase:

1. Activating knowledge structures

ACTIVATE initially sets up the knowledge structure by marking network concepts that match terms of the input description (data describing the problem, and goals) as active. Other concepts are activated by mechanisms that trigger activation of associated concepts. This spreading of activation has to be controlled in some way, either implicitly by limiting the types and numbers of associative links, or by explicitly controlling the selection of links to spread along dependent on the problem solving state of the system. For example, if a diagnosis system is trying to infer what new measurements to take, it may choose another subset of triggering associations than if it is trying to generate possible diagnostic hypotheses. Explicitly defined concepts as well as relationships (facts) may be activated. Examples of activation spreading are similarity matching of cases, rule chaining, propagation of constraints and deriving of consequences in semantic networks. The purpose of this step is to activate possible candidate concepts. Even if some control knowledge is available to constrain and guide the process, the activation phase is not necessarily a very knowledge-intensive process.

The phase names are chosen to highlight the major role of each phase. At a more detailed level, however, activation of concepts, rejection/justification of activated concepts, and focusing by reducing a set of candidate concepts, are general mechanisms that may be subprocesses of any phase - at varying levels of detail.

In addition, some problem solvers accept an input type referred to as support knowledge, i.e. input intended as support for the problem solving, reasoning and/or learning process. Support knowledge is control level knowledge such as what problem solving strategy to use (first), what task model or reasoning method to apply, etc. Since the role of support knowledge is to take some of the burden away from the problem solver or learner, and since the framework does not assume that support knowledge is entered, it is not further discussed in this report.
2. Explaining candidate facts

EXPLAIN starts to work on the activated concepts, and its job is to use the knowledge (and the user, if necessary) to justify, confirm or reject candidate facts by producing supporting explanations. For example, if two activated facts contradict, the one with the strongest explanation will be chosen. The strength of explanatory support depends on the strength of single explanation chains as well as the number of alternative explanations that support the fact. The goal of the reasoning process, e.g. the particular subproblem to solve, focuses the generation of explanations. The outcome of the explanation phase is a coherent knowledge structure and a set of supported hypotheses that are good candidates for achieving the goal of the reasoning process.

1. Focusing on a conclusion

FOCUS is the final step; it uses constraints and pragmatic criteria on the conclusion to check a candidate conclusion, or to select a conclusion if more than one candidate for satisfying the goal came out of the explaining phase. While EXPLAIN generates support and evaluates its hypotheses according to whether a conclusion makes sense and may be useful, FOCUS refines and adapts the candidate set in order to pick the best (i.e. most plausible) conclusion.

This three-phase model focuses on knowledge-intensive reasoning, but is also general enough to subsume most reasoning schemes used in AI systems. ACTIVATE may trigger hypotheses in a data-driven or goal-driven manner, by chaining of rules, retrieving of past cases, or inheritance in hierarchical knowledge models. It may associate facts on the basis of shallow or deep knowledge, and at the object level or control level. Many expert systems (e.g. OPS5-based, PROSPECTOR-like and MYCIN-like systems) do most of their reasoning within ACTIVATE. The EXPLAIN step plays a minor role in these systems, since they generally do associational reasoning based on shallow, compiled knowledge. SELECT may be a calculation of probabilities or a decision based on heuristics.

The model should not be viewed as a strictly sequential process. For example, instead of initially activating all possibly relevant structures, a reasoning process may activate one or a few structures, try to explain the consequences and expectations they generate, activate new structures, etc.

The model also serves as a framework for describing multi-paradigm reasoning: The three phase model may be applied to each reasoning method separately, in order to specify particular
characteristics of each method. It may also describe an integrated reasoning process, by describing the top level process within the same three-phase model.

3.5.6.1. Reasoning from Past Cases

The reasoning model described above subsumes knowledge-intensive case-based reasoning. A generic model of reasoning from past cases may be viewed as containing the following three steps (illustrated in figure 3.16):

1. ACTIVATE: Retrieve a set of cases that matches the current problem according to some criteria. This step activates cases that will be the subject of further examination.
2. EXPLAIN: Evaluate the applicability (relevance) of retrieved cases to the current problem. This involves looking for contradictions, checking constraints, generating and testing expectations set up by the retrieved case.
3. FOCUS: Adapt the solution (or solution method) of the case to the new problem. The adaption may be a simple transfer of the past solution to the new problem, or an elaborative process of modifying the solution to better fit the current problem. A modification process would typically involve a generalization of the past solution followed by a specialization satisfying constraints on the current problem.

![Figure 3.16: Main Steps in Reasoning from Past Cases](image)

These steps are - at the top level - specializations of the ACTIVATE-EXPLAIN-FOCUS stages. At a more detailed level, each of the steps in Retrieve-Evaluate-Adapt may be regarded as separate tasks. The ACTIVATE-EXPLAIN-FOCUS process - being a general model of explanation-based reasoning - is then recursively applicable to each task\(^1\). This illustrates the generic role of the reasoning model as a generator and evaluator of explanations.

\(^1\)The whole process is controlled at the task level, using criteria for evaluating the conclusions of each sub-process - along with threshold values for their acceptance/rejection. This issue will be further elaborated when discussing the CREEK approach, chapter 5.
3.5.7. An Explanation Based Model of Sustained Learning

The learning model of this framework is tightly integrated with problem solving. The focus is on how to capture relevant experience from a problem solving session, and make it available for solving similar problems later. The learning model integrates a case-based and explanation-based learning approach with a learning apprentice view.

Schank has studied explanation mechanisms related to learning in people [Schank-86a, Schank-86c], and views learning as a process of explaining a failed expectation:\footnote{From [Scank-86c], page 144.}

What situation needs to be explained? People have powerful models of the world. Through these models, which are based on the accumulated set of experiences that a person has had, new experiences are interpreted. When the new experiences that a person perceives fit nicely into the framework of expectations that has been derived from experience, an understander has little problem understanding. However, often a new experience is anomalous in some way; it doesn’t correspond to what we expect. In that case, we must reevaluate what is going on. We must attempt to explain why we were wrong in our expectations. We must do this, or we will fail to grow as a result of experiences. Learning requires expectation failure and the explanation of expectation failure.

Within the framework presented here, learning in a knowledge rich domain is viewed as a process of

1. Observing phenomena
2. Inferring consequences, i.e. generating expectations
3. Trying to explain what is observed and inferred
4. When expectations fail, trying alternative explanations or asking for more information from the external environment
5. Extracting new knowledge relevant for future problem solving
6. Updating the knowledge model

Since learning within our framework is tightly connected to problem solving, the steps from observation through explanation of expectation failures is assumed to be part of the problem solving process, as described by the two preceding process models. When the learning ‘starts’, the system state is an activated structure of knowledge representing an understanding of the actual problem, and a plausible, explained solution. This activated structure should not be regarded as fixed, however, since the learning task may infer new information that leads to contradictions within the structure. Resolving such contradictions may introduce changes in the knowledge structure. Given this assumption, the learning process may be described generally as follows:
1. Extracting learning sources
2. Constructing new knowledge structures
3. Storing and indexing

These steps - with corresponding types of input/output data - may be illustrated as below:

![Figure 3.17: A Model of Learning from Experience](image)

The knowledge structure that has been activated during problem solving is the primary source of learning. A subset of structures (e.g. input descriptors, goals, successful solution path, failed attempts) is extracted as the 'training set' to actually learn from. The middle step - CONSTRUCT - may be viewed as the primary learning step. Finally, the new case and/or general knowledge structures that have been constructed are associated with index concepts (important features, remembrings to similar cases, etc.) for later retrieval.

1. Extracting learning sources

This subprocess is active throughout problem solving, and its task is to keep track of information and knowledge that will later be used as sources of learning. The kind of knowledge relevant to mark as candidates is determined by the learning method. Some methods use the problem description given as input to the system together with the problem solution as the only type of candidate examples for learning, while other methods also learn from inferred facts and traces of the problem solving process. The set of learning sources form a selected training set that is given as input to the constructive learning method.

This step is absent in knowledge-poor, similarity-based learning methods: Since no knowledge model is available to guide the selection and extraction of relevant training instances, they are all regarded as equally relevant.

2. Constructing new knowledge structures
This step is, generally, based on methods for constructing cases, for inducing general rules from a set of cases, and for modifying knowledge structures by integrating new knowledge that has been inferred or accepted from the external environment.

The methods relevant for our descriptive framework are case-based and explanation-based methods. A method for storing a problem solving experience in a new case may be characterized by the answers to the following four questions:

- What determines whether a new case should be constructed?
  - degree of dissimilarity with closest case
  - failed attempt to solve problem
  - successful attempt to solve problem

- What is to be stored in a case?
  - problem description
  - solution
  - trace of problem solving path
  - dead ends (attempted paths that failed)
  - explanation chains
  - dependencies, preconditions for activation
  - remindings to other cases

- How general is the contents of a case?
  - cases are distinct individuals
  - cases are hierarchical structures
  - cases are generalized to rules

- How can the existing case structure be modified?
  - changing strength of remindings
  - merging with new case

Within the framework, explanation-based methods are related to the notion of knowledge integration, i.e. integrating a new piece of knowledge by explaining it using existing knowledge. In other words, new knowledge is learned in the sense that it is 'understood' by being explained on the basis of what is already known. Models of learning as knowledge integration have been developed, e.g., by Murray and Porter (see [Murray-88a, Murray-88b], and Van de Velde [Van de Velde-88b, Van de Velde-89]). The model of Murray and Porter has been developed while conducting experiments within a very knowledge-intensive environment: The CYC system [Lenat-89, Guha-90]. The learning task investigated is in itself knowledge
integration, i.e. how to ‘assimilate’ a new piece of knowledge that is presented to the system. The model identifies three knowledge integration tasks:

1. Recognition - identifying relevant knowledge structures to guide the interpretation of new information
2. Elaboration - expanding the information content by applying expectations set up by the retrieved knowledge structures to determine consequences of the new information
3. Adaption - modifying the knowledge base to accommodate the elaborated information

New knowledge structures are constructed and existing structures are modified by a method of continuous refinement of the knowledge structure. Knowledge is modelled as a network of concepts, where entities as well as relations are explicitly defined. The knowledge is basically of the deep, elaborative type, but associational knowledge may be incorporated as well.

These knowledge integration steps may be viewed as a special case of the EXTRACT-CONSTRUCT-STORE cycle of our framework - focusing on additions and modifications of the basic conceptual knowledge model by explaining consequences of new knowledge. Although generating and evaluating explanations guides both the EXTRACT and STORE tasks, knowledge integration as a learning method - as expressed by the Recognize-Elaborate-Adapt model - should be viewed as a subtask of the CONSTRUCT phase: CONSTRUCT builds or modifies knowledge structures by attempting to integrate candidate structures into the knowledge model. If the integration attempt is unsuccessful, the candidate structure is rejected, modified or presented to the user - depending on the actual learning method and decision criteria of an actual system.

3. Storing and indexing
Learning is not successful unless updates and refinements are available for new problems. The problem of storing involves deciding in which form the knowledge should be saved, and where it should be put. Essential questions for characterizing a case-based learner's storing and indexing scheme are:

- Are cases separate knowledge units, or are they virtual structures of connected nodes?
- How are cases indexed (if at all)?
- What kind of indexing vocabulary is used?

The storing problem related to indexing for efficient later retrieval, is the main task here. If knowledge is learned in a generalized form, and integrated into a conceptual knowledge model or a rule set, the problem is what existing concepts and rule structures to update. If a case is to
be stored, the problem is what concepts to use as pointers to the case, and how to structure the index network. In both situations it is a question of creating the necessary remindings and associations to new pieces of knowledge (concepts, cases, rules).

**Testing and evaluation**

The learning steps described will ensure that the knowledge base gets updated after a problem has been solved. These updates are assumed to improve the system’s ability to solve new problems, but in order to gain confidence in the correctness of the learning, the modifications should be tested on a real problem. One way to do this is to enter the same problem once more, and observe if the system now reaches the right conclusion. If this for practical reasons is difficult to do at once, the solution should be marked as a non-evaluated solution, and the mark not be removed before the case later is used to successfully solve a problem. If such a case later turns out to lead to an erroneous solution, explanation-based failure-recovery methods should be able to guide the user in repairing the fault. This process may lead to modifications of the case description, modifications of the general knowledge model, or both.

### 3.5.7.1. Learning by Retaining Past Cases

![Figure 3.18 Main Steps in Learning by Retaining Past Cases](image)

It should be clear from the previous section that case-based learning is the primary learning paradigm within this framework. In order to cover the integration of explanation-based methods, The EXTRACT-CONSTRUCT-STORE model illustrated in figure 3.17 is at a level of generality that also subsumes the explanation-based subprocesses. Focusing on the case-learning mechanism alone, the model gets specialized to the one illustrated in figure 3.18.
3.5.8. An integrated model of problem solving, reasoning and learning

Figure 3.19 summarizes the framework’s model of problem solving, reasoning and learning. It shows how the submodels presented in this chapter interact and cooperate. The middle part of the figure shows the basic problem solving steps: UNDERSTAND-GENERATE-SELECT.
The underlying reasoning process within each phase is represented by the ACTIVATE-EXPLAIN-FOCUS cycle. During problem solving, and after the problem has been solved, the learning process extracts relevant sources for learning, which is used to construct new (or modified) knowledge structures. The final step of the EXTRACT-CONSTRUCT-STORE learning process is to index and save new knowledge constructs for retrieval in future problem solving. As the input arrow to the Solution box indicates, the model assumes that a system receives feedback to whether the solution was successful or not. This is crucial information for the learning process, and should trigger an interaction between a competent person and the system in order to learn from the failure.

3.6. Chapter conclusion

This chapter has described a framework for specification and analysis of knowledge-based systems that continually learn from problem solving experience. The objective of the framework is twofold: One objective is to serve as a modelling approach for competent and robust knowledge-based systems that maintain their competence by learning from experience. The other objective is to serve as a unified reference system for description and comparison of existing systems and methodologies.

The framework represents a perspective on knowledge, reasoning and learning that aims at satisfying three fundamental system requirements. All requirements are based on the assumption that competent and robust systems need a thorough, relatively deep knowledge model as a fundament for understanding, i.e. as a background pool of knowledge from which plausible explanations are generated to support reasoning steps and learning decisions concerning the more shallow, associational knowledge in past cases and heuristic rules.

First, the multiple types and levels of knowledge that need to be modelled require an expressive and extendible knowledge representation formalism. This has been achieved by defining a conceptual model - in which all knowledge to be reasoned about should be explicitly represented - as a tightly coupled structure of entities and relations. A relation, as well as an entity (object, method, procedure), is viewed as a concept to be explicitly defined by its relations to other concepts. A model of representational terms is defined - at the knowledge level - which enables a description of the kind of knowledge content a system is able to represent.

Second, reasoning by matching, retrieving and adapting past cases to new problems needs to be combined with reasoning from more general domain knowledge, captured within a conceptual model, or within heuristic rules. This has been achieved by defining two process models: A
strategic level problem solving model (UNDERSTAND-GENERATE-SELECT) to control the multi-paradigm reasoning process, and a task level reasoning model which specifies a generic structure (ACTIVATE-EXPLAIN-FOCUS) for making inferences by an integration of conceptual model knowledge, past cases, and heuristic rules. Both process models rely on explanation methods that utilize the general domain knowledge to guide, support and justify strategic and task level reasoning steps. An explanation method produces and evaluates explanations, which are paths in the conceptual network.

Third, the case-based learning method needs to be integrated with methods that guide and justify inferences of the learning process by explaining the relevance and plausibility of derived results. This has been achieved by defining a learning model (EXTRACT-CONSTRUCT-STORE) that extracts relevant knowledge types for learning, builds a new case (or modifies existing cases) by explaining that the new case construct is a correct and useful representation of the problem solving experience, and finally indexes and saves the case for future use.

In the following chapter, the framework will be used to describe existing approaches to knowledge-intensive, multi-paradigm reasoning and learning, and to compare them with respect to how well they satisfy the three requirements.
Chapter 4

Competent Problem Solving and Sustained Learning
- A Review of Current Approaches

4.1. Essential Properties and Current Systems

The preceding chapter presented a framework for a knowledge intensive approach to integrated problem solving and sustained, case-based learning. In this chapter, important properties of relevant existing approaches are described with reference to the framework. A comprehensive study of current research has been undertaken in order to identify existing systems and ongoing research that address these problems. Case-based reasoning and learning is a young research field, but a few relevant systems and methodologies have emerged. The purpose of this chapter is to evaluate to what degree the most promising of these approaches satisfy our requirements.

All the systems reviewed have the following characteristics:

- Incremental, sustained learning - they learn incrementally from each problem solving case.
- Case-based learning - they retain cases for future use in problem solving.
- Explanation based learning - they utilize an existing knowledge model to actively support the learning process.
- Multi-paradigm reasoning - they integrate reasoning from past cases with reasoning from a model of more general knowledge.

Four current systems are discussed. Two systems have been found to be particularly relevant, and will be discussed in more detail than the others:
In addition to the characteristics above, these two systems address classification type problems - diagnosis and repair tasks. The following two systems also suggest interesting system architectures, although they address a different type of problem domain: Planning problems.

CHEF - developed at Yale University [Hammond-87]
JULIA - under development at Georgia Institute of Technology [Kolodner-87]

These systems have arisen from the latest research into methods for integrating problem solving and sustained learning. Not all of them have been developed primarily for attacking the problem of sustained learning per se. Machine learning research is generally concerned about how to train a system to become a better problem solver, and a distinction between a system’s training period and its operational problem solving phase is often assumed. This distinction is not important here, however, as long as the resulting system is able to continually learn during operation. (Some systems, e.g. Protos, have a user settable switch for choosing between optimization of performance or learning).

In addition to these systems, some recently reported work in progress have been studied with particular interest, although they are not discussed in this report. This includes a system for legal reasoning, under development within Porter’s group at the University of Texas [Branting-89, Porter-89], an architecture for mixed case-based and rule-based reasoning being developed within Rissland’s group at University of Massachusets, Amhearst, and a particular approach to combining explanation-based and case-based methods taken by Schank and his group at Yale University [Schank-89], where the explanation process itself is viewed as a case-based reasoning process in the sense that the cases themselves are explanations or generic explanation patterns. Interesting work on industrial applications of case-based reasoning is being done at, e.g., the Lockheed AI Center in Palo Alto (case-based management of autoclaving - the Clavier system [Barletta-88, Mark-89]), and the University of Kaiserslautern (CNC-machinery operations - the MOLTKE system [Althoff-89]).

4.2. Critical Factors

The four systems listed in the preceding section will be described according to the descriptive dimensions listed in section 3.3. Following this description, the systems will be discussed and compared with respect to a set of factors that are critical in achieving the system properties
specified in the framework (requirements R1-R3). These are qualitative factors, and not suitable for any kind of quantitative, objective assessment of the systems. Nevertheless, by focusing on the core properties needed to build knowledge intensive problem solvers that learn from experience, these factors are crucial in judging to what degree the system requirements of the framework are met.

The following critical factors will guide the comparison of each system with respect to the framework requirements:

**CF1. Thoroughness of knowledge model.**
This factor characterizes the quality and role of the conceptual knowledge model:
To what degree is the knowledge model able to provide the system with the understanding necessary for competent and robust problem solving, and for knowledge base updating through experience? Is there an explicit model of problem solving strategy and reasoning structure that actively controls the problem solving and reasoning processes? What are the strengths and weaknesses of the knowledge model?
Descriptive data points that have strong impact on this factor are:
- Limitations on expressiveness
- Integration aspects
- Problem solving strategy

**CF2. Active role of explanations**
This factor characterizes the active role of explanations in reasoning and learning. To what extent are the explanation-based methods able to utilize the knowledge model to a desired degree? What are their strengths and weaknesses?
Descriptive data points that have strong impact on this factor are
- Structure of explanations
- Reasoning structure
- Learning structure

**CF3. Combined use of reasoning methods**
This factor characterizes the control and operation of multi-paradigm (mixed) reasoning. To what degree do the reasoning methods mutually support each other, and to what degree is the combined reasoning explicitly controlled? What strengths and what weaknesses does the integration of reasoning methods exhibit?
Descriptive data points that have strong impact on this factor are:
- Reasoning methods
- Control of multi-paradigm reasoning
CF4. Combined learning methods

This factor characterizes the integration of learning methods. It involves primarily explanation-based support, but may also involve similarity-based methods working in conjunction with the case-based learning process. What are the strengths and weaknesses of the model support methods for learning cases and general knowledge?

Descriptive data points that have strong impact on this factor are:
- Case learning
- General knowledge learning

4.3. PROTOS

4.3.1. General

Overview
Protos\(^1\) [Bareiss-88a, Bareiss-88b, Porter-90] is an exemplar-based approach to learning and classification problem solving. An exemplar-based method is a type of case-based method where solved cases are stored exclusively as concrete, non-generalised exemplars. A concept is viewed extensionally, as a category defined by the collection of exemplars (members) that belong to the category. A category’s exemplars are ranged according to prototypicality, estimated by how successful an exemplar is in problem solving (i.e. in matching new cases successfully). Protos addresses classification type problems, and a system for diagnosing hearing disorders has been developed as a test application for refinement and evaluation of the methodology. The learning task is concept learning, where the concepts to be learned are diagnostic categories. A semantic network of categories and features forms a model of general domain knowledge.

An example of such a category structure is shown in figure 4.1. The case matching process is explanation-based, in that an attempt is made to match two non-identical features by searching for a path (an explanation chain) in the semantic network that connects them. Two features match if the strength of the explanation chain is above a certain threshold. The strength of a particular relationship (explanation unit) is a function of a numerical default value assigned to each relation in the category structure, associated qualifiers (e.g. always, sometimes, possibly), and contextual constraints.

\(^{1}\)Protos was originally implemented in Prolog, and has later been rewritten in Common Lisp. The Lisp version is almost identical to the Prolog version, but contains some minor enhancements and limitations. The original Prolog version has also been further developed. This description is based on the documentation of Protos as reported in [Bareiss-88a].
A typical problem solving session
A problem is described as a set of features, and the system’s task is to retrieve the exemplar that best matches the feature set describing the problem. A new problem case is assumed to be an exemplar of the same category as the best matching previous exemplar. The diagnostic category of the retrieved exemplar is therefore proposed as a solution to the new problem. If the proposed solution is rejected by the user, a learning session - controlled by the user - is initiated. The user is forced to define entered terms that are unknown to Protos, by describing the terms’ relations with existing terms in the category structure. If the proposed solution is rejected by the user, Protos is either asked to look for alternative solutions, or accept a solution from the user.

Figure 4.1: A Category Structure in Protos
The figure is reproduced from [Bareiss-88a]. The example illustrates a category structure for learning of the concept "chair". Chairs is the category, and Chair 1 and Chair 2 are exemplars (past cases) with features as indicated by unlabelled arrows. These nodes are embedded in a semantic network of domain knowledge concepts and relations.
After a problem is solved, learning takes place by incorporating new user supplied knowledge, and by extending or modifying the exemplar set and the indexing structure.

### 4.3.2. The Problems Addressed

Protos addresses the knowledge acquisition problem by suggesting a bottom-up approach to knowledge base construction: A knowledge model is gradually built by focusing on actual problem cases. It addresses classification problem solving by retrieving a previously solved case, and by taking the class to which the matched case belongs as a solution to the new problem. It addresses the concept learning problem by viewing a concept definition as a collection of exemplars with varying degree of concept prototypicality, and the sustained learning problem by integrating each solved problem case into the category structure.

#### Type of Problem to be Solved

<table>
<thead>
<tr>
<th>Problem type:</th>
<th>Diagnosis in natural domains.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application focused:</td>
<td>Diagnosis of human hearing disorders.</td>
</tr>
<tr>
<td>Type of input:</td>
<td>Qualitative values of observed symptoms, measurements, test results, etc.</td>
</tr>
<tr>
<td>Type of output:</td>
<td>A disease (single value).</td>
</tr>
<tr>
<td>External environment:</td>
<td>Heavy user interaction - system is an assistant.</td>
</tr>
<tr>
<td></td>
<td>Self-contained system - no linking to other computing system.</td>
</tr>
</tbody>
</table>

#### Type of Learning

<table>
<thead>
<tr>
<th>Learning type:</th>
<th>Learning of natural concepts, i.e diagnostic categories. General domain knowledge is supplied by the user.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of input:</td>
<td>A set of relevant features describing the problem, the assigned diagnostic category, a previous case. Failed attempts.</td>
</tr>
<tr>
<td>Type of output:</td>
<td>A new case linked within the category structure, modified category structure.</td>
</tr>
<tr>
<td>External interaction:</td>
<td>Learning apprentice approach. If Protos fails, the user/expert is asked to solve the problem, and the system learns by explaining the case within the semantic domain model. Unexplained terms have to be explained by the user.</td>
</tr>
</tbody>
</table>
Protos has been evaluated against a group of senior medical students, using a set of more than 200 audiology cases. Protos performed well: After it had seen 150 cases, it classified new problems at the level of the students in the reference group.

4.3.3. The Knowledge Model

Knowledge Types

The knowledge in Protos is almost exclusively at the object level. This knowledge is held in the category structure, as illustrated in figure 4.1. The knowledge model contains the knowledge types shown in figure 4.2.

![Figure 4.2 Protos’ Knowledge Dimensions](image)

A subclass hierarchy of important knowledge types in Protos is shown. The figure should be related to figure 3.1.

There is no problem solving strategy or reasoning model explicitly represented. There is one type of control knowledge, however, related to the searching for good explanations. This is a set of heuristics for selecting the best continuation of an explanation chain. The heuristics are used to explain a feature-to-feature match or the relevance of a feature to a category. Examples:
• A multi-step correlational explanation is weak, especially if it involves more than two inference steps.

• The length of a sequence of causal relationships should not diminish belief in its quality.

The category structure of PROTOS captures deep, descriptive, object level knowledge of the domain. The exemplars contain shallow associations between features and diagnostic categories, and its main role is as operational problem solving knowledge. At the control level, explanation heuristics are shallow, operational knowledge that has been abstracted from a deeper analysis of each relation’s role in explanation. The knowledge forms of concept network, cases and rules, represent the semantic network of the category structure, exemplars, and explanation heuristics, respectively.

Figure 4.3 shows the components of expertise in Protos. Problem solving and learning strategies are in general fixed algorithms, and not explicit models.

Figure 4.3: Expertise Components of Protos
Protos has explicit models of general knowledge, past cases, and a set of heuristics to guide explanation construction. The problem solving strategy, task structure and reasoning structure and methods are implicit, hidden in algorithms, and inaccessible for inferencing.
Representational Terms

Protos’ model of representational concepts is illustrated in figure 4.4. The primary term is the category, which is defined by the properties of its members, and by relational links to other categories. Thus, a concept is not purely extensionally defined: In a classification system built

![Diagram of representational primitives in Protos](image)

**Figure 4.4 Representational Primitives in Protos**

The figure shows basic terms for representing knowledge in Protos. The structure shown is a specialization of the generic structure of our framework, shown in figure 3.10. Important characteristics include a (basically) extensional definition of concepts, and a rich set of terms to describe relations. On the other hand, a relation is merely a program-defined link between concepts - not an explicitly defined concept as well. All bi-directional relations have inverses.
by Protos, a class is defined primarily by its instances, but also (intensionally) by semantic relations with other concepts. The structure and instances of semantic relations and relational qualifiers is shown in the figure. Indexing links serve the purpose of fast exemplar retrieval during case matching.

A reminding may be regarded as a compiled explanation chain, directly associating a feature to another feature or to a category. Censors are also remindings, but with a negative sign - serving to avoid a match that previous experience has shown to fail. An exemplar link is a particular type of member link, connecting a category to its exemplars.

A difference link is a pointer that may be established between exemplars that share some, but not all, important features. Difference links get established as a result of failed, but close, matches during problem solving.

**Representation of General Knowledge**

General domain knowledge is represented as a semantic network of categories, features and relations. A category is represented by its name, its set of exemplars, and its set of links to other categories. A feature is a category which may or may not contain exemplars. A feature is represented either by an atomic term or a predicate with arguments. A relational link, generally, has a relation name, a list of qualifiers, a from node and a to node. Categories, links and qualifiers are put together to form relationships. For example:

- legs(0) is-consistent-with bean-bag chair
- legs(5) and swivel moderately-suggest office-chair

Relational links and qualifiers have strength values associated with them, that are combined to give the resulting strength of a relationship. Strengths range between 0 and 1, and the strength of a qualified relationship is computed by simple multiplication, e.g.:

- strength(moderately) = 0.7, strength(suggest) = 0.8, strength (moderately suggest) = 0.56

Since the meaning of a relationship - and hence its explanatory power - usually depends on context, a relationship may be preceded by a conditional term. The context is determined by the category under consideration, e.g. (from [Bareiss-88a]):

*if* the category is apples *then* color(green) is sometimes equivalent to color(red).

The network of categories and links defines the object level domain category structure. All general domain knowledge is captured within the category structure. Heuristic rules may be
expressed within the category structure, by choosing an appropriate ‘shallow level’ relation (e.g. implies, suggests). However, the role of heuristics are the same as for deeper knowledge: To produce explanations for justification and guidance of the case-based reasoning and learning processes. There is no rule-based reasoning mechanism in Protos.

Protos represents features as non-structured, distinct concepts. For example, the feature Legs(4), as shown in figure 4.1, is a distinct concept. Adding a chair with 5 legs would lead to a new concept being defined - Legs(5). There is no convenient way to express - and reason with - structured concepts in Protos\(^1\).

Protos handles qualitative feature values only. Numeric values may be given, but they are interpreted qualitatively (e.g. Legs(5) may equally well be expressed as Legs(five)).

**Structure of Explanations**

An explanation is a chain of relationships between two features or between a feature and a category. For example (see figure 4.1), the equivalence of Legs(4) and Pedestal with respect to the category Chairs, is established by the explanation:

```
Pedestal specialization-of Seat-support,
Legs(4) specialization-of Seat-support,
Seat-support enables Holds(person),
Hold(person) function-of Chairs.
```

An explanation is accepted if its strength is above some threshold value.

**Representation of Cases**

An exemplar is represented by its name, its set of features, and the category to which it belongs. Each feature in a stored exemplar is associated with a numerical importance value. This value expresses how important the feature is for classifying the exemplar as a member of the category.

Exemplars are indexed by remindings, censors, and difference links. Within a category, the exemplars are ordered according to degree of prototypicality.

All features of a new problem are potential indexing features for retrieving a past case. There is no fixed indexing vocabulary.

\(^1\)Although structured features are not enabled, a kind of inheritance interpretation method is implied by the following explanatory heuristic, related to the specialization/generalization relation:

“When relating a feature to a category, traversing a specialization link from a more general category is a reasonable step in the explanation.”
Integration Aspects
An exemplar is tightly integrated into the category structure by having its features and category class defined within this structure.

4.3.4. The Problem Solving Process

Protos utilizes general domain knowledge in its problem solving, but it does not require that a knowledge model is present. The way Protos is designed facilitates incremental development of the general knowledge model, by having the expert define relationships for all unexplained features as new cases are entered. As more cases are entered, the knowledge model grows and gradually increases the amount and quality of support for the case-based process.

Problem Solving Strategy
There is no reasoning at the strategic level in Protos. The problems presented to Protos are all of the same type, and require the same type of actions. A single built-in strategy combines featural remindings in the new case to retrieve a best matching exemplar, and proposes the exemplar's solution as the solution to the new problem. Protos has no mechanisms for modifying a past solution; instead, the solution is presented to the user for confirmation, rejection or modification.

Problem Solving Tasks
A three-step top-level problem solving process may be described by the following phases and tasks:

1. UNDERSTAND: • Combine remindings from input features.
   • Retrieve exemplar.
   • Justify the match.
2. GENERATE: • Copy solution of matching exemplar.
3. SELECT: • Suggest copied solution as solution to new problem
   • Discuss suggested solution with the user.

4.3.5. The Reasoning

Reasoning Methods
The reasoning method is case-based, or - more precisely - exemplar-based. The sub-processes of feature matching, and of evaluating relevance of features to a category, are supported by semantic network model reasoning. There are no methods that use the knowledge model to derive a solution if the case-based method fails.
### Reasoning Structure

Figure 4.5 shows Protos reasoning structure, expressed within the ACTIVATE-EXPLAIN-FOCUS model. The reasoning model of the framework is viewed as a generic model that may apply to all three problem solving phases. In an implemented system, however, a particular reasoning step may apply to only some of the problem solving phases. For Protos, as well as the other systems analysed, the reasoning model turned out to be more like a specialization of the entire problem solving process, than a description of a single sub-process. An interpretation of this is that current systems have not yet reached the level of knowledge-intensive support, in all subprocesses of problem solving, that is covered by the framework.

All features are regarded as relevant problem descriptors, and the first thing Protos does is to check whether some input features trigger remindings to exemplars or categories. This set of categories and past cases (called hypotheses) are sorted according to strength of remindings\(^1\). If no remindings are triggered, Protos is unable to classify the case. Exemplars typically also contain other features than those responsible for the remindings. That is, a retrieved exemplar sets up expectations that need to be explained. The next step is therefore to pick the strongest

\(^1\)Multiple remindings to one exemplar or category are combined according to an algorithm that considers strengths of remindings and censors, and inheritance of remindings through specialization links and exemplar links.
reminded exemplar\(^1\), and assess its similarity to the new case by attempting to explain all its features. This is done by using the domain knowledge in the category structure. Depending on the success of this explanation process, the exemplar's solution is assigned to the new case, or the exemplar is rejected and the next best hypothesis is tested for similarity. The user may be consulted in these decisions.

**Control of Multi-Paradigm Reasoning**

Case-based reasoning is the only problem-solving paradigm. Model-based reasoning is used only in generation and evaluation of supporting explanations. All control is through the case-based process.

### 4.3.6 The Learning

Protos always learns from a problem solving case:

- If a problem is successfully solved in the first attempt, no new case is constructed, but the remindings from relevant features to the case are strengthened.

- If a problem is successfully solved in second or later attempts, Protos tries to find the cause of the initial failure. Protos learns from the failure by weakening remindings from the features to the faulty retrieved case, and may suggest modifying the relative importances of each feature for classifying the case.

- If Protos is unable to suggest a solution, the case is stored as a new exemplar. Remindings to the case from discriminating features are generated, and difference links to similar cases are installed.

During the learning process, the user is asked to confirm or change suggested modifications to the case structure, and to revise explanations if needed.

**Structure of the Learning Process**

Figure 4.6 shows the EXTRACT-CONSTRUCT-STORE process structure for Protos. The sources of learning are the recently solved case (set of features, and solution), failed attempts to match the new case, and the successfully matched exemplar. The starting step in learning is having the expert/user explain features in the new case that Protos failed to explain during problem solving. If the similarity between the new case and the retrieved exemplar is sufficiently close, Protos will merge the two cases, otherwise a new exemplar is created. Merging involves increasing the exemplar's prototypicality, and may include generalization of features that have a common node along specialization-of, caused-by, or function-of relations. Protos performs no such generalizations on its own; the user is always asked for confirmation.

---

\(^1\)If a category is reminded of, its most prototypical exemplar is chosen.
Storing a new case involves assessment of feature importances, and generation of remindings, censors and difference links. These tasks are executed by algorithms supported by domain heuristics. Typically, Protos proposes a construct, which the user confirms, modifies or rejects. Existing cases that have generated failures in the attempt to solve the problem, get their remindings, prototypicalities and feature importances re-evaluated.

Figure 4.6: Structure of the Learning Process in Protos
The EXTRACT-CONSTRUCT-STORE structure of Protos, (see figures 3.16 and 3.17 for the corresponding generic structures within the descriptive framework). The user is plays an active role in all but the first step.

Case Learning
A new exemplar contains:
- the set of features that proved relevant for the case during problem solving
- the solution, i.e. the category that successfully classified the problem

The exemplar is stored within the category structure by creating explanation links to unexplained features, and is indexed by remindings, censors and difference links. All features are candidates for indexing, there is no limited set of indexing terms.

General Knowledge Learning
General domain knowledge is acquired through user-supplied explanations. This is an active process, where Protos checks whether a new link should be added as it is given, whether it
should replace an existing link (e.g. an existing conditional link, weaker than the new link, is replaced by the new link), or whether there is a conflict that needs to be settled by the user.

**The User's Role**

The user plays an active role in the entire learning process. Protos is a typical learning apprentice, which attempts to solve problems - and learn from the experience - on its own, but relies on an expert or skilled user to supply the necessary explanations when an attempt fails. Gradually, of course, Protos will be able to handle an increasing number of situations on its own.

### 4.3.7. Relating Protos to the Critical Factors

Below, important properties of Protos - as seen by the framework described in chapter 3 - are summarized by relating them to the critical factors defined in section 4.2. A '+' sign indicates an advantageous property, while a '+' sign indicates a missing property or a disadvantage:

**CF1: Thoroughness of knowledge model**

- Strong set of relations, enables knowledge modelling beyond structural and causal relations.
- Explanation generation mechanism that takes advantage of the relation language.
- Qualifiers and conditionals to moderate the strengths of relations.
- Fixed set of relations, difficult to extend or modify.
- Relations are built-in terms, not concepts explicitly defined.
- Category and feature terms are atomic, no structuring of features facilitated.
- Incrementally acquired knowledge model, no model support to begin with.
- No explicit diagnostic strategy, task model or reasoning structure.

**CF2: Active role of explanations**

- Explaining similarity of different features, within context.
- Explaining unexplained features when learning a new case.
- Control heuristics for building explanation paths.
- No use of explanations to induce generalised features or categories.
- Explanation paths are not stored within cases, disabling justification of match by matching of explanations.

---

1 Even if it is possible to enter explanations independently of discussing cases, this involves working against the idea and design of Protos. In particular, general knowledge can not be modified unless referred to in a case discussion.
CF3: Combined use of reasoning methods

+ Model-based reasoning in matching syntactically different features, and in explaining featural relevance to a category.
÷ No model-based reasoning method to fall back on if case-based reasoning fails.
÷ No problem solving or reasoning knowledge at the task/strategy level.

CF4: Combined learning methods

+ Active interaction with the user.
+ Learning remindings by 'compiling' explanations.
÷ Limited use of the system's knowledge model for learning, the main model is the user.

4.3.8. Relating Protos to the Framework Requirements

A Summary of Important Characteristics.

Protos represents an extensional approach - an exemplar approach - to concept definition. Classification problem solving is a process of matching a problem description to a previous exemplar, and choosing the matching exemplar's category as the category for the problem case. The process is supported by explanations generated from a model of general domain knowledge, incrementally developed by focusing on knowledge related to each problem as it is presented to the system. The general knowledge is used to explain similarity of syntactically different features. The solution of the best matched case is suggested as the solution to the problem, to be confirmed or rejected by the user.

Protos learns from problem solving experience by creating new exemplars of problem cases, and through interaction with the user. All new knowledge entered is checked against the existing knowledge model, and contradictions are reported back to the user.

Below, some properties of Protos are assessed with respect to the framework described in chapter 3. Deficiencies of the system compared to the framework are not necessarily deficiencies of Protos as such, since the motivations, goals and requirements for developing Protos were different from ours.

Requirement 1: Expressive and Extendible Knowledge Representation Formalism.

Protos has a powerful and expressive language of relations, involving a comprehensive relation set, relational qualifiers and conditionals. This enables the modelling of precisely defined relationships, that help the generation of focused, context sensitive explanations.
The representational system has two major deficiencies. First, features and categories are not structured concepts composed of properties with values, but flat property-value pairs or single values. All structuring has to be done within the semantic network, leading to problems as previously illustrated (e.g., Legs(4) and Legs(5) being different concepts). This fact does not limit what is possible to express, but makes it more complicated to inspect the knowledge, and to reason with and learn structured concepts. Essentially, it reflects a view to conceptual modelling that is too simple to satisfy the requirements of the framework.

Second, relations are defined in the program code only. Relations are not regarded as concepts, i.e. as knowledge to be reasoned about and learned. The set of relations cannot be extended, their pre-defined explanation strengths not modified, etc., without changing the Protos program code.

**Requirement 2: Combining Case-Based and Elaborative Reasoning.**

The framework primarily addresses the problem of knowledge maintenance, while Protos addresses knowledge elicitation and initial modelling problems as well. The discussion in this subsection assumes that Protos has seen a significant number of cases, and has developed a relatively comprehensive general knowledge model.

Protos generates explanations from the general knowledge model to justify similarity of features during matching. The ACTIVATE step is not based on elaborations upon the problem description, but on remindings compiled from previous explanations. The explanation methods based on the relatively strong relational language gives Protos a knowledge intensive case matcher that retrieves past cases from semantical similarities with the problem. It is a strength of this method that control knowledge for generating a good explanation is modelled as a separate set of heuristics.

The major deficiency with respect to the requirement is that general knowledge is used only for feature matching. The semantic network knowledge is not utilised to solve a problem 'on its own', if the case-based method fails. Instead, the user is asked to solve the problem. Neither is general domain knowledge used to justify a solution, this is also left to the user.

Further, there are no mechanisms for modelling and reasoning with a strategy and task model of diagnostic problem solving. Hence, Protos is not able to distinguish between measurements, patient observations, symptoms, physiological states, etc.

---

\(^1\)This is one of the directions along which continued work on Protos has been undertaken.
Requirement 3: Combining Case-Based and Explanation-Based Learning.

When learning a new exemplar, Protos tries to explain input features of the problem that were not explained during problem solving. The features that Protos is unable to explain requires an explanation from the user, otherwise they are disregarded. General domain knowledge is also used during assessment of a feature’s importance for predicting a category.

Protos is a learning apprentice that relies heavily on its user. This is both a strength and a weakness. A positive effect is a quick adaption to the real world problem environment; the system will always be up to date with knowledge related to cases it has recently seen. The major weakness is that the knowledge model of the system will eventually represent a resource that is only partially utilized.

There is no learning of how to discriminate between relevant and irrelevant features of a problem description. Combined strengths of remindings are calculated using all the input features that are entered.

There are no methods for learning explicitly generalised concepts. This is a general characteristic of most case-based approaches, but integration of explanation-based methods potentially enables learning of explicit generalisations.

4.4. CASEY

4.4.1. General

Overview

CASEY [Koton-88, Koton-89] is a system that combines case-based and model-based reasoning. When a problem turns out to be unsolvable by retrieving a past case, a general domain knowledge model is used in a second attempt to solve the problem. The domain model also plays an active part in supporting the case-based reasoning and learning processes. The general knowledge model in CASEY is a pure causal model. The type of problem addressed is the diagnosis of heart diseases. CASEY essentially extends the Heart Failure program [Long-86] - a system for causal reasoning about heart failure - with a case-based reasoning and learning module. CASEY’s case memory structure is based on Kolodner’s self-organizing memory system [Kolodner-83a].

The basic idea investigated in Protos is not learning of generalised diagnostic class descriptions, so a lack of methods for explicit generalisation of concepts is what one might expect. Nevertheless, the learning of generalised features would potentially help the system in its case matching process. Matching by generating explanation paths to explain feature similarity could then be used if a generalised feature were unknown or only partially could solve the problem.
A problem is solved by retrieving a case, and adapting its solution to the problem. Each case contains a causal explanation that relates its features to the diagnosis. The solution to a new problem is derived by using the knowledge model to modify the explanation of the retrieved case.

**A typical problem solving session**

The input to CASEY is a set of features (feature-name feature-value) pairs describing the patient. A set of matching cases is retrieved, and the best explained case is chosen. The differences between the case and the problem description is then identified. Figure 4.7 shows the difference-features between the description of a patient, Uri, and a retrieved matching case, Sarah.

The middle part of the figure shows the causal explanation of Sarah’s diagnosis. Some of the differences between the two patients are explained by the causal model to be significant for heart failure (e.g. dyspnea, chest pain and chest x-ray). In order to use the explanation of Sarah’s case to solve Uri’s problem, the explanation needs modification. It is seen that the retrieved explanation does not cover for Uri’s chest x-ray value. On the other hand, it explains unstable angina, which is not a feature of Uri. The causal model is now used to modify the explanation, resulting in the explanation at the bottom of the figure. (History of anginal chest pain is a feature of both patients. Dyspnea on exertion is added as an unexplained feature).

The explanation for Uri’s case is the same explanation as the Heart Failure program ends up with, but CASEY’s case-based method is reported to be significantly more efficient.

If the user does not accept an explanation derived from modifying a retrieved case, the Heart Failure program is used to produce an explanation. In either situation, a new case is integrated into the memory structure.

### 4.4.2 The Problems Addressed

CASEY addresses classification problem solving through mixed paradigm reasoning, combining associational and interpretative reasoning. It starts out with a causal domain model, and addresses the learning problem by storing each case together with its causal explanation. The type of learning addressed is primarily learning of performance efficiency, since a problem is assumed solvable by the causal model.
## Feature Table

<table>
<thead>
<tr>
<th>Feature</th>
<th>Sarah</th>
<th>Uri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sex</td>
<td>Female</td>
<td>Male</td>
</tr>
<tr>
<td>Age</td>
<td>72</td>
<td>67</td>
</tr>
<tr>
<td>Dyspnea</td>
<td>None</td>
<td>On exertion</td>
</tr>
<tr>
<td>Chest pain</td>
<td>Unstable angina</td>
<td>None</td>
</tr>
<tr>
<td>Blood pressure</td>
<td>138/81</td>
<td>135/80</td>
</tr>
<tr>
<td>Heart rate</td>
<td>76</td>
<td>87</td>
</tr>
<tr>
<td>Chest x-ray</td>
<td>Normal</td>
<td>Aortic-valve calcification</td>
</tr>
</tbody>
</table>

### Diagram

- **Fixed Coronary Obstruction**
- **Regional Flow Deficit** ➔ **Unstable Angina**
- **Exertional Angina**
- History of anginal chest pain

---

**Retrieved explanation of Sarah’s heart failure**

### Modified Explanation

- **Aortic Valve Disease**
  - Aortic valve calcification

- **Fixed Coronary Obstruction**
  - Dyspnea on exertion

- **Regional Flow Deficit**
  - Exertional Angina
  - History of anginal chest pain

---

**Figure 4.7: An Example of Problem Solving in CASEY**

The figure is reproduced from [Koton-89]. Upper case items generally indicate physiological states, while bold items are diagnostic goal states. Lower case items are input features of the problem, and arrows symbolize the *causes* relation. Sarah is a stored case and Uri is the new problem. The upper part of the figure is a table that shows the difference between the cases Sarah and Uri (their similar features are not shown). The middle part is the explanation of Sarah’s case, while the resulting modified explanation that fits Uri is shown at the bottom.
Type of Problem to be Solved

Problem type: Diagnosis in natural domains.
Application focused: Diagnosis of human heart diseases.
Type of input: Qualitative and quantitative values of observed symptoms, measurements, test results, etc.
Type of output: A single or multiple disease.
External environment: No user interaction. System is linked to an existing system for diagnosing heart failures from a casual model of symptoms, physiological states, and diseases.

Type of Learning

Learning type: Learning of performance efficiency by incrementally learning useful associations between symptoms and diseases.
Type of input: A set of input features describing the problem, a set of physiological states that explain some of the features, a diagnosis, an explanation of the diagnosis, a treatment, a matching previous case.
Type of output: A new case indexed within the memory structure, modified feature importances.
External interaction: No user interaction. Heart Failure program used to solve the problem if the case model fails.

4.4.3. The Knowledge Model

Knowledge Types

Object level knowledge is contained within the causal model and cases (see figure 4.8). There is no explicit problem solving strategy or reasoning structure in CASEY.

There is no knowledge of rule form in CASEY.

CASEY uses a set of heuristic principles - called evidence principles - to justify a retrieved case, and a set of particular strategies (called repair strategies) to adapt the explanation of a case to a new problem. Examples:

Evidence principle:
States in the retrieved case that are incompatible with features of the problem must be eliminated. Incompatibility is defined as zero probability for a feature coexisting with a state.

Explanation repair strategy:
- When two numerical feature values have the same qualitative value, the cases' value gets replaced by the problem's value.

Figure 4.8: CASEY’s Knowledge Dimensions
The figure shows a subclass hierarchy of important knowledge types in CASEY. The figure should be related to figure 3.1.

In addition to strategies for repair of previous explanations, there are also repair strategies for adaption of previous diagnoses and therapies.

A component model of expertise in CASEY - covering explicitly as well as implicitly represented expertise - is shown in figure 4.9.

CASEY’s learning model is essentially a case-based learner, but it also contains methods for explanation-based and similarity based generalisation.
Representational Terms
CASEY does not possess a rich set of terms for talking about its knowledge (figure 4.10). Concepts in CASEY are either states\(^1\) or features. States are general physiological states, or goal states of the problem solver (diagnosis states or therapy states).

Representation of General Knowledge
General domain knowledge is contained within the Heart Failure program. Causal links connect states to other states and to features. When a feature is associated with a state via other relationships than causal, a general association link that includes a probability estimate is established between them. These two relations are the only types of links in the model. Control level heuristics are contained within procedures of program code.

---

\(^{1}\)In the framework described in chapter 3, a state was considered to be a conceptual primitive of the domain, not a representational term. In CASEY, state is used in both interpretations: a state has a representational and a conceptual aspect.
The unit of general, object level knowledge is the causal relationship, a composite structure of two states, or a feature and a state, connected by a causal relation. Causal relationships in the domain model have probabilities attached to them, ranging from 0 (expressing an incompatibility) to 1 (expressing a necessary effect). The model is represented as a causal inference network, where nodes are states and features.

A hierarchy of causal relations also implies a kind of generalisation hierarchy. The causal relation is transitive; hence, a feature that is caused by a state A, will also be caused by state B, if state B causes state A. In CASEY, the causal hierarchy is used to learn generalisations of features - by learning to associate features with states at a higher level than the states to which they are directly related.

Structure of Explanations.
An explanation is a chain of causal relationships from a state to another state or to a feature. Explanations generated by the Heart Failure model calculate a combined probability for the
presence of a feature given a state. Explanations stored within cases are just lists of states with no probabilities.

**Representation of Cases**

Cases are stored in a dynamic memory structure as described in [Schank-82] and [Kolodner-83a]. The structure is a discrimination network, where the top node contains common properties of all cases in the structure. Downwards in the memory structure cases are indexed according to their differences with other cases. The cases themselves (or pointers to them) are leaves in the tangled tree-structure. An intermediate node represents a generalised description of the cases indexed under the particular node (see figure 4.11). A feature is regarded as more general than another if it is contained within more cases than the other.

**Figure 4.11: The Memory Structure in CASEY**

Cases are stored in a discrimination network, where a node (gray in the figure) is either a generalised node - i.e. a node containing descriptors common to two or more cases - or a single case. Cases and general nodes are indexed by their differences with superior nodes. Each index is a two-level construct; first a discrimination between descriptor names is made, then a discrimination between values of the same descriptor name.

The contents of a stored case include:

- information about the patient, i.e. all the input features
- the patient’s diagnosis
- the therapy suggestions made
• the causal explanation of the input features
• the states for which there is evidence in the patient
  (called generalised causal features)
• the source of the solution (either a previous case or the Heart Failure program)

All features of a new problem are potential indexing features for retrieving a past case. There is no pre-determined indexing vocabulary.

Integration Aspects
At the representational level, the memory structure of cases and the Heart failure model are two separate bodies of knowledge. However, since all cases have been explained by using the causal model, the two models are integrated in the sense that some features and states referred to in cases are nodes in the causal network.

4.4.4. The Problem Solving Process

CASEY depends on - and utilizes - an existing causal knowledge model in its problem solving. The problem solving process is a sequence of case matching and retrieval, previous case justification, and adaption of the previous solution. If case matching and justification does not end up with a case that satisfies a similarity criterion, a solution is produced by the causal model.

Problem Solving Strategy
There is no explicit problem solving strategy in CASEY. The system is designed to solve one type of problem, using the same built-in strategy each time, as described in preceding sections.

Problem Solving Tasks
The top level problem solving tasks are:

1. UNDERSTAND: • Retrieve a syntactically similar case
   • Find states for which the input-features are evidence
   • Match retrieved case semantically
   • Justify explanation of retrieved case as relevant for problem

2. GENERATE: • Adapt solution of past case to the new problem
   • If case retrieval was unsuccessful, produce a new solution from the causal model

3. SELECT: • Present justified solution
Task 3, selection, is just a presentation of the solution arrived at. Once the presumably best match is found (end of task 1) CASEY sticks to this case until it terminates by presenting a solution to the new problem by modification of the existing solution. The solution is not discussed with the user, but if the solution is unsatisfactory, the user may of course rerun the system with a modified input data set. The other option is to manually modify or extend the causal model (the Heart Failure Model).

4.4.5. The Reasoning

Reasoning Methods
The reasoning method is a combination of case-based and model-based reasoning. The case based method is applied first, model-based reasoning within a causal model is performed if the case method fails to find a sufficiently similar past case. In addition to being a separate reasoning method, model-based reasoning also supports the case-based process.

Reasoning Structure
Figure 4.11 illustrates the reasoning structure in CASEY. The initial step is a registration of input features, corresponding to an activation of feature terms within the system. All features are initially regarded as relevant problem descriptors. Case matching is a two step process, involving a retrieval and a justification step.

![Figure 4.12: Structure of the Reasoning Process in CASEY](image)

The ACTIVATE-EXPLAIN-FOCUS abstract reasoning structure of CASEY (see figures 3.14 and 3.15). Activate involves activating input features and evidence-states, which in turn activate a subset of existing cases. Explain is a justification of similarity of features not involved in the initial retrieval process. Focusing involves modifying the solution of the best matching case, by modifying the solution's explanation so it fits the new case.
Two notions need to be introduced in order to describe the matching process: *evidence-states* and *general causal features*. An evidence-state is a physiological state in the causal model that is supported by one or more input features. In other words, an evidence state is a state for which there is evidence in the input case. This does not necessarily mean that a state will end up being a part of the causal explanation of the input features, just that it *may* do so. A general causal feature, on the other hand, has been justified as a state in the causal explanation of a patient. General causal features are stored with a case, and used as primary features in the matching process.

The retrieval step retrieves all cases which have general causal features that match evidence-states of the input case. In this way, the input case and the retrieved cases will have features that are evidence for the same states in the causal model. The primary matching criterion is, therefore, the matching of states that explain the features rather than a matching of the features themselves. The retrieved cases are ranged according to the number of matched states. If unable to choose a best match by this criterion, the number of common features is used as a second criterion.

The second step in case matching attempts to justify the similarity of features not explained by the initially matched states. The set of evidence principles guides this process by looking for alternate explanation paths, additional supporting evidence, or justification for ruling out insignificant evidence. If the justification does not succeed, the next best retrieved case is tried.

**Control of Multi-Paradigm Reasoning**

As already stated, case-based reasoning is the primary method. Model-based reasoning (as a self contained reasoning method) only come into play when the case-based method fails.

### 4.4.6. The Learning

CASEY always learns from a problem solving case:

If a problem is successfully solved by case-based reasoning, CASEY stores the new case if it has significant features different from the previous case. There is no learning from failed justification attempts. If the new case is identical to the previous one, information about the importance of a feature is updated.

If case-based reasoning fails, and the causal model solves the problem, a new case is created and indexed in the memory structure.
The learning in CASEY does not involve user interaction. The system is designed to improve performance efficiency of model-based reasoning from the Heart Failure model. Hence, CASEY learns associative knowledge only, by extending or modifying\(^1\) its case base.

**Structure of the Learning Process**
Figure 4.12 shows the **EXTRACT-CONSTRUCT-STORE** process structure for CASEY. CASEY primarily learns a new case from the differences between a new problem and the previous case that was used to solve the problem. After a problem is solved by using a previous case, a substructure of the case memory has been activated. This substructure contains the nodes in the discrimination network that - taken together - hold the indices of the previous case. In addition, as part of the matching process, the nodes corresponding to the input features, evidence-states, general causal features, and therapy for the new problem have also been activated. With the exception of evidence states\(^2\), this structure is given as input to the learning process. A new case is built, irrespective of whether the problem was solved by a previous case or by the causal model alone.

**Case Learning**
If a new case is created, it contains
- all input features of the problem
- all general causal features
- a causal explanation of the disease
- the recommended therapy

If the problem is solved without adapting a previous case, the explanation to be stored is the one produced by the causal model. The new case is indexed in the memory structure by its general causal features, by its input features, as well as by the therapy it recommends for the patient. If no relevant features are different, CASEY updates importance measures for the input features.

It should be noted that cases are indexed by every input feature, although general causal features are the primary indices. Indexing by all input features has the advantage of retaining features that may seem irrelevant at the time of case storage, but that may prove to be relevant later. On the other hand, an obvious disadvantage is larger memory requirements\(^3\) - depending on the number of features for each case and the variability between cases.

---
\(^1\)Modification of feature importances, measured by the number of times a feature is seen in a case, and the number of times it is used in a causal explanation.

\(^2\)This makes sense, since the problem's general causal features are a subset of its evidence-states, i.e. the subset that has been causally explained for the new problem.

\(^3\)The dynamic memory structure creates a new index (splits the discrimination tree) for every feature name and every feature value of a general node that differs between a new case and the existing cases indexed under the node.
General Knowledge Learning

There is no learning of general, conceptual domain knowledge in CASEY. Its domain is defined by the Heart Failure model, which - for learning purposes - is considered complete. As reported by Koton [Koton-89, page 70], CASEY can in principle find a causal explanation that is outside the Heart Failure model: It may find a solution by combining an insufficient causal explanation with unexplained features, thereby deriving a match with a previous case. This mechanism seems to be at the fringe of the system, and is not discussed elsewhere in [Koton-89].

CASEY learns generalised associations between input features and diagnoses, since the associations (i.e. cases) are indexed in the memory structure by placing their descriptors in nodes that are as general as possible\(^1\). The general nodes may be used to make predictions about new problems, e.g. by assuming the presence of node features that are unknown, or by constraining a solution to a disjunction of the solution of cases indexed under the general node.

\(^1\)That is, only as specific as required to discriminate from nodes or cases with different descriptors (see figure 3.11).
CASEY performs two types of generalisation: A kind of similarity-based generalisation is implicitly performed when unexplained features are stored in general nodes, while storing of general causal features in general nodes represents an explanation-based generalisation. The two types of features are stored in two different types of general nodes (nodes called FEATURE-GEN store all features, while CAUSAL-GEN NODES store general causal features only).

**The User’s Role**

in CASEY is none. All user interaction to improve overall problem solving quality is via manual modifications to the causal model.

4.4.7. Relating CASEY to the Critical Factors

Below, important properties of CASEY - as seen by the framework described in chapter 3 - are summarized by relating them to the critical factors defined in section 4.2. A "+" sign indicates an advantageous property, while a "÷" sign indicates a missing property or a disadvantage:

**CF1: Thoroughness of knowledge model**

- + Strong, tightly coupled causal model.
- ÷ Only causal relations.
- ÷ Fixed knowledge model, with respect to CASEY.
- ÷ No explicit problem solving or learning strategy.

**CF2: Active role of explanations**

- + Matching based on explained features as primary criterion.
- + Storing of explanation paths within cases.
- + Explanation-based adaption of a previous solution to fit a new problem.
- ÷ Simple explanation structure, no qualifiers or conditionals.
- ÷ Generalisation of unexplainable features by similarity-based method, instead of interacting with expert/user.
- ÷ Only diagnoses are explained, not therapies
- ÷ Explanations are not rated, the first one acceptable is used.

**CF3: Combined use of reasoning methods**

- + Model-based method supports matching of cases, and adaption of solution.
- + Separate model-based method to fall back on when case method fails.
- ÷ No problem solving or reasoning knowledge at the task/strategy level.
4.4.8. Relating CASEY to the Framework Requirements

A Summary of Important Characteristics

CASEY has two general properties that explain several of its more special characteristics. One is the use of a deep causal domain model for reasoning and learning if the case-based method fails; the other is that the only significant learning that takes place is improvement of performance efficiency\(^1\). In terms of our framework, the first property is particularly interesting, and places CASEY in the position it has got in our analysis. The second property limits the scope of CASEY, since the goal of learning is not problem solving improvement within a part of the real world, only efficiency performance within an existing computer model\(^2\). However, as noted by Koton [Koton-89, page 74], most of the methodological approaches taken in CASEY should be generalizable to a more open learning context, involving learning of new knowledge through interaction with a competent user.

Requirement 1: Expressive and Extendible Knowledge Representation Formalism

There are two different representational formalisms in CASEY, one for the Heart Failure model, and another for the case memory. The only relation for deep modelling in the Heart Failure model is causes\(^3\). The only moderator of the causal relation is a numeric probability - or likelihood - measure. This measure does not capture the underlying reasons for one cause being more plausible than another. Expressiveness is also limited by features and states being flat (property-name property-value) pairs, with no structuring of properties. The framework requires that all relevant relationships in a domain model are expressible, either directly or by explicitly defining a new relation when it is needed. Neither of these options are possible in CASEY.

The representation of cases, on the other hand, is done within a dynamic extendible structure of cases and collections of more general features. Although the memory structure is primarily an indexing network, a general node represents a description of some virtual "concept", defined by

---

\(^1\)This is what Dietterich calls symbol level learning ([Diettrich-86]), as opposed to knowledge level learning. Symbol level learning systems improve their performance without changing their knowledge content.

\(^2\)An advantage of this, of course, is that evaluation of the case-learning method is facilitated, since the correctness of a solution is precisely defined - as the solution arrived at by the Heart Failure program.

\(^3\)The other relation is a general association.
the fact that its features describe a set of distinct phenomena in the real world. This seems to be a suitable structure for storing cases, but it also has a couple of major weaknesses:

First, the structure is formed through syntactic matching of new problems with existing cases; existing knowledge is not used to guide the formation of nodes. Second, by establishing difference links between all values of all features, the number of nodes in the structure may grow exponentiallyจำ.

**Requirement 2: Combining Case-Based and Elaborative Reasoning**

The role of the general knowledge model as support for the reasoning from past cases, and as a separate reasoning method, satisfies - in principle - the framework’s requirement for combined reasoning. In particular, CASEY demonstrates how past solutions are effectively modified by use of a general knowledge model. However, the following three deficiencies of the reasoning process in CASEY violate basic ideas of our framework:

The major weakness is the lack of interaction with the user. A reasoning process should be able to come up with expectations and other consequences which it may want to check with the user. The user may also be able to supply additional information about the problem, which the reasoning process can use in case of difficult conflicts.

Second, there is no explicit strategy or task model of diagnosis and therapy. CASEY discriminates between features, physiological states, evidence-states, etc., but they are all defined in the code, not as a separate control level model.

Third, the reasoning about therapies seems very rudimentary. The diagnosis and therapy phases are regarded as two separate, sequential phases, not as interleaved processes. A therapy is stored within a case, and retrieved once a modified explanation for the disease is found. The relation between diseases and therapies are mere associations, with no causal explanation of why a certain treatment is successful.

**Requirement 3: Combining Case-Based and Explanation-Based Learning**

A strength of the learning method in CASEY is that a case is learned even if the problem was solved by pure model-based reasoning. Another interesting aspect is that explanations are learned in the sense that they are stored with the case, and indexed for future matching purposes. Further, CASEY combines explanation-based and similarity-based learning of cases.

---

1The growth rate depends upon the variability of cases. During work on the MEDIATOR system [Simpson-85], a system for the domain of dispute mediation where the same kind of case structure was used, the performance on a Symbolic computer was significantly reduced, by the memory structure holding 9 cases.
The major weakness of the learning process is the closed world that CASEY operates in. Only cases are learned, since the general domain knowledge is regarded as fixed.

A second weakness is the limitation on knowledge intensive learning imposed by the restricted domain model. For example, CASEY attempts to generalise a causal feature when learning an explanation, but the only generalisation possible is along causal links.

4.5. CHEF

4.5.1. General

Overview
CHEF [Hammond-86a, Hammond-86b, Hammond-87] is an AI planner that views planning as a process of retrieving a previously successful plan, and adapting this plan to the present problem. Although planning problems are different from classification problems like diagnosis, the CHEF system contains several properties of interest to knowledge-intensive case-based reasoning in general. The system will, therefore, be treated at a more abstract level than the previous two systems, focusing on the problem solving process, the learning process, and the role of explanations.

CHEF was the first well-known case-based system to incorporate model-based reasoning and explanation-based learning methods. CHEF’s plans are indexed in memory by the goals they achieve and the particular problems they avoid. Explanations from a causal domain model support the problem solving process of repairing a plan that has failed, and the learning process of finding blame for mistakes made by the planner.

CHEF’s problem domain is Chinese cooking: the system attempts to build a recipe describing how to cook a particular dish, based on a user’s request.

A typical problem solving session
The problem entered is to make a stir fry dish with chicken and snow peas (the example is adapted from [Hammond-86a], see figure 4.14). The first step CHEF takes is to anticipate any problems that may occur in creating a plan for this problem. CHEF is reminded of a previous plan for a stir fry beef and broccoli dish. In making that dish by stir frying the ingredients together, it turned out that the liquid produced by the beef made the broccoli soggy, and not crisp, as it should have been (figure 4.14b). CHEF then attempts to retrieve a previous plan that matches the input goals while avoiding this problem. It uses goals and avoided problems as
PART II - Framework and comparative analysis

Figure 4.14: Problem Solving and Learning in CHEF
The figure shows parts of a tracing, adapted from [Hammond-86a]. The upper box (a to d) shows how a new problem is solved by retrieving a past plan, finding that a particular failure must be avoided, and modifying the previous solution so the failure is avoided in the current plan. Since the only failure that may occur was stored with the previous plan, the new problem is successfully solved. Earlier, when the retrieved case was solved, the solution turned out to be unsuccessful. The lower box steps back in time and shows how the planner learned from this failure, by repairing the plan using repair strategies stored in Thematic Organization Packets (TOPs), and identifying the causes of the failure using the causal domain model.
index terms to retrieve a plan (figure 4.14c). A plan for beef and broccoli that copes with the anticipated problem is found, but it needs modification (figure 4.14d).

A modified plan is tested by simulating it within the causal domain model. For the current problem, the test is successful since CHEF was able to adapt a previous plan that avoided the anticipated problems.

If the simulation of a plan fails, CHEF attempts to repair the plan using a set of repair strategies. When planning for the beef and broccoli dish used in the example above, the problem of soggy broccoli was experienced during simulation of the plan. A repair strategy was retrieved to repair the plan (figure 4.14.e). The strategy addresses the problem of interaction between concurrent plans, where a side effect of one plan (stir frying of beef causes liquid to be produced) violates a precondition of the other (stir frying of broccoli requires a dry pan).

After a successful plan is produced, CHEF learns by identifying the reasons of its failure (figure 4.14f), and stores the plan.

4.5.2. Knowledge Model and Problem Solving Process

CHEF's object level domain knowledge consists of a causal model of domain concepts and relationships, and a structure of previous cases. The causal model is represented as a network of states and steps (i.e. actions) leading to states. A step has a set of results rules, expressing the states that are caused by the step. Each result has associated a set of conditions that must be satisfied for the rule to have the specified effect. A state also has a set of inference rules associated with it, specifying preconditions that must be satisfied for the state to trigger one or more steps.

The cases are stored in a dynamic memory structure similar to the structure used in CASEY [Schank-82]. Previous plans are held in a discrimination network, indexed by the goals they satisfy. Further, since an important part of a planning process is to avoid generation of action sequences that lead to a failure, a plan is also indexed by the particular problems it avoids. A solved case contains:

- features describing the planning problem
- features indicating problems that the plan has learned to avoid
- the successful plan

---

1A problem description for a planning problem is a set of goals to be satisfied, and the solution to a planning problem is a sequence of steps to take that will achieve the goals.
There are two types of control knowledge in CHEF. A previous plan that has been retrieved usually needs modification, which is controlled by a set of modification rules associated with particular goals. A modification rule is triggered if a retrieved plan is unable to satisfy a current goal. The effect of firing a modification rule is that some steps are added and/or deleted from the retrieved plan, in order to make it satisfy the goal. A concept referred to in a modification rule may have one or more object critics associated with it. An object critic specifies actions to take in order to avoid particular problems related to a concept, as shown in figure 4.14d.

The second type of control knowledge enables CHEF to suggest repairs to a plan, if a suggested plan has failed. A set of repair strategies is organized in the discrimination network by particular structures, called "Thematic Organization Packets" (TOPs, see [Schank-82]). A repair strategy is selected on the basis of the explanation of the failure that is generated from the causal model. Hence, repair strategies are indexed by features of these explanations.

Past plans as well as repair strategies are indexed by a pre-determined, fixed vocabulary.

CHEF consists of six modules, each representing a problem solving or learning subprocess. Within the UNDERSTAND-GENERATE-FOCUS model of the framework, the four problem solving modules are grouped in the following way:

1. UNDERSTAND:  • Problem anticipation.
   The planner tries to understand a planning situation by anticipating the problems that a plan generated from the input features may run into.

2. GENERATE:  • Problem retrieval.
   Search for a previous plan that satisfies as many goals as possible, while avoiding the problems anticipated.
   • Plan modification.
   Change the retrieved plan, so it satisfies all goals specified in the input. The modification rules are used in this process.

3. SELECT  • Plan repair
   This is the largest and most complex module of CHEF. After a retrieved plan has been modified, CHEF checks that the plan does not violate any goals. This is done by running a simulation of the plan within CHEF’s model of the world, which is the causal model. If the plan leads to a failure, the causal path leading to the failure constitutes an explanation.

\[\text{ Since the framework primarily addresses classification type problems, the model would need some revisions and extensions to fit planning problems. However, at an abstract level, the three-step model may be viewed as a general problem solving model. The SELECT step is the weakest part of the model, since plan 'selection' is generally a rather complex process of simulation, partial plan application, re-planning, etc.}\]
of why the plan failed. Index terms from this explanation chain are used to retrieve the appropriate repair strategy, which is then applied.

CHEF always attempts to retrieve a matching case and to modify its plan according to current goals, using the causal domain model as support. CHEF has no model-based or rule-based reasoning method to fall back on if the case matching process fails.

4.5.3. The Learning Process

CHEF learns by retaining successful plans. A successful plan is either a plan that was successful after Plan modification, or after Plan repair. After the Plan modification process is completed, CHEF has produced a plan that it assumes to be correct. If the plan turns out to fail, and need a repair, it means that CHEF’s understanding of the planning problem was faulty. To learn from this mistake, CHEF produces a causal explanation of the failure, so it can anticipate the failure next time it solves a similar problem. The explanation is generalized to the highest level of causal description that is permitted by the inference rules (for example, the failure caused by liquid being produced when stir frying beef is generalized to a failure caused by meat, figure 4.14f). Due to this method of learning a failure by explaining and generalizing it, CHEF is often included among explanation-based learning systems (as in [DeJong-88] and [Minton-89]).

CHEF has two learning modules. They may be described by the EXTRACT-CONSTRUCT-STORE model of our framework when the EXTRACT phase is viewed as part of the final steps of the problem solving process:

1. **EXTRACT:** Learning sources are results and activated structures from the Plan modification and Plan repair processes.

2. **CONSTRUCT:**
   - Credit assignment.
   This module uses the explanations generated during Plan repair to find out which input features are to be blamed for the failure. The features responsible for the failure will be retained in order to enable CHEF to predict the failure when a similar case is entered in the future.

3. **STORE:**
   - Plan Storage.
   A plan is stored by indexing it within the dynamic memory structure by the goals and failures that they predict. As described for CASEY, this structure is a discrimination network containing nodes with features common to several plans as well as nodes pointing to concrete cases. The latter are leaf nodes in the structure.
4.5.4. Relating CHEF to the Framework Requirements

Several of the decisions made when designing the system are particularly related to the domain of planning problems. The features of CHEF of interest here are those relevant for classification problem solving as well.

**Requirement 1: Expressive and Extendible Knowledge Representation**  
**Formalism**

The causal model is represented as a network of states and steps, where the relation between a state and the steps it may trigger, as well as the relation between a step and the effects it causes, have the form of inference rules. A rule is represented in a frame-like structure, with slots describing its variables, expressions for testing of variables and conditions, actions taken if the rule fires, etc. This gives CHEF a high degree of expressiveness and flexibility for representing cause-effect rules.

Another strength of CHEF is its modelling of strategic knowledge, i.e. its modification rules, and repair strategies. The way repair strategies are modelled and indexed, and the fact that they are retrieved by features from failed explanations, makes it a good example of knowledge-intensive strategic reasoning.

A major weakness of CHEF is its simple model of general domain knowledge. In most domains, a number of other relationships than pure causal are needed in order to build a thorough and deep domain model. It is hard to see that the domain of cooking is an exception.

**Requirement 2: Combining Case-Based and Elaborative Reasoning**

A strength of CHEF is the important role explanations play in repairing a plan that failed, and in learning from a mistake. Generalized domain knowledge is also contained within the object critics. In the critics, domain knowledge is expressed by action steps that should be taken when a particular ingredient is involved. The critics are used in connection with modification rules in adapting a retrieved case to a new problem.

A major weakness of CHEF is that the retrieval of a previous case is based on a superficial similarity of features, using a fixed vocabulary of concepts. The causal model is not used at all during case retrieval. Neither is it used in the adaption of a previous solution (Plan modification); this process is guided by general modification heuristics and the object critics. It seems that the Plan repair process would have been relieved of some of its burden, if the causal model had been used to justify a modified plan.
Requirement 3: Combining Case-Based and Explanation-Based Learning

The discrimination network model of case memory enables learning by retaining cases, and by creating generalised nodes containing index features common to a substructure of nodes and cases. The plan itself is stored as a concrete case, without any attempts to generalise it. A major strength of CHEF is that it learns from its failures by using explanations produced by the causal model to identify the steps that were responsible for the failure. When the plan is later retrieved, this information enables a similar planning problem to avoid the same mistake.

CHEF also learns object critics, which are repairs made to previous plans. Since critics are stored under the relevant concepts, and not within plan cases, this is also a kind of general knowledge learning.

There is no learning of causal knowledge - i.e. cause-effect relations - in CHEF, and no interaction with the user for other learning purposes (e.g. confirmation, justification). This is obviously a weakness when CHEF is viewed as a real world sustained learning system1. A further weakness is that the index features referring to planning goals are not learned by evaluating their importances and predictive strengths, but picked from a fixed vocabulary of indexing concepts.

4.6. JULIA

4.6.1. General

Overview

JULIA is a case-based reasoning system under development [Kolodner-87, Hinrichs-88, Hinrichs-89]. It implements a problem solving architecture integrating case-based and constraint-based reasoning, controlled by a goal scheduler that decomposes problems into subtasks. A problem specification is assumed to be under-constrained, and the constraint-based reasoner attempts to infer (or get from the user) additional information about the problem. JULIA’s domain type is planning, and Schank’s dynamic memory structure is used to store the cases. The nodes in the network are frames with slots and facets, allowing a complex object representation. JULIA is designed to be a reactive planner, i.e. it addresses planning in an open world context, and should respond dynamically to changes in its environment. Hence, each decision made by the system is justified by referring to its sources, enabling dependency-directed backtracking in a justification-based truth maintenance system (Doyle’s system is used [Doyle-79]).

1As for CASEY, however, the closed world view of CHEF may be an advantage for testing and evaluating particular methods.
JULIA’s problem domain is meal planning; the system attempts to compose a complete meal for a party of people. Figure 4.15 illustrates the main modules in JULIA’s architecture.

**Figure 4.15: JULIA’s Problem Solving Architecture**

The figure shows the four main modules. A goal scheduler controls the problem solving, the reasoning is a combination of constraint propagation and case-based reasoning, and a truth maintenance system handles the dependencies of inferred values. (Adapted from [Hinrichs-88]).

A typical problem solving session
JULIA is given the problem of planning a dinner for a research group. First, a problem frame is instantiated, and the goal scheduler reduces the problem to a set of tasks that must be achieved. The basic strategy is to constrain the problem enough to enable a matching case to be retrieved from the plan memory. The first task is to identify a sufficient set of descriptors for the meal. This is done by attempting to satisfy constraints associated with slots of meal-descriptor frames. This task ends by asking the user what cuisine he prefers, and filling the cuisine slot of the problem frame with the value Mexican. JULIA then proceeds to the next task, which is to select a main course. On the basis of information so far, chili is selected. Now the case based reasoner has enough information to retrieve a previous similar case, with chili as the main course. The previous meal turned out to be unsuccessful, since some guests did not eat hot food. JULIA did not think of that possibility when solving the previous case, because there were no constraints regarding hot spices at that time. JULIA learned from the failure, and will attempt to satisfy the constraint this time. This is done by asking the user whether some persons in the party do not like hot food. The user answers that 3 persons prefer mild spices, and adds this as a constraint to the current plan. This leads to rejection of Mexican food, backtracking through the steps leading to the question about spicy food, and reactivating the task of selecting a cuisine. Given the anticipated problems with Mexican food in this case, the user now suggests Italian.

---

1 An additional piece of information given is that the dining room can not seat all the guests (the example is described in more detail in [Kolodner-87]).
The same pattern of constraint satisfaction, prompting the user, and retrieving and using previous cases, is repeated until all subtasks (including the appetizer-task, salad-task and dessert-task) have been completed.

4.6.2. Knowledge Model and Problem Solving Process

All knowledge in JULIA is contained within the discrimination network that represents and indexes the memory of past plans. There is no separate model of general domain knowledge. Generalized knowledge is represented partly as general nodes in the discrimination network and partly as constraints associated with particular concepts\(^1\). Constraints are stored as facets under slots, enabling activation of constraints when a value is entered. Constraint types are defined in separate frames that contain generator functions to produce instances of constraints.

JULIA stores a case by breaking it up into pieces, and it indexes and stores each piece separately. A set of pointers link the pieces, and is used when a complete case is reconstructed. JULIA may use parts from several cases to solve a particular problem.

The problem solving process is controlled by a task tree set up by the goal scheduler. The core reasoning process is a repeated sequence of trying to satisfy constraints, retrieving a matching case, using the case (if it was successful) or adding a new constraint based on the failure in the previous case (if it was unsuccessful):

1. UNDERSTAND:  
   • Decompose into subtasks. The goal scheduler sets up a subtask tree.  
   • Constrain subtasks. Propagates constraints on slots and values of subtask frames. Asks user for more information if needed.

2. GENERATE:  
   • Retrieve previous case information.  
   • Use retrieved case information to solve problem or anticipate failure.  
   A failure in a previous similar case produces an additional constraint for the new problem.

3. SELECT:  
   • Suggest solution based on a previous case  
   • Discuss with user

\(^1\)Constraints may be viewed as a kind of inference rules, and constraint propagation as a kind of rule chaining.
4.6.3. The Learning Process

Learning has so far not been a particularly focused issue in JULIA, the literature emphasizes the problem solving architecture and the combined reasoning method. However, being a case-based reasoner, JULIA learns by retaining cases in its discrimination network structure. Both success-ful and unsuccessful plan cases are retained. As for all systems using Schank’s dynamic memory, a case description is generalized by storing its indices in as general a node as possible, while still discriminating between the cases:

1. EXTRACT: A learning source is the solved case, including the constraints added, and possible failures experienced while executing the suggested plan.

2. CONSTRUCT: Build a case consisting of a pre-defined set of slots supplied with constraints or features derived from constraints in the present plan.

3. STORE: Store the case by breaking it up into parts useful for later problem solving.

4.6.4. Relating JULIA to the Framework Requirements

The primary strength of JULIA is its comprehensive problem solving architecture, containing a top level goal scheduler as well as a bottom level truth maintenance system. Problem solving is the focus of the methodology, relatively little emphasis is put on the learning model.

Requirement 1: Expressive and Extendible Knowledge Representation Formalism

The modelling of knowledge as constraints, and the use of constraints to infer information and to control the user dialogue, seems to be a powerful way to operationalize and utilize domain knowledge. The frame representation of cases, constraints and tasks enables a high degree of expressiveness for these types of knowledge.

JULIA lacks an explicit conceptual model of its domain. Such a model would have provided an environment for semantic interpretations of constraints and matching of cases. The system’s understanding of a problem is based on syntactic match between problem descriptors and constraint terms. JULIA - in its present state of development - does not seem to meet the requirements for a knowledge-intensive approach to problem solving and sustained learning.
Requirement 2: Combining Case-Based and Elaborative Reasoning
A strength is JULIA’s dynamic use of case knowledge and general domain knowledge, operationalized in the form of constraints. Its fundamental weakness is its inability to reason within a deeper, conceptual model.

Requirement 3: Combining Case-Based and Explanation-Based Learning
An advantage in the learning strategy of JULIA is that the user is involved in the learning process. The system is, thus, able to learn new constraints by getting feedback from the user about success or failure of a suggested plan. The method for learning of cases, however, does not seem far developed in JULIA. In particular, there seems to be no support from general knowledge in selecting index features or in the generalization process.

4.7. Chapter Conclusion

The systems analysed in this chapter represent the state-of-the-art in methods for knowledge-intensive problem solving and learning. The analysis of the systems and methods has been performed by using the framework specified in chapter 3. In accordance with the framework’s requirements for knowledge-intensive sustained learning, focus has been of the systems’ knowledge modelling aspects and their combined methods for reasoning and learning. Special attention has been paid to the role of general knowledge in producing explanations to support the case-based methods, as well as in problem solving using general knowledge only.

Each system discussed represents interesting contributions to the problems involved. Compared to the system requirements of the framework, however, their most significant deficiencies are:

1. All systems suffer from relatively weak models of general domain knowledge. None of the systems can be characterised as using a thorough, tightly linked knowledge model as a basis for its understanding of the domain. To develop such a model takes a lot of time and effort, of course, and should not be expected to have been undertaken by the research projects described. However, the analysis has shown that the systems described lack the representational expressiveness that is needed to develop such a model.

2. None of the systems combine separate reasoning paradigms within an open learning context (e.g. a learning apprentice context). The only system with multiple, self-contained reasoning methods - CASEY - has a closed-world view to its knowledge model. The two systems that are able to learn from interaction with the user - Protos and JULIA - are essentially case-based reasoning systems.
3. No system uses a top level, explicit strategy or task model (e.g. a model of diagnosis or planning) in their reasoning and learning.

The four systems that have been analysed all address the problem of combining several reasoning methods in their problem solving, and several learning methods in their approach to sustained learning. The systems focus on different aspects of this problem, leading to different strengths and weaknesses:

PROTOS emphasizes concept learning where a concept is defined extensionally as a collection of exemplars. It is able to represent a thorough and extensive body of general knowledge to support parts of the exemplar-based reasoning and learning processes. No other system has comparatively the same expressive power for representing a general, conceptual model. A final characteristic of Protos is its interaction with the user, making it a learning apprentice.

CASEY’s discriminating features are its thorough and extensive causal domain model, its use of this model for adapting a previous solution to a new problem, and - last but certainly not least - its ability to perform problem solving 'from scratch' if a matching case is not found. For CASEY, as well as Protos, the learning problem is particularly focused. CASEY’s learning differs from Protos’ in two ways: It is performed without any user interaction, and it learns generalized descriptions as features common to several cases.

CHEF is characterized by its case-based approach to planning, and the use of a causal domain model to support repair of a solution that fails, and to learn from the failure by explaining why a failure occurred followed by a generalization of this cause.

JULIA’s most notable features are its integrated case-based and constraints-based reasoning method. JULIA emphasizes problem solving rather than learning, and has the most comprehensive problem solving architecture of the systems discussed. Learning is taken care of by the dynamic memory structure and the associated methods for retaining and indexing cases. A particular property of JULIA is that a case is stored in separate pieces, and reconstructed - partly or totally - on demand. All the other systems store cases as separate entities.

In summary, the approaches discussed have shown that an integration of knowledge-intensive and case-based methods is a feasible and very promising approach to competent problem solving in weak theory domains, and to sustained learning. The systems reviewed have developed a set of particular techniques, addressing different aspects of this problem. So far, no work has been done to develop a system that integrates and further develops existing methods into a total system or architecture that satisfies the framework requirements.
In the following chapters, such a system is presented. The system - called CREEK - is a model of knowledge-intensive multi-paradigm reasoning and sustained learning, based on the requirements for such systems specified in the framework. The system focuses on a thorough and deep domain model as the basis for understanding, explicit control strategies, combining multiple reasoning methods, and knowledge-intensive case-based learning.
PART III

CREEK - AN ARCHITECTURE FOR KNOWLEDGE INTENSIVE PROBLEM SOLVING AND SUSTAINED LEARNING
Chapter 5

System Overview and Knowledge Structure

This chapter describes the building blocks of the Creek system architecture, and outlines the knowledge representation system. The next chapter describes the methods and basic algorithms for knowledge-intensive combined reasoning, and sustained learning, within Creek. The final chapter gives an example of the functionality of a Creek system, by showing how to describe an initial knowledge-model, how to solve problems and how to learn from experience. The application domain is fault-finding and repair during off-shore drilling operations, particularly related to properties of the drilling fluid (mud).

5.1. The CREEK Architecture

Creek is a knowledge intensive approach to problem solving and learning. That is, an approach which is based on an intensive use of knowledge of the domain and relevant parts of the surrounding world in its problem solving and learning methods. The basis for the structure and functionality of Creek is the framework described in chapter 3. The framework specifies a set of general requirements concerning knowledge modelling, problem solving and sustained learning. It also outlines a set of generic models of problem solving, reasoning and learning, through which existing methods may be described and analyzed (as was done in chapter 4), and new system architectures proposed (as is done in this chapter). Creek instantiates the framework by specifying an integrated system architecture for knowledge-intensive problem solving and sustained learning. The emphasis is on capturing as much as possible of principled knowledge and practical, concrete problem solving expertise within a domain, and to use an appropriate set of inference methods to operate on this knowledge.

The knowledge representation system in Creek is an open, extendible system. The knowledge representation language - named CreekL - is a frame-based language [Minsky-75, Bobrow-77,

---

1 This is different from an approach that focuses on a particular inference method, e.g. logical deduction, and where the expressiveness in the representation language is constrained to suit that particular inference method. These different views of knowledge and reasoning are discussed in [Lenat-87].
Greiner-80], where each term that has meaning within the system is explicitly defined. Each slot and named value symbol contained in a frame is defined in its own frame, since they are all regarded as meaningful concept types that are part of the total knowledge model. A knowledge base may be viewed as representing a tightly coupled network of concepts - i.e. physical entities, relations, processes, etc. - where each concept is defined by its relations with other concepts. A Creek knowledge base contains general domain knowledge, in the form of concept classes and relationships, and specific domain knowledge, in the form of concept instances and previously solved cases. The general knowledge consists of deeper, fundamental knowledge (e.g. a causal model or a combined component and functional model), as well as more shallow associations (e.g. heuristic rules). Within the Creek architecture, a simple heuristic association is represented merely as a particular relation, called implies\(^1\). More complex heuristic rules are represented as particular frames, called rule frames\(^2\). Thus, concept definitions, rules and cases are all represented as frames. A CreekL frame is a four-level structure which contains slots, which in turn contain facets, which contain value-expressions, which is a list containing value and value annotation fields\(^3\).

The inference methods operating on the semantic network of frames are typical frame language methods like property inheritance, frame matching (concept recognition), and constraint enforcement (constraint propagation). The basic reasoning approach in Creek is case-based reasoning from previously solved problem cases, supported by model-based reasoning within the knowledge network. In addition, heuristic rules may be used for associational reasoning when the case base is still small and covers few actual problems, or if case-based problem solving fails. Creek is designed for classification type problems, and incorporates a generic model of diagnosis and repair tasks, which is used to guide the actual problem solving process.

Since a knowledge base in Creek is supposed to be (or become) rather extensive - representing different types and aspects of knowledge - some way to direct the reasoning according to the current purpose and context is needed. This is achieved by a method called goal-focused spreading activation, in which only the part of the knowledge base that is semantically and pragmatically related to the current goal is activated.

The case based learning method in Creek is supported and guided by the network model of specific and general knowledge. Crucial steps in the learning process, such as what remindings to create for a case, have to be explained by the more general knowledge model.

\(^1\)This relation should be read as 'typically-implies', and not as its interpretation within mathematical logic.

\(^2\)The rule language, and corresponding inference method, in Creek is rather simple. The architecture primarily aims at combining specific case knowledge with general knowledge, and heuristic rules are just one type of general knowledge. However, the architecture allows for a more powerful rule-based reasoner to be added. A study of methods for combining case-based and rule-based reasoning was made by Christian Larssen in his Diploma Thesis [Larssen-90], where he used the Creek representation to implement an experimental system.

\(^3\)An overview of the representation language is given in Appendix 2.
The memory structure of past cases borrows several ideas from the Porter/Bareiss model (described in Appendix 1). Thus, Creek’s basic methods for indexing, storing and retrieval of cases have some similarities with Protos’ methods. Although the case representation and indexing methods are not identical Protos’, the major differences are in the higher level processes of selecting relevant features, case matching, and modification of solutions. These differences are due to the more expressive representation language and the combined reasoning approach of Creek, allowing a more knowledge-intensive support for the processes that operate on the case memory.

Compared to existing approaches, Creek suggests an improved architecture for integration of problem solving and knowledge-intensive sustained learning. Creek defines a knowledge representation and a set of problem solving and learning methods which enable the development of systems that meet the framework requirements defined in chapter 3.

5.1.1. Structural Architecture

The Creek architecture contains four modules, as depicted in figure 5.1 (the dark ellipses). Each module represents a particular sub-model of knowledge and associated reasoning methods. An

![Figure 5.1: The Knowledge Modules in Creek](image)

A body of conceptual knowledge forms the basic environment in which the object level and the three types of strategic level knowledge are modelled. The conceptual knowledge fundament also contains an explicit model of the system’s representational terms (not shown in the figure). In this way, one unified conceptual model tightly connects the four embedded modules into a coherent knowledge base.
underlying model of descriptive knowledge called the *conceptual knowledge fundament* (CKF) is shared by all the four modules. Upon this knowledge fundament, each of the four sub-models may express operational knowledge like heuristic associations (cases and rules), and procedures (e.g. strategies, action sequences).

In addition to the object-level concept descriptions (also considered part of the CKF), the object-level knowledge model (OKM) includes the previously solved cases and the object level rules. At the control level, the combined reasoning model (CRM), has knowledge about what type of reasoning method - and associated knowledge structure - to activate at a given point of time. The kind of concepts reasoned about by the CRM are, e.g., case-based-reasoning, model-based-reasoning, concept-network, rule, case, etc. The model of diagnosis and repair (DRM) represents the problem solving process by a set of actions and their conditioned execution, e.g. when to apply a particular piece of domain knowledge, when to look for more evidence, when to hypothesize, etc. The sustained learning model (SLM) basically contains the algorithms (procedures) for knowledge-supported case based learning, and a collection of rules to guide the matching and feature generalization processes.

The conceptual knowledge fundament glues the four system modules together by being a common descriptive pool for all the terms and relations between terms used in the four models. Thus, diagnosis task concepts (e.g. symptom, diagnostic-hypothesis) as well as learning task concepts (e.g. case-index, failure-generalization), are defined within the same representation language as domain concepts (e.g. car, weak-battery, has-color). The conceptual knowledge fundament also contains an explicit model of the Creek system’s own representational primitives. This model describes the internal organisation and structuring of representational terms, like the type of frames (e.g. case-frame, rule-frame), slots (e.g. inheritable-slot, bookkeeping-slot), and facets (e.g. value-facet, constraint-facet) that the system contains.

The CKF may - and indeed should - also contain knowledge about bordering domains and parts of the general world. The desired level of robustness should always be kept in mind.

In figure 5.2 the CKF and the four specific models are pictured within the expertise model of the framework described in chapter 3. Note that the Object Knowledge Model is put under the subclass *definitional knowledge model*. Thus, definitional knowledge subsumes descriptive knowledge (CKF) as well as operational object-level knowledge. Possible knowledge forms, i.e. rules and cases, are used to show subclass models of the OKM, while this level of detail has not been used for the control modules.

---

1 This backbone structure is represented in a separate system knowledge base, as described in section 5.2.7.
Knowledge intensive reasoning and learning at the object level requires as complete a domain model as possible, while reasoning at the control level needs to know about the level, depth, role and form of the object knowledge structure in order to apply appropriate inference and learning methods to different knowledge types. Since a Creek system will contain several types of reasoning and learning methods, and different types of knowledge, each reasoning type needs to be associated with its appropriate knowledge types. The knowledge dimensions described in section 3.4.1.1 (and illustrated in figure 3.2) are explicitly represented within the system (part of the CKF), and available to the control modules for accessing particular knowledge types.

Figure 5.2: The Creek Expertise Model
The figure illustrates the type of expertise constituted by each of the four modules and the conceptual knowledge fundament in which they are embedded (cf. figure 3.4).

5.1.2. Functional Architecture

Creek integrates problem solving and learning into one architecture. The flow of control and information between the knowledge base and the processes of problem solving and learning is
shown in figure 5.3. The figure illustrates that problem solving in Creek is performed by a combination of model-based, case-based and rule-based reasoning (MBR, CBR and RBR, respectively). The learning combines case-based (CBL) and explanation-based (EBL) methods.

![Creek's Functional Architecture](image)

**Figure 5.3: Creek’s Functional Architecture**

At a high level of abstraction, Creek is an integrated architecture containing three building blocks: An object level knowledge base (representing the OKM), a problem solver (incorporating the CRM and DRM,) and a learner (containing the SLM). The object knowledge base contains a conceptual knowledge model, a collection of past cases, and a set of heuristic rules. Creek contains multiple methods for reasoning and learning, where each method communicates with a part of the knowledge base. (Abbreviations: RBR = Rule-Based Reasoning, MBR = Model-Based Reasoning, CBR = Case-Based Reasoning, CBL = Case-Based Learning, EBL = Explanation-Based Learning.)

The diagnosis and repair task controller and the combined reasoning controller (figure 5.3) uses knowledge in the DRM and CRM modules, respectively (figure 5.2). For example, the DRM initializes the problem solving process through the start-up task 'Understand problem', which activates a set of relevant features. This is followed by the task 'Generate candidate solutions' (the Diagnosis and Repair Model is described in connection with problem solving, subchapters
6.1.2 and 6.1.3). This triggers the CRM, whose initial task is to decide whether the case base or heuristic rules should be used in the initial attempt to solve the problem. When presented with a new problem, the reasoning controller receives a problem frame containing the input features and the current goal. In a system for diagnosis of car troubles, for example, the initial goal could be find-starting-fault. If a feature is known to the system, it will have a frame that describes its current properties (e.g. parent class, constraints, relations, associated faults, cases reminded of). If an unknown feature is presented, the system asks for a description of the feature, and creates the corresponding frames. The system checks whether this description violates any constraints or current facts. The input features (observed features - also called findings) trigger a chain of activations - along a particular subset of relations - in the knowledge model. By this process, a context for further inferencing is established. This process is guided by the current goal and subgoals and the current state of the Diagnosis and Repair Model. The subgoals are continually updated by the DRM as the problem solving process proceeds.

The process of selecting the initial reasoning paradigm starts when a set of relevant features of a problem has been identified. This feature set typically contains input features as well as inferred features, i.e. features that the system derives from the input features by using its knowledge. For example, when given the feature

\[
\text{engine-temperature 95 °C}
\]

the system may infer

\[
\text{engine-temperature very-high}
\]

Other examples of inferred features are synonym features and features that always co-occur with an input feature. Inferring of features extends or transforms part of the feature set. There is usually also a need to exclude features that are irrelevant, spurious and noisy. For a feature to be relevant it has to be justified by the knowledge model, by finding the feature’s relations with the current goal or sub-goals.

If the set of relevant features gives a reminding to a previous case that is above a particular strength - called the reminding threshold - case based problem solving is tried, otherwise activation of the rule set is attempted. Relevant features may be input features or features inferred from the object domain model. If either the case base or the rule base fails to produce a result, the controller re-evaluates its previous decision, given the current state of the system.

The value of the reminding threshold depends on the relative contents and power of the case based and rule based subparts. The initial threshold value is set manually, and adjusted over

---

1 The method that is used to achieve this - called goal-focused spreading activation - is described in section 6.3.2.
2 This is the basic picture; however, a method for trying to improve a match that is slightly below the reminding threshold by further elaboration within the knowledge model is also available.
time according to how well the chosen reasoning paradigm performs. Changing the threshold value may be done manually by the user, or automatically - by a simple procedure that rewards a reasoning type that has had success, and punishes one that tends to fail more than it succeeds. For example, if an application system has a well developed collection of rules, but has not seen many problems yet, the user may want to optimize performance by setting the threshold level to a high value - close to 1.0. The rule-base will then be used in the initial attempt to solve the problem, unless there is a very strong reminding to a case. As more cases are added, the ratio of case-based to rule-based successes is likely to increase, and the system will gradually lower the threshold value. The exact regime for this process will depend on the actual application system.

The sustained learning control module (figure 5.3) is the operational part of the SLM (figure 5.2). The learning controller guides the learning process by providing a strategy for when to perform particular learning tasks, and how to combine tasks. Creek learns from every problem solving experience. If a successful solution was copied from a previous case, the reminding to that case from each relevant feature is strengthened. In this situation, no new case is stored, but an attempt is made to merge the two cases by generalizing their feature-values.

If a solution was derived by modifying a previous solution, a new case is stored and difference links between the two cases are established. A new case is also created after a problem has been solved from rules or the deeper knowledge model. Thus, the main target for the learning process in Creek is the case base. But Creek may also learn general knowledge through interaction with the user during problem solving. There is no inductive learning of explicit, generalized concept definitions or rules in Creek. The only explicit generalization performed is the 'lazy' generalization of cases mentioned in the above paragraph.

Heuristic rules are integrated within the conceptual model, and available for the same tasks as the conceptual domain model in general. A rule may be used to support learning, for example. Even if the explanatory strength of a shallow relation like a rule in general is low, it may add to other explanations for the same hypothesis and, thus, contribute to a justification of an hypothesis.

With reference to figure 3.7, Integrated Four-Layered Expertise Model, the learning controller specifies the strategic and task knowledge of the learning-specific model. The reasoning controller is contained within the strategic and task layers of the problem solving model (i.e,

---

1A threshold value of 1.0 means that rule-based reasoning will always be tried first, while a value of 0.0 always leads to an initial case-based attempt.
2Generalizations are implicitly learned, however: Concept classification by explanation-based matching of cases implies a common generalization of the features - or feature sets - that participate in the match (see Appendix 1).
3The term 'lazy generalization' is sometimes used to characterize a learning method different from the more 'eager' generalization that is characteristic of classical, knowledge-poor inductive learning.
Problem solving specific knowledge). The object layer of the four-layered model is represented by the Object Knowledge Base of figure 5.1, while the inference layer holds the inference methods of Creek's knowledge representation system, described in the following section.

5.2. Knowledge Representation

Creek’s combined approach to problem solving and learning requires a representation system that is able to express conceptual models, heuristic rules, and previously experienced cases. The representation must be able adequately to express deep as well as shallow knowledge, at the object level as well as at the control levels. The framework of chapter 3 emphasizes the mutual support of various knowledge types in problem solving and learning. A requirement for such a synergy effect is that each piece of descriptive knowledge is understood in the same way by different reasoning methods. This is facilitated by using a common, unified representation scheme to express all types and forms of knowledge.

5.2.1. Representing Different Knowledge Types in Creek

Figure 5.4 illustrates the different types of knowledge at the object and control levels. Domain knowledge at the object level consists of a conceptual domain model in which specific experiences (past cases) and general heuristics (premise-conclusion rules) are integrated. Explicit knowledge about how to use domain knowledge for problem solving and learning is described partly as control level concepts, partly as control rules. The general definitions of concepts and relations at this level is part of the unified conceptual knowledge fundament (CKF). This enables an explicit definition of each type of concept that will be reasoned about. A function, for example, may be explicitly defined by describing its input arguments, output, side effects, contextual constraints, dependencies of other functions, etc. In this way, the system is able to understand the effect and operation of a function, and use this understanding in its reasoning at a higher level (control level).

1In general, whether all types of knowledge should be represented within a single, uniform representation language, or within a set of specialized languages for various types of knowledge, is still a question of debate in AI. Our knowledge-intensive approach relies on a common understanding of all concepts shared among different types of reasoning and learning methods. The CKF provides a common consensus between the more specific knowledge models. This is enabled in Creek by adopting a single, uniform representation language, while allowing for specific interpreters to attach to specific knowledge types. See also section 3.4.3.
Figure 5.4: Knowledge Representation at the Different Levels

The figure shows the four levels at which knowledge is described within the framework. The conceptual model at each level is a partial model within the fundamental knowledge model, the CKF. In the figure, the CKF is distributed among the four knowledge models. Upon this conceptual knowledge core, heuristics (conclusive rules and conditioned procedures) are used to represent the operational parts of the control knowledge. At the object level, past cases and conclusive rules constitute the operational knowledge. The case knowledge will have increased importance over time, due to the case-based learning method.
There are two types of heuristic rules within the Creek architecture, the premise-conclusion rule - called *conclusive rule* - and the premise-action rule - called *conditioned procedure*. While a conclusive rule expresses a *premise -> conclusion* statement, a conditioned procedure represents a *premise -> action sequence* structure. The conclusion part of a conclusive rule contains an assignment of a value to some variable (a frame-slot pair), which is a different type of knowledge than a specification of a sequence of steps to follow in order to reach a (sub)goal.

Thus, although they both are regarded as heuristic rules, conclusive rules and conditioned procedures are different types of knowledge, used for different purposes. Both types of heuristic rules are used at the control level, while - as noted above - the type of rule at the object level is the conclusive rule.

*Cases* in Creek are used to represent *object level knowledge* only. The idea of 'strategy cases' for control level reasoning and learning is not addressed in this research, but would be an interesting extension to study within the Creek architecture.

In order to capture this diversity of knowledge types, an expressive and flexible representation formalism is needed. A purpose of the representation system is to achieve a common consensus between the user/expert and the system as to the meaning of the actual knowledge that is represented. Hence, the semantics of the representation language has to be clearly defined. Flexibility - in the sense of an extendible representation language - and a clearly defined semantics may easily be viewed as conflicting requirements. An approach to solving the conflict is to choose a representation system where the semantics of added representational constructs become explicitly defined within the representation system itself. For example, if a new interpreter (an uncertainty handler, say) is needed to represent a new type of knowledge, an extendible representation system should allow for such an interpreter to be included in the representation system. In order to preserve a clear semantics of the representation language, the properties of the new interpreter should be explicitly defined within the system.

Several formalisms was considered as candidates for a representing the various types of knowledge in Creek. A representational notation based on frames was assessed to give the highest degree of flexibility, expressiveness and user transparency, while avoiding some limitations imposed by representation formalisms like production rules or Horn clause logic.

Frame systems may also be stiff and inflexible, however, depending on how open - or closed - their design is. Frame systems marketed as expert systems shells, for example, are often very restricted in the way a knowledge base designer may model the knowledge. The frame systems

---

1 A case-based system for strategic reasoning and learning, called BOLERO, is described in [López-90].
2 Framework requirement R1.
of tools such as KEE, NEXPERT OBJECT, GoldWorks, etc. are examples of restricted frame formalisms. The type of frame system needed for Creek must allow for definition of any type of relation, and provide a basic, general inference mechanism upon which higher level inference methods may be defined. A particular interpreter - i.e. the procedure that implements a particular inference method (like constraint propagation) - should be defined in a control-level frame by the conditions that have to be satisfied in order for the inference procedure to apply, the type of results it produces, its dependencies on lower level inference methods (like, e.g., default inheritance, backward chaining), etc. This is the approach taken in open frame systems like the RLL system [Greiner-80], KODIAK [Wilensky-86], KRS [VanMarcke-88], and CYCL [Lenat-86, Lenat-89]. It is also the approach taken in Creek’s representation system. The frame representation system in Creek has been developed as part of the research reported here, based on the frame system of METAKREK (Sølvberg-88, Aakvik-88).

It should be noted that an open frame system formalism may have the disadvantage of being too flexible. That is, the power of built-in, well-defined inference methods that more restricted formalisms possess, is sacrificed for flexibility. Awareness of this problem, however, has guided the design strategy for the representation system in Creek. Ideally, an open frame system should have explicitly defined definitions of all methods, to preserve a clear semantics. However, since this is a very time consuming effort - compared to building some of the methods into the code - some compromises will have to be taken when implementing (instantiating) the representation formalism for a particular application. This does not have to be catastrophic, as long as the semantics of the non-explicit methods are clearly defined and comprehensible to the knowledge engineer/expert team. However, the loss of flexibility in control level reasoning that results from implicit knowledge interpreters (i.e. interpreters defined in program code only), should always be kept in mind.

Knowledge representation in Creek constitutes a tightly coupled, semantic network of frames. As previously emphasized, every entity and every relation which has meaning within the knowledge model is explicitly defined in a separate frame. Relations relate concepts to each other, whereby the semantic network is formed. The meaning of each concept is defined through the concept's relationships with other concepts. This bottoms out on a set of primitives defined by Lisp functions.

---

1The major restriction of these systems is that they are built up around a very limited, fixed set of relations. None of them allows a developer to define his own set of relations, with particular inheritance methods, contextual dependencies, etc. Neither is modifications of existing relations allowed.

2METAKREK is a methodology and toolkit for interactive knowledge acquisition, developed at ELAB-RUNIT.
5.2.2. Creek’s Model of Representational Primitives

The representational primitives in Creek, shown in figure 5.5, are a specialization of the framework’s model (figure 3.11). The structure of knowledge in Creek may - at the knowledge level - be expressed in the following way (an enclosure in curly brackets indicates one or more occurrences and a vertical bar indicates optional choices):

\[
\begin{align*}
\text{knowledge} & : = \{\text{relationship}\} \\
\text{relationship} & : = \text{concept} \ | \ \text{relation} \ | \ \text{value} \\
\text{concept} & : = \text{entity} \ | \ \text{relation} \\
\text{entity} & : = \text{entity-name} \ | \ \{\text{relation} \ | \ \text{concept}\} \\
\text{relation} & : = \text{relation-name} \ | \ \{\text{relation} \ | \ \text{concept}\} \\
\text{value} & : = \{\text{concept} \ | \ \text{relationship} \ | \ \text{number} \ | \ \text{string} \ | \ \text{special function}\}
\end{align*}
\]

Knowledge is viewed as a collection of relationships. A relationship is an ordered triple of a concept, a relation, and a value. A concept is either of entity type (corresponding to physical or abstract entities in the real world) or relation type. A relation type concept explicitly defines the relation within the knowledge model. A relation is a link between a concept and its value and, hence, links two concepts into a relationship. Both an entity and a relation is defined as a structure containing its name, its set of 'attached' relations, and the set of values (mostly concepts) pointed to by each relation. Values of a relationship may be explicit concepts or other relationships. Values that are not defined within the knowledge model may also occur, for instance numeric values, strings of text, or subroutines (functions) in a programming language.

A relation in Creek is always regarded as a verb, and will always have a verbal form. An entity is regarded as a noun.

A simplified example:

```
vehicle           car
has-subclass      subclass-of
has-function      has-function
transportation    transportation-of
                  function-of
transporation-of-people

has-function      used-to-describe
                  has-inverse
                  instance-of
```

\footnote{In frame systems, binary relationships are generally assumed.}

\footnote{Numbers, symbol strings, and special functions may be regarded as concepts as well. However, this requires that the number or string has meaning, i.e. that it is explicitly defined in a way that makes its knowledge content interpretable and usable for the intelligent agent. In Creek, this is generally not the case. If the content of a string or number is to be represented, it will be as a concept or conceptual structure within the domain model, not as a string expression, numeral, or Lisp function.}

\footnote{An exception to the verbal form is that relations ending in "-of", like "is-instance-of" for simplification reasons are named without the "is-". There is no danger of confusing the relational form with a noun in this case, as opposed to deleting "has-" in, e.g., "has-color".}
The four concepts in the example are identified - labelled - by their entity-names (vehicle, car, transportation-of-people) or relation-name (has-function), and defined by their sets of

![Diagram of Creek Architecture](image)

**Figure 5.5: Representational Primitives in Creek**

The representational concepts and structure of Creek is based on the framework's model of representational terms (figure 3.11). The figure does not show all primitives at the most specific level - i.e. the instance level.

relation-concept pairs. The four concepts are structured into objects, where each object contains a set of relations, and each of these relations points to a set of values. The two relationships that
characterize the vehicle concept are, therefore, vehicle has-subclass car and vehicle has-function transportation. Note the relation used-to-describe in the has-function frame. This is not a type of book-keeping relation, but a semantic relation that describes for which entity classes (vehicle, car) the relation has-function makes sense.

Although not shown in the simplified example above, values are typed according to their semantic roles, called facets. A facet may state that a value is, e.g., 'the actual value' of the relationship, a default value, or a value class. While a relation characterizes the link between a concept and its value, a facet characterizes the link between a concept-relation pair and its value. In the examples above, an 'actual value' facet was generally assumed. A concept described with facets is shown below (the facet name value denotes an actual, current value).

```
van
  subclass-of value car
  has-color value-class color
  has-number-of-wheels default 4
  has-age value-dimension years
    if-needed (time-difference
                 current-year self.has-production-year)

my-car
  instance-of value van, private-car
  has-make value Ford
  has-color value green
  has-production-year value 1982
  has-number-of-wheels value 6
```

The facet of a value tells a reasoning agent how a value should be interpreted, and used. The value-class facet, for example, is a constraint on the actual values allowed in the relationship. It may be used as a check on added values, and as a type of knowledge in the reasoning process (even if a specific value is unknown for the relationship, it may help to know the value's superclass).

As shown in the van frame, a dot notation is used to specify a path from a frame name to a particular slot and facet value. For example, my-car.has-color.value.green denotes the fact that my car is green. If the facet part is left out, the value facet is assumed (e.g. my-car.has-color.green is identical to my-car.has-color.value.green). If both the facet and value parts of a dotted expression is left out, the expression refers to the actual value (i.e. the contents of the value facet). This is exemplified in the has-age slot of van, where the expression self.has-production-year refers to the 'actual value' of the has-production-year slot. (The frame name self is a reserved name used as a pointer to the frame within which the expression is contained, i.e. the van frame in this case).

So far, the description of the epistemic structure of Creek's representation system has - to a large extent - been kept at the knowledge level, i.e. at the level of knowledge content and structure independent of any underlying computational realization. Thus, terms like concept,
entity, relation, role, etc. have been the terms most frequently used. However, parts of the description have also referred to computational constructs such as frame, slot, facet, etc., which are symbol level - rather than knowledge level - terms. The shift of levels has been made in order to link Creek’s epistemological model (at the knowledge level) to its design model (at the symbol level). The remaining sections of this chapter consist mainly of a symbol level description of the representation language and the basic inference methods associated with it.

5.2.3. The Representation Language

In the Creek representation system - called CreekL - concepts are viewed as nodes in a semantic network where each node (i.e. each concept) is defined in a separate frame. Each frame is described by a set of slots. Slots represent the properties of a concept, i.e. its relationships with other concepts. The frames form a structural hierarchy implied by the set of structural relations that connect the frames. Four types of structural relations are predefined: Subclass-of, instance-of, part-of and member-of. They define a parent-child relation between general classes (e.g. car subclass-of vehicle), an individual exemplar of a class (car54 instance-of car), a component description of an object (wheel part-of car) and set membership (my-car member-of norwegian-cars), respectively. A slot expresses the <relation value> pair of a relationship, and facets allow for typing of values according to the roles they have. A facet in the representation language represents a value type (role name) and its value contents. Typical facets are actual values, default values, value constraints and procedural attachments. Each value may have annotations attached to it that represent a justification for the value, a value source, or the point of time when the value was entered. A value with its annotations list is called a value-expression. The term structure of CreekL is as follows:

```
knowledge-base ::=
frame ::= framename {slot}
slot ::= slotname {facet}
facet ::= facetname {value-expression}
value-expression ::= proper-value value-justification value-source value-timestamp value-miscellaneous
value ::= {frame | number | string | lisp-function}
value-justification ::= value
value-source ::= value
value-timestamp ::= number
value-miscellaneous ::= value
```

1The relations isa and ako are deliberately avoided, due to their ambiguous interpretations [Brachman-83b, Kuczora-89].
2The term ‘value’ is clearly ambiguous. As used here, it refers to the value of a particular facet of a particular slot. In cases of possible confusion, for example when talking about the ‘value’ facet of a slot or the ‘values’ of an annotation field, the actual value will be referred to as the proper value.
The content of a value justification field may be, e.g., confirmed-by user, by-strict-definition, or an explanation chain - i.e. a sequence of chained relationships. The source of a value is either a user name or a part of the system from which the value was generated or derived (just 'system' in the general case). For dynamic frames, i.e. frame instantiations manipulated by reasoning or learning processes, the value source may also contain the concept frame from which a value has been inherited. The field denoting miscellaneous annotations may hold particular information preceded by a key. A non-default qualifier for a relationship, for example, is represented in this field (under the key 'qualifier', followed by one of the qualifiers always or sometimes, as described in section 5.2.5.1).

Within the permanent knowledge base, a value is stored in the most general frame, and not duplicated downwards in the inheritance hierarchy.

By calling the network of frames tightly coupled we refer to the fact that all terms are explicitly defined. Every relation is a concept defined in its own frame. Every slot (e.g. has-number-of-wheels) and every value concept (e.g. green) is defined by their relations with other concepts (i.e. by the slots and values in their concept frames).

Concepts are also typed according to knowledge form, i.e. whether they are described as a non-specific network object, a rule or a case. A heuristic rule is represented in a particular type of frame, with a premise slot and a conclusion - or action - slot. Contextual constraints for a rule may be specified in a separate slot. Rules are linked to the common knowledge fundament by its relations with premiss and conclusion concepts, contexts of applicability, etc. The integration of rules within the semantic network structure enables the system to understand the role and potential effects of its rules, e.g. whether it is a control or object level rule and what rules may help to achieve a particular goal.

5.2.4. Inference in Creek

A frame representation system in AI is not just a particular syntactic structure for representing what human beings may regard as knowledge. A knowledge representation system also carries with it a typical set of inference methods that operate on the knowledge structure, and enables the stored structures to be used for reasoning within a system. Inference methods define the basis for interpreters that enable syntactic data structures within a system to represent meaning within the system, i.e. to represent useful knowledge for a system’s problem solving efforts.

---

1A summary of the representation language is given in Appendix 2, section 2.1, including a complete list of facet types. Subsequent sections of Appendix 2 describe the most important knowledge manipulation functions, illustrated by examples.
5.2.4.1. Inference Methods

The notion of a frame system - as introduced by Minsky [Minsky-75], and further developed by a number of authors [e.g., Bobrow-77, Greiner-80, Lenat-86, Minsky-88] has some important characteristics, including:

- A general concept is described by its **typical properties**, which are inherited by more specialized concepts and instances. The inheritance is not absolute, however, and an inherited property may in general be overridden by a specific, local property (default inheritance).

- The basic inference method in a frame system is **property inheritance**. Default inheritance is the most characteristic form of property inheritance in frame representation systems. Property inheritance also subsumes more restricted inheritance methods, like forced inheritance and condition dependent inheritance. A frame system with default inheritance needs to be able to resolve conflicts due to multiple inheritance of mutually exclusive properties.

- A more complex inference method is **frame matching**. While property inheritance is an inference method for retrieving a property given a particular concept, frame matching is an inference method that retrieves a concept given a set of properties.

A third inference method of importance to frame systems - although not a characteristic method of frame systems only - is **constraint enforcement**. Constraints are specified as facets and used to check possible values, and to perform constraint-based reasoning steps. Some types of constraints - e.g. a value class - may be used to infer the superclass of a value if no specific value is inferrable. Constraints may be propagated along specified relations, and inherited and combined into more specific constraints, from which conclusions may be inferred, explanations generated, and focused questions presented to the user.

The last type of inference method in the Creek architecture is **rule chaining**, i.e. the method that links a conclusion from one heuristic rule to the premise of another rule, etc., in a chained way (forwards or backwards).

It is important to note that the kind of frame system referred to here, and adopted in Creek, reflects a **non-classical view of concept definition**. While a classical definition defines a concept as a set of necessary and sufficient properties, this type of frame system defines a concept in terms of a **prototype** - i.e. as a set of typical properties. Taking this stance is motivated by the
problems of defining naturally occurring phenomena in a classical way (chapter 1.6), and has lead to a choice of other representation and inference methods in CreekL than those based solely on logical deduction\(^1\). Thus, Creek's realization of a frame system is different from frame representations based on first order predicate logic, like the KL-ONE [Brachman-79] and KRYPTON [Brachman-85] systems, which do not have default inheritance. Frame systems have strong similarities with semantic networks, and research on properties of semantic network systems generally also applies to this type of frame system\(^2\).

In figure 5.6 the inference methods within Creek are structured into a subclass hierarchy.

The inference methods described above are based on the three basic inference types described in section 3.2. The primary inference type in CreekL is abduction: That is, a set of uncertain hypotheses are derived from input descriptors and justified by explanations that - to varying degrees - support them.

![Diagram of inference methods within Creek Architecture](image_url)

**Figure 5.6: Inference Methods within the Creek Architecture**

A subclass structure of inference methods within the Creek architecture. The four main classes of inferences are each split into a set of subclasses, as described in the text. The method of plausible inheritance is further described in section 5.2.4.2.

Deductive inference may be used to infer a result when the type of knowledge involved in the inference makes it suitable. For example, value constraints may be regarded as an absolute (certain) type of knowledge, subject to universal quantification rather than prototypicality. In this case, deductive inference is used to enforce and propagate constraints. Deductive inference is also applied to derive properties that are inherited according to the scheme of forced

---

\(^1\)The motivation for this is further elaborated in chapter 1.6 (relevance for learning), chapter 3.2 (related to basic inference types) and chapter 3.5.3 (related to explanations).

\(^2\)For example, in a recent article Shastri [Shastri-89] presents an analysis of the two inference methods property inheritance and frame matching (called recognition) in semantic networks, relevant to the frame representation system described here.
inheritance: If, for example, the justification of a value is by-strict-definition, the value represents part of a classical definition of a concept, i.e. a necessary and universally quantified property. This, in turn, implies forced inheritance of the property to all subclasses and instances of the concept, where overwriting of the inherited value is illegal.

Inductive inference is used in the learning of generalized features when two similar cases are merged. The induction method is explanation-based, since the system evaluates a generalization by explaining its plausibility, i.e. checking for contradictions and trying to find additional paths between each of the special features and the generalized feature. Induction, in Creek, thus has strong similarities with abduction (which is consistent with influential models of induction, as described in, e.g., [Holland-86, Thagaard-88]).

5.2.4.2. Plausible Inference

A plausible inference step derives information that seems reasonable, and therefore may be used in further inferencing. A plausible inference is uncertain, and its basic inference type is abduction or induction. Reasoning with uncertain knowledge and information is still a major research area in AI; the methods include propagation of certainty factors [Shortliffe-84] or Bayesian probabilities [Duda-79], fuzzy logic [Zadeh-65], explicit endorsements [Cohen-85], and methods based on explanation support [Schank-86c, Leake-88, Bareiss-88b].

The way uncertainty is treated in Creek is by evaluation of explanations. Explicit numeric assignments of certainty factors and probabilities have been criticized for being difficult to interpret and evaluate in terms of meaning content (e.g. [Cohen-85]). In Creek there is no explicit assignment of uncertainty factors or probabilities. All facts (i.e. relationships) are regarded uncertain (unless otherwise stated), and their certainty within a problem solving process is determined by the strength of supporting explanations. That is, the degree of belief in a proposition is determined by the strength of explanatory support of the proposition within the knowledge model. Propositions that are justified by the annotations by-strict-definition or confirmed-by-user are regarded certain. Otherwise, the degree of belief is expressed by a numeral representing the strength of a justifying explanation. Hence, a system’s degree of belief in a proposition is related to relevance within the current problem solving context. In generating explanations, standard default inheritance is used to infer new relationships that may become part of an explanation. In addition, relationships may also be inferred by what will be referred to as plausible inheritance. The method is based on work by Cohen et. al [Cohen-88].
Plausible inheritance enable inferences beyond the standard inheritance mechanisms, by explicitly specifying particular combinations of entity classes and relations which trigger particular inheritance actions. While the standard inheritance methods in Creek inherits along subclass relations only, plausible inheritance makes it possible for relation-value pairs to be inherited along other relations as well. For example: From the facts \(A \text{ has-part } B\) and \(A \text{ has-location } L\), one may infer: \(B \text{ has-location } L\). This is a highly plausible inference step, at least if \(A\) and \(B\) are physical objects. Another example is: \(A \text{ contained-in } B\) and \(A \text{ causes } E\) infers \(B \text{ causes } E\). This inference applies, e.g., to a particular drug (\(A\)) that causes a medical effect (\(E\)) and the tablet (\(B\)) containing the drug, which also causes the effect. However, it does not apply if \(A\) is the gas contained in a car (\(B\)), since gas causes an engine to fire, but a car does not cause an engine to fire. Therefore, this type of plausible inference corresponds to conditioned inheritance, valid only within particular contexts.

The examples above specify context dependent inheritance along the \textit{has-part} and \textit{contained-in} relations, respectively. In the first example, the \textit{has-location } \(L\) assignment is inherited from \(A\) to \(B\) via the \textit{has-part} relation, while in the second example the cause of an effect is inherited from \(A\) to \(B\) via the \textit{contained-in} relation, as illustrated below:

\[ \begin{array}{c}
A \downarrow \text{has-location} \quad L \\
\quad \downarrow \text{has-part} \\
B \quad \downarrow \text{has-location} \quad L
\end{array} \] \quad \begin{array}{c}
A \quad \downarrow \text{causes} \\
\quad \downarrow \text{contained-in} \\
B \quad \downarrow \text{causes} \\
\quad \downarrow \text{causes} \\
C
\end{array} \\
\]

Plausible inheritance is represented in Creek by storing, in the frame representing the inheritance relation, the relations that may get inherited. For example:

\[
\text{has-part}
\text{inheritance-relation-for value has-location}
\text{inheritance-condition value (has-location physical-object)}
\text{has-location}
\text{inherits-via value has-part}
\]

(The inheritance condition in the example says that in order to inherit \textit{has-location} via \textit{has-part}, the value to be inherited must be a physical object)

A value inherited in this manner is checked for contradictions and constraint conflicts, and accepted if none is found.
5.2.4.3. Inference Control

The term ‘inference’ is a general term, used at different levels of granularity. An example of this is the distinction made here between inference types and inference methods (see, e.g., the last paragraph of section 3.2). In the four-layered model described in section 3.4.2.1 (adapted from KADS) the Inference layer has a broad interpretation, as illustrated in figure 5.7.

The upper three layers of the model (strategic, task, and inference layers) are control layers. Corresponding to the integrated four-layered expertise model of the framework (shown in figure 3.7), the problem solving and learning tasks have their own set of control layers, of which some knowledge may be shared. When the Four-layered model is related to the Creek architecture, the two upper levels - i.e. the strategic and task levels - should be viewed as contained within the Diagnosis and Repair Model of Creek\(^1\). The Combined Reasoning Mode

---

**Figure 5.7: Inference Layer Control in Creek**

The figure illustrates the three types of control in the layer of inference and reasoning - or learning method - of the four-layered model. Within this layer, the inference types (e.g. abduction) are the building blocks of the inference methods, which in turn are used by - and controlled by - the reasoning and learning methods.

and the Sustained Learning Model are at the Inference layer of their particular submodels (see figure 3.7). As shown in figure 5.7, the Inference layer contains both the inference methods - including their underlying inference types - and the reasoning and learning methods.

In figure 5.8 the internal control structure of the Inference layer is shown in more detail. The *reasoning method control* and the *learning method control* controls the selection and execution of the reasoning and learning methods shown within the two upper boxes. As illustrated by the dashed arrows each reasoning and learning method, in turn, has internal control procedures which handles the selection and use of inference methods within each

\(^1\)As previously described, this model is related to *problem solving*. In the current version of Creek, no corresponding model of strategies and tasks have been defined for the learning task.
reasoning and learning method. Each reasoning - and learning - method will typically use inference methods belonging to more than one of the inference method classes shown in figure 5.8.

The actual selection and execution of the proper inference methods during reasoning is controlled by the component called inference method control, which is triggered by the upper level processes of reasoning and learning control.

![Figure 5.8: Control of Reasoning and Inference Methods in Creek](image)

The figure shows the two sub-levels of methods and control that reside at the Inference layer of the four-layered model. The bold-lettered control procedures each control a set of methods. The reasoning and learning methods make use of an inference control procedure to draw the proper kind of inferences, given a particular reasoning method and type of knowledge.

### 5.2.5. Expressing Different Forms of Knowledge within the Representation Language

The following four sections describe how general concepts, cases, rules and explanations are represented in the CreekL language.

#### 5.2.5.1. The Representation of descriptive concepts and facts.

Descriptions of general concept classes and their instances are defined in frames named by the concept name and by a nested list of slots, facets, and value expressions - as described in sections 5.2.2. and 5.2.3. Facts - also called propositions, statements, or relationships - are thus represented by frame-slot-value triplets, where the value referred to is the 'value-proper' field of the 'value' facet for the frame-slot pair.
Within the CreekL language - and the Creek architecture - there is a substantial degree of
certainty, as to how a 'piece of knowledge' may be represented. It is up to the application
development team to choose the level of granularity and explicitness of representing the various
parts of the domain knowledge. In the car starting domain, for example, the fact that an engine
does not turn may be expressed by the relationship:

\[
\text{engine-1.has-status.does-not-turn}
\]

Another way is to specialize the status concept of 'not turning' (i.e. does-not-turn) into the
concept of an 'engine not turning':

\[
\begin{align*}
\text{engine-does-not-turn} & : \text{subclass-of} \text{ engine-status, finding, does-not-turn} \\
\text{status-of} & : \text{value-class} \text{ engine} \\
\text{caused-by} & : \text{value} \text{ elsysteem-fault ... qualifier sometimes, mechanical-fault ... qualifier sometimes}
\end{align*}
\]

where a value expression in the caused-by relationship is a simplified and easier to read notation
for the complete list expression: (elsystem-fault nil nil nil (qualifier sometimes)).

The status value of engine-1 may then be defined as:

\[
\text{engine-1.has-status.engine-does-not-turn}
\]

A fact may also be represented directly, in its own frame, by explicitly naming the fact and
defining it as an explicit concept (class or instance) within the knowledge model. This is
particularly relevant for important problem features such as certain observations and physical
states. Explicit definitions of such relationships makes it possible to describe a specific fact by
its relations to other concepts (e.g. other facts). For example:

\[
\begin{align*}
\text{engine-1-does-not-turn} & : \text{instance-of} \text{ engine-does-not-turn} \\
\text{status-of} & : \text{value} \text{ engine-1} \\
\text{last-reported-by} & : \text{value} \text{ John Doe} \\
\text{observed-at} & : \text{value} \text{ "02/10/89 03:50" "03/10/89 14:15"}
\end{align*}
\]

As previously noticed, all relations in Creek are explicitly defined in a relation type concept, for
example:

\[
\begin{align*}
\text{has-status} & : \text{instance-of} \text{ status-relation} \\
\text{used-to-describe} & : \text{value} \text{ engine, battery, carburettor} \\
\text{has-entry-type} & : \text{value} \text{ status}
\end{align*}
\]

The two latter relations only occur in concepts of relation type. The used-to-describe relationship
lists the concepts for which the target relation (has-status) makes sense, while the has-entry-type
relationship specifies the type of values in relationships where the target relation is involved.
Both these slots are important in defining the semantics of a relation, and they are treated as constraints when the relation is used in a relationship.

Values (proper values) in concept frames are generally regarded as *typical values*. For example, the relationships *A causes B* and *C subclass-of D* should be interpreted as *A typically causes B* and *C is typically a subclass of D*, respectively. However, a value may be explicitly *qualified* by a term which expresses how 'broadly' the corresponding relationship should be interpreted. There are three such qualifiers in Creek: *always*, *typically*, and *sometimes*. If other qualifiers than typically (which is the default) is used, it is represented by the key qualifier, followed by its name, and stored in the 'miscellaneous' field of the value expression. For example, the fact that a flat-battery always causes a starter motor not to turn, may be expressed by:

```
flat-battery causes value engine-does-not-turn ... qualifier always
```

If non-default qualifiers are frequently used, it should be considered whether a qualifier and a general relation had better be explicitly represented by a more specific relation. In the example above, a relation named *always-causes* could be defined, as a subclass of *causal-relation*. While the *causal-relation* class holds general properties of causal relations, the *always-causes* instance describes the properties of a causal relation which always holds. Explicit definitions of qualified relations will be used in the examples throughout the remaining of the report. That is, relation names not starting with 'always' or 'sometimes' should be interpreted as starting with 'typically'.

The miscellaneous field may also be used to express the *dependency* of a relationship on a certain condition or specific context. That is, a relationship may be valid only within a particular context, where the context may be a particular fault or a system state in general. In the medical domain, for example, a relationship such as *physical-work.always-causes.dyspnea* is valid within the context of a serious obstructive lung disease (airflow resistance), but not in the context of some other lung diseases: This may be expressed by the miscellaneous annotation key context:

```
physical-work always-causes dyspnea ... context obstructive-lung-disease
```

The context key states that the fault or physical state currently investigated must be a subclass or instance of the concept(s) listed (obstructive lung disease, in the example). This type of contexts may (and should) be explicitly defined in a submodel where the domain is partitioned into a set of contexts.
Sometimes it is necessary to express changes of feature values with respect to some reference value (for example a fixed standard value, or a previous value). This may be done by defining concepts such as, e.g., increased-air-flow-resistance. The rate of change (increase or decrease) may then be expressed by the miscellaneous key called :rate. For example:

```
case-001
    has-relevant-finding increased-air-flow-resistance ... rate strong
    increased-temperature ... rate weak
    decreased-blood-pressure
```

expressing a set of case features: A strongly increased air flow resistance, a weakly increased temperature and a moderately decreased blood pressure (the default rate - moderate - is assumed if there is no :rate key).

### 5.2.5.2. The Representation of Cases

A case in Creek is represented as a particular type of frame. A case is a dynamic structure which gets modified during the problem solving and learning process. It contains the following type of information, depending on its state in the problem solving and learning process:

<table>
<thead>
<tr>
<th>Input case</th>
<th>Case in process</th>
<th>Learned case</th>
</tr>
</thead>
<tbody>
<tr>
<td>input findings</td>
<td>input findings</td>
<td>relevant findings</td>
</tr>
<tr>
<td>goal</td>
<td>derived findings</td>
<td>successful diagnosis</td>
</tr>
<tr>
<td>solution constraints</td>
<td>diagnostic states</td>
<td>explanation of successful diag.</td>
</tr>
<tr>
<td></td>
<td>possible diagnosis</td>
<td>successful treatment</td>
</tr>
<tr>
<td></td>
<td>explanation of diagnosis</td>
<td>explanation of successful treatment</td>
</tr>
<tr>
<td></td>
<td>possible treatment</td>
<td>differential cases</td>
</tr>
<tr>
<td></td>
<td>explanation of treatment</td>
<td>failed diagnosis</td>
</tr>
<tr>
<td></td>
<td>similar cases</td>
<td>explanation of failed diagnosis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>failed treatment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>explanation of failed treatment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>side-effects of treatment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>book-keeping information</td>
</tr>
</tbody>
</table>

The snap shot of the *case in process* shown above assumes that the case has been built up after the entering of a new problem. If a suggested solution turns out to fail when applied to the problem, the system enters a second pass in the learning process (see subchapter 6.4): The learning from a failed solution. In this situation the input case is set to the previously learned
case, and the dynamic case will contain extensions and modifications to the input case related to explaining the failure and suggesting a new solution.

The different types of information contained in a case, as listed above, will be explained throughout this and the following chapter.

A case is indexed by three types of indices:

- **Relevant findings.** This is the subset of the input descriptors that has been explained to be relevant for solving the problem. If a numeric descriptor was transformed into a symbolic value, this becomes the value of the index feature.

- **Differential findings.** These are findings which may disqualify a case, since they point to cases that probably are better matches if such a finding is involved. Differential findings are inserted in a case reminded of when this case does not lead to a successful solution. Hence, they are indices between rather similar cases, although different enough to associate different solutions.

- **Faults and treatments:** Successful as well as failed solutions provide indices to the cases. The primary use of indices which are successful diagnoses is for the retrieval of cases in order to find proper treatments. An index which is a successful treatment is used to retrieve cases for checking of consequences and possible side-effects of applying the treatment. Indices which are failed solutions (faults as well as treatments) are used for case-based failure recovery and to avoid similar failures when solving new problems.

The structure of cases and general knowledge is illustrated in figure 5.9. Each case is a separate entity, and the collection of cases constitute what is referred to as the *case memory*. The cases are integrated into the model of general domain knowledge by having all its descriptors defined within the semantic network structure which constitutes the general knowledge. (This is illustrated by the thin lines (i.e. relations) between concept nodes and case boxes). The index pointers from findings, faults and repairs are illustrated by the dashed, thicker line, while the unbroken line between the cases represents a difference link. The cases pointed to by difference links are those that have only one relevant finding different from the present case.

Cases are stored in a particular type of frame, which inherits slots from the representational type called case (see figure 5.5). Individual cases may represent a single past case or a generalization (i.e. a merging) of several single cases. Two cases may form a generalized case if they have a feature name whose feature values have a common superclass. A feature is generalized by climbing instance-of or subclass-of links. During construction of a case for
permanent storage, the relevance of each finding to the solution of the case is assessed - if necessary by asking the user.

Figure 5.9: The Case Memory Model of Creek

The figure illustrates how cases (rounded boxes) are integrated into the general knowledge model (nodes and semantic relations). Findings, faults and treatments provide indices to the case memory, as well as difference links between cases.

The relevance is expressed by two measures, importance and predictive strength. The importance of a finding indicate how important the existence of the finding is for the classification of a particular case. Importance is rated as necessary, characteristic (the default), informative and irrelevant. The predictive strength of a finding expresses how strong a role the finding plays in predicting a particular problem solution, and is classified as sufficient, strongly-indicative (the default), indicative, and spurious. Non-default finding importances and predictive strengths are stored in the miscellaneous field of a value expression, under the keys importance and predictive-strength, respectively. A description of a relevant finding within a case may, for example, look as follows:

\[
\text{case-999 has-relevant-finding value } ((\text{starter-click-observed}) :\text{importance necessary} :\text{predictive-strength indicative})
\]
The importance and predictive strength is combined into a numeric *relevance factor*, as explained below.

A *relevant finding* is a finding which, at least, is characteristic and strongly indicative. A relevant finding is part of a meaningful problem description, and contributes to discriminate one particular solution from others. Importance and predictive strength is only assigned to relevant findings, not to other case features.

A simple example case - where the 'value' facet is assumed for all slots - is shown on the top of next page.

<table>
<thead>
<tr>
<th>instance-of</th>
<th>car-starting-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>has-process-status</td>
<td>fault-suggested</td>
</tr>
<tr>
<td>has-input-time</td>
<td>18/04/89 07:35</td>
</tr>
<tr>
<td>has-relevant-findings</td>
<td>ignition-key.has-position.starting-position</td>
</tr>
<tr>
<td></td>
<td>no-lights</td>
</tr>
<tr>
<td></td>
<td>battery.has-voltage.high</td>
</tr>
<tr>
<td></td>
<td>starter-click-observed</td>
</tr>
<tr>
<td></td>
<td>engine-sound-not-observed</td>
</tr>
<tr>
<td>has-fault</td>
<td>starter-solenoid-defect</td>
</tr>
<tr>
<td>has-fault-explanation</td>
<td>(....)1</td>
</tr>
<tr>
<td>has-similar-cases</td>
<td>case-29 case-152 case-204 case-58</td>
</tr>
<tr>
<td>fault-based-on</td>
<td>case-152</td>
</tr>
<tr>
<td>fault-modified-by</td>
<td>none</td>
</tr>
</tbody>
</table>

The example illustrates a case which represents a problem in the process of being solved. Since this is a case in process, no importance or predictive strength of findings have been determined yet. The case description shows that a diagnosis has been suggested, but no feedback on the suggested diagnosis has been given. Four cases similar to the problem at hand have been found, and one of them has a sufficiently strong match with the present problem to enable its solution to be reused.

Based on the importance and predictive strength of each finding for a particular fault or treatment, a numeric *relevance factor* is assigned to each finding - case pair. A table holds the relevance factor for each combination of importance and predictive strength. For example:

<table>
<thead>
<tr>
<th>Predictive strength</th>
<th>Importance</th>
<th>Relevance factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>sufficient</td>
<td>&lt;any&gt;</td>
<td>1.0</td>
</tr>
<tr>
<td>strongly-indicative</td>
<td>necessary</td>
<td>0.95</td>
</tr>
<tr>
<td>strongly-indicative</td>
<td>characteristic</td>
<td>0.90</td>
</tr>
<tr>
<td>strongly-indicative</td>
<td>informative</td>
<td>0.85</td>
</tr>
<tr>
<td>indicative</td>
<td>necessary</td>
<td>0.80</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>spurious</td>
<td>irrelevant</td>
<td>0.0</td>
</tr>
</tbody>
</table>

1The representation of explanations is described in chapter 5.2.5.4.
Note that the importance of a finding to a solution primarily tells how unlikely the solution is if the finding is absent. Hence, the major impact on the relevance factor comes from the predictive strength. The relevance factors are stored together with the case pointed to by a finding.

Example finding:

```
no-lights
  subclass-of elsystem-finding, visual-observation
  relevant-feature-in (case-19 0.8) (case-213 0.95) (case-466 0.85) (case-351 0.9)
```

Example fault:

```
starter-solenoid-defect
  subclass-of esystem-fault
  fault-in |case-48 case-152 case-99 case-351
```

The relevance factors are used in computing combined remindings from a set of findings to one or more cases (described in section 6.3.3).

### 5.2.5.3. The Representation of Heuristics

Simple heuristic associations are represented by the `implies` relation. More complex structures, i.e. heuristic rules or conditioned procedures, are represented as instances grouped into classes according to the following scheme:

```
<heuristic class>
  subclass-of value <heuristic class>
  has-subclass value <rule class> | <procedure class>

<rule class>
  subclass-of value <rule class> | <heuristic class>
  has-instance value <rule instance>

<procedure-class>
  subclass-of value <procedure-class> | <heuristic class>
  has-instance value <procedure instance>

<rule instance>
  instance-of value <rule class>
  has-antecedent value <lisp expression>
  has-consequent value <lisp expression>

<procedure instance>
  instance-of value <procedure class>
  has-antecedent value <lisp expression>
  has-consequent value <lisp expression>
```

There is no higher level language in Creek for specifying the actual rules or procedures. The most flexible language available is used: Lisp.
Value concepts referred to in heuristics have an antecedent-in or consequent-in slot, with a list of rules or procedures as values.

5.2.5.4. The Representation of Explanations

An explanation in Creek is a structure consisting of a single fact (i.e. proposition) or a chain of facts, as described for the framework, section 3.5.4.2. For general concepts, an explanation is attached to the particular value of a target concept that it explains. Since an explanation usually is associated with a single value, the explanation is - in general - contained in the value-justification field of the value expression. For example, if an explanation for why a starter motor (target concept) will not turn is that it has a weak battery, this will be represented as:

```
(starter-motor-1 instance-of value (starter-motor nil)
connected-to value (battery-1 confirmed-by-user)
has-battery-circuit-status value (closed confirmed-by-user)
has-turning-status expected-value (turning nil)
value (does-not-turn
(0.9
(battery-1.has-voltage.very-low)
(battery-1.instance-of.battery)
((battery.has-voltage.very-low)
causes
(starter-motor.has-turning-status
.does-not-turn)

(starter-motor.has-instance.
.starter-motor-1)

(starter-motor-1.has-turning-status
.does-not-turn)
)
)
```

The proper value and the value justification are shown for each value expression (nil denotes an empty field). The value of the expected-value facet is not justified in this example, but this could have been done, e.g. by the following explanation:

```
(1.0
(starter-motor-1.instance-of.starter-motor)
(starter-motor.has-battery-circuit-status.closed)
((battery.has-voltage.normal)
causes
(starter-motor.has-turning-status
.turning))
)
```

Particular types of explanations may also be stored in particular slots, as shown for cases in section 5.2.5.2.

The first element of an explanation list is a numeric value indicating the explanatory strength of the explanation. This is a measure for the degree of support a fact has in the underlying domain model, and hence the degree of belief in the fact (as described in section 5.2.4.2). The generation and evaluation of explanations are described in detail in the chapter 6.2.
5.2.6. The CreekL Implementation

The representation system has been implemented in CommonLisp on a TI Explorer, as described in Appendix 2. The root concept in a Creek knowledge base is thing. A Creek thing is either an internal system thing or a problem domain thing. This corresponds to two subconcepts of thing, namely Internal-Structure-and-Organization-thing and Problem-and-Object-Domain-thing, abbreviated iso-thing and pod-thing, respectively. The concept iso-thing is specialized into the various types of representational terms described in section 5.2.2, while the concept pod-thing is the 'root' concept of the domain knowledge model, and therefore needs to be specialized by the knowledge engineering team. The system knowledge base where this structure is stored - called ISOPOD\(^1\) - thus holds an explicit model that defines the types of frames, slots, and facets used by the system.

A screen image showing parts of Creek's internal knowledge base - viewed along structural relations - is shown in figure 5.10. The network was generated by the graphical editor of METATOOL [Aakvik-88]. The figure shows that the internal concepts knowledgebase, slot, facet, and value-expression with sub-components are next level subclasses of iso-thing. A knowledge base has an instance called current-kb, and has part frame. All (explicit) knowledge are contained in frames. Different types of frames are shown, as is the fact that a frame has a part called slot, which has part facet - and so on as earlier described. The slot subclass called transitive-slot has instances for which the inference step "A relation B" and "B relation C" implies "A relation C" is assumed to be valid. A daemon-facet contains a method (lisp-function) that is executed when the condition expressed by the daemon is satisfied. In the van example in section 5.2.2, the if-needed facet is a daemon facet, where the condition that needs to be satisfied is that no explicit value is inferrable for the has-age slot.

In Appendix 2, the representation language is described in more detail, illustrated by examples of how to build and modify frame structures and how some simple inheritance and constraint enforcement methods work. A simple example of basic spreading activation is also included.

The next chapter presents Creek's algorithms for problem solving, reasoning and learning at a certain level of abstraction. The algorithms are described within the framework of chapter 3. This also involves a more detailed description of the OKM, DRM, CRM and SLM modules, and how they interact. The chapter following the next one introduces the application domain that has been used to focus the development of Creek: Off-shore drilling problems, particularly related to diagnosis and treatment of drilling mud. This domain is then used to exemplify

\(^1\) (An isopod is a small animal with seven pairs of legs. Seven is also a rough estimate of the number of concepts that a human mind can hold in its short term memory [Miller-56]. Some isopods live in creeks and other muddy waters.)
characteristic properties of the Creek architecture, by following a problem solving and learning process step by step. In the final chapter, the Creek approach is discussed by relating it to the requirements of section 3.1 and the critical factors of section 4.2, as well as by comparing it with the system approaches discussed in chapter 4.

Figure 5.10: Structural relations in the ISOPOD knowledge base
The figure shows the structural relations between internal organizational concepts of Creek. Unbroken lines denote subclass-of relationships, broken lines denote part-of, and dotted lines denote instance-of relationships. All instances of SLOT are predefined, primitive relations, and all instances of FACET are predefined facets. (The figure is a screen dump from a version running on Sun.)
Chapter 6

Problem Solving and Learning

The integrated problem solving and learning methods of Creek are based on the process models of the framework described in chapter 3 - and summarized in figure 3.19. The problem solving process is a specialization of the UNDERSTAND-GENERATE-SELECT process, as described in chapter 3.5.5. Problem solving in Creek is controlled and guided by the Diagnosis and Repair Model (chapter 5). This is a generic model of diagnosis and repair that specifies a goal and task structure for the diagnostic process. The generic nature of the DRM model requires that it be specialized and refined for a particular application. However, the descriptions in this chapter will be held at a general level, presenting an overview of the different methods and algorithms, and how they play together. The final section of this chapter summarizes the architecture and properties of Creek by relating them to the framework requirements and critical factors as done for the four systems described in chapter 4.

Initially, the Diagnosis and Repair Model (DRM) will be described, since this model is important for focusing the UNDERSTAND-GENERATE-SELECT steps in Creek. The underlying methods within each of these steps are then described and related to the control structure defined by the DRM. In describing each of the three problem solving steps, the underlying reasoning methods are outlined. The reasoning cycle within each problem solving step is based on the ACTIVATE-EXPLAIN-FOCUS model. The emphasis is put on the methods for activating relevant parts of the knowledge base, and for generating and evaluating explanations. The final part of the chapter describes the learning method based on the EXTRACT-CONSTRUCT-STORE steps, i.e. retaining relevant information from a problem just solved (or an attempt made to solve it) and integrating the new experience into the existing knowledge base.

Figure 6.1 illustrates the major roles of the three framework models in integrated problem solving and learning. It should be noted that the DRM is a control module at the level of problem solving strategy and tasks, i.e. a model describing an external behaviour of a system, specifying what the system should do. The Combined Reasoning Model (CRM) and Sustained Learning Model (SLM), on the other hand, control the inner workings of a system, i.e. how it achieves its results.
While a role of the CRM is to specify a strategy for combining associational reasoning from past cases with reasoning from heuristic rules, as described in the previous chapter, knowledge-intensive case-based reasoning is the primary reasoning paradigm investigated in this research. Hence, the description of the problem solving and reasoning processes in this chapter will primarily focus on the mutual support of model-based and case-based reasoning. In this respect, an object level rule is merely regarded as a shallow, complex type of relationship.

![Diagram](image-url)

**Figure 6.1: Control Modules and Top Level Processes in Creek.**

The figure shows the correspondence between control modules and the processes of problem solving, reasoning, and learning which they control. DRM = Diagnosis and Repair Model, CRM = Combined Reasoning Model, SLM = Sustained Learning Model.

Within the Creek architecture, the CRM and SLM have the potential of being developed into comprehensive control modules for reasoning and learning. The explicit control level knowledge of these two models has not been thoroughly developed in the present Creek system.

The next subchapter describes the Diagnosis and Repair Model. In order to prepare for later subchapters, the presentation of the DRM is followed by a discussion of the explanation mechanisms in Creek. On this basis, the problem solving and learning methods are then described.
6.1. The Diagnosis and Repair Model

Diagnosis (and, to some extent, repair) is the problem type that expert systems have traditionally attacked, since the early days of MYCIN [Shortliffe-76] and PROSPECTOR [Duda-79]. Many problems in industry are diagnosis type problems, being either trouble-shooting problems or closely related types of data interpretation problems. A lot of research has been done to understand and develop generic models of diagnostic and trouble-shooting processes [Chandrasekaran-83, Feltovich-84, Clancey-85, Bennett-85, Sembagamoorthy-86, Keravnou-86, Fink-87, Breuer-87]. The Diagnosis and Repair Model in Creek (DRM) has partial similarities with some of these models, but it differs from all of them in its multi-paradigm problem solving approach with an emphasis on case-based methods.

The DRM contains three submodels: A model of diagnosis and repair concepts, a high level process model of diagnosis and repair, and a more specific task model of diagnosis.

The model of diagnosis and repair concepts defines concepts like finding, diagnostic state, diagnostic hypothesis, fault, treatment (used to denote a particular repair action), etc. In order to achieve a satisfactory degree of understanding, a system needs to relate its object level domain concepts (OKM concepts) to their role in the diagnosis and repair process. For example, the system must know that 'no dash-board lights' is a finding (i.e. an observation or measurement), and that a broken battery cable is a fault. The model of diagnosis and repair concepts is used, for example, to check whether a problem descriptor is classified as a finding within the problem domain, and to discriminate between normal and abnormal states. In general terms, the most important role of this model is to enable a system to generate better explanations, based on knowledge of the meaning of domain concepts within a diagnosis and repair context.

The high level process model represents the sequencing of general fault finding and repair tasks, and what conditions that should be satisfied in order to move from one task to the next. It defines the top level, major tasks to be conducted, and is used to guide the system throughout the entire diagnosis and repair process.

The task model of diagnosis is a specialization of the diagnosis task, i.e. the task of arriving at a plausible, justified diagnosis. Although the Creek architecture is aimed towards both diagnosis and repair, diagnosis is still regarded as the most complex task. Therefore, a more detailed model of diagnosis than provided by the high level model is specified. Diagnosis will also be the task given the most attention in the subsequent description of problem solving and learning methods.
6.1.1. A Generic Model of Diagnosis and Repair Concepts

Figure 6.2 illustrates parts of Creek’s concept model of diagnosis and treatment. The most general concept is called feature, i.e. a descriptor in the form of a relationship between a target concept - the concept to be described (e.g. my-car) - and a feature value (e.g. has-make Saab). Features may be findings, physical states or diagnostic hypotheses. A physical state is an intermediate state description, between a finding (e.g. an observation) and a diagnostic hypothesis (e.g. a fault). In a medical domain, for example, a finding may be dyspnea (limited breathing ability), and the fault (disease) may be bronchitis, while an abnormal physical state could be constricted bronchiolies. The figure illustrates some important relationships between concepts related to diagnosis and repair.

Figure 6.2: Diagnosis and Repair Concepts
The figure shows some major relationships in the semantic network that describes some basic diagnostic (and repair) concepts. The top level term - feature - is specialized into concept classes which in turn will be used as superclasses for application-dependent concepts. (Some high level concepts are in bold font to improve readability of the figure.)
For example, following a chain of causal relations, a fault typically causes an abnormal-state which in turn (as subclass-of a physical-state) causes a finding (e.g. bronchitis causes bronchoconstriction, which causes dyspnea).

More complex relationships, such as the fact that a successfully repaired abnormal state transforms into a normal state, may be expressed as if-then rules in separate rule frames. A finding is either irrelevant (i.e. noise), a normal finding or an observation indicative of a fault. The latter is called observed evidence. Observed evidence is one type of diagnostic evidence, the other type is inferred evidence, i.e. evidence which has not been observed, but which has been derived from observed evidence (e.g. via a quantitative to qualitative transformation, see upper right part of network).

6.1.2. A High Level Model of the Diagnosis and Repair Process

Creek's top level model of general diagnosis and repair tasks is shown in figure 6.3. The entire process basically contains three types of tasks: Diagnosis (fault finding), treatment (repair), and evaluation of the result of the treatment. This evaluation gives - if negative - feedback to the treatment task, and leads to a revision of the suggested treatment. A negative treatment result may also lead to a revision of the diagnosis.

The main steps of diagnosis are suggesting a fault on the basis of the problem description, predicting its consequences, and checking whether the expected consequences are confirmed by observed or inferred problem findings. If the system is unable to check all relevant expectations using its own knowledge, the user is asked for assistance.

The structure of the repair task is - at this level - similar to diagnosis: Suggest a treatment based on the suggested fault, predict its consequences, and derive their impact on the system being treated. The user may play an active role in anticipating and evaluating consequences of a treatment.

If no negative consequences are found, the treatment is applied, and the result of the treatment is evaluated. The process terminates if the treatment was successful, or if a stop criterion is reached, possibly after several iterations of the process or some subprocesses.

---

1Following the Creek representation paradigm, all semantic relations in the model have their inverse relations explicitly defined. Inverse relations are not shown in the figure.

2The term 'evidence' is used here in a general sense, i.e. as a grouping of certain features that normally are indicative of certain faults diagnostic hypotheses.
The figure illustrates the generality of the model. It is domain independent and, hence, weak. It must therefore be refined into more detailed models, like the generic diagnostic task model described in the next section, and linked to the Object Knowledge Model (OKM). The diagnostic concept model described in the previous section provides a bridge into the OKM. The process model reflects the interleaved nature of diagnosis and repair problems addressed by Creek.

**Figure 6.3: A Top Level, Generic Procedure for Diagnosis and Repair**
A high level algorithm describing the diagnosis and repair process. The gray ellipses are processes, while the polygons are tests that determine the next subprocess to initiate.
6.1.3. A Task Model of Diagnosis

This submodel specifies the major tasks of the suggest fault process, i.e. the initial process of the top level diagnosis and repair algorithm described in the previous section. An illustration of the diagnostic task model is shown in figure 6.4. At an abstract level, it has some similarities with the model of 'systematic diagnosis' shown in figure 3.6, but it is more refined and adapted to the Creek architecture. This task model is static, showing the tasks and their input/output dependencies, not their temporal relations. The Data Gathering task, for example (see figure), is initially guided by the description of the problem environment and the actual problem, while additional gathering of data during the problem solving process also is influenced by the set of candidate faults which have been hypothesized.

Figure 6.4: Diagnostic Tasks
The tasks (corresponding to sub-processes) are shown as ellipses, while input/output from the tasks is shown as boxes. The unbroken arrows are input/output relations, while the short, dashed arrows from the candidate faults and current goal boxes indicate influences from these boxes upon the tasks they are pointing at. That is, the fault hypotheses under consideration influences (constrains) further data gathering and the inference tasks of expanding and restricting the findings into a set of relevant findings. The current goal - dynamically updated during the diagnostic process - has impact on all the tasks.

Similarly, no fault candidates exist to influence the Expansion and Restriction tasks the first time these tasks are performed (since no hypotheses has been generated at that time), so the only focusing information is the current goal. The temporal relationships among diagnostic

---

1This task model is a continued refinement of a diagnosis model developed in cooperation with Ray Bareiss and Rita Duran. The model is briefly described, and compared to other models of diagnostic problem solving (MYCIN, PROTOS, MOLE) in [Duran-88].
tasks are highly complex, since solving a problem often involves backtrackings, shortcuts, iterative loops, etc. among tasks. A generic temporal model of diagnostic tasks would probably not be of much help, since the space of possibly relevant task sequences within, e.g., diagnosis is extremely large. This space of possible time sequences of tasks should therefore be shrunk by the actual application, not by a model at the generic level. In Creek, the temporal (dynamic) task model is assumed to be part of an application-specific control program.

The initial task, however, is always Data Gathering, resulting in a set of observed findings. This findings set is elaborated using the domain knowledge model: The two subsequent tasks, Expansion and Restriction, elaborates on the problem description in order to infer additional problem descriptors and to filter out irrelevant ones. This is a two step process, since expanding the observed findings derives other findings closely related to those observed, but not necessarily relevant for the actual problem at hand. It is the job of the Restriction task to get rid of presumably irrelevant problem features, since such features may disturb the case matching process. The set of relevant findings will consist of the subset of input findings which have not been ruled out as irrelevant, and the set of findings derived from this subset, minus the findings removed because they violate some constraint criteria or have too weak a justification. The methods for inferring additional features are based on a set of feature derivation rules operating within the general knowledge model, and may also retrieve a set of cases pointed to by the existing feature set and use the case findings as expected findings for the current problem.

In order to guide the expansion of a set of input findings in a large knowledge base, a method for focusing this search is needed. In Creek this is a two-step process: First, a relatively broad context is established, by a method called goal-focused spreading activation. Next, a more specific method is applied that expands the initial set of findings within the established context. The job of the restriction task is to constrain the set of observed and derived findings into a set of relevant findings for some classes of diagnostic hypotheses. This is a highly knowledge intensive process, where the model of general domain knowledge is used to identify relevant problem descriptors, given the current goal and the current state of the diagnostic process.

The set of relevant findings for the current problem triggers a reminding to one or more diagnostic hypotheses. The Reminding task will try to match the relevant features of the current problem with previously solved cases, and attempt to retrieve a set of candidate cases. However, the task model also covers the situations where a rule set - or the deep model - is

---

1 The relevant findings are findings regarded as relevant for solving the problem given the current state of understanding in the system.

2 The corresponding Creek methods that enable these tasks - as well as the other tasks of the model outlined here - are described in detail in chapter 6.3, and exemplified in chapter 7.5.
invoked to produce a set of candidate faults\(^1\). The candidate faults (or cases) are evaluated according to some criteria in order to restrict this set to a subset that best matches the current problem (the Evaluation task). From each of these ‘most likely’ fault hypotheses, a set of consequences - i.e. expected findings or physical states - are inferred (the Prediction task). In the Comparison task, these expectations are compared to the currently active states and relevant findings. If more data is needed for this comparison, an attempt to gather the necessary data (for example by asking the user) is made. During a refinement process, where a particular candidate fault is evaluated by checking expectations and inferring consequences, the set of relevant findings is gradually refined into a set of diagnostic evidence (see figure 6.2) for that particular fault hypothesis. The outcome of the comparison task is either a suggested fault, or a rejection of the current best hypothesis. In the latter case another hypothesis with sufficient strength may be tried, another reasoning method (rule-based or model-based) may be tried, or the user/expert may be asked to give a solution.

Problem solving in Creek splits into the three subprocesses of problem understanding, generation of good candidate solutions, and selection of the best solution. The task model presented in this section defines what subtasks of the UNDERSTAND-GENERATE-SELECT process that a diagnostic problem solving process contains. The associated reasoning model - the ACTIVATE-EXPLAIN-FOCUS process describes the underlying methods, i.e. how to enable the

---

\(^1\)Hence, the Reminding task may be viewed more generally as the task of generating candidate solutions, i.e. retrieving and/or deriving a set of diagnostic hypotheses.
tasks to do what is specified in the task model. Figure 6.5 illustrates the coupling between the task model described in this section and the top level problem solving model, achieved by grouping the diagnostic tasks into the three main problem solving steps.

When the steps of Creek’s reasoning model are later presented, an understanding of the basic mechanisms for generation and evaluation of explanations will be assumed. The next subchapter is therefore devoted to this issue.

6.2. Generation and Evaluation of Explanations

This section discusses the basic methods for generating and evaluating explanations, related to their use in inference processes. The intended use - or purpose - of an explanation is an important criterion for guiding the generation and evaluation of explanations. The first section discusses the use of explanations in Creek. This is followed by describing the basic method for generating an explanation, which in turn is followed by discussing explanation strengths and evaluation of explanations.

The general notion of an explanation, and its use in inferencing and reasoning, is discussed as a part of the framework, chapter 3. The representation of explanations in Creek is described in section 5.2.5.4., and their role in abductive inference and plausible reasoning within the architecture is emphasized in chapter 5.2.4.

6.2.1. Use of Explanations in Creek

Based on ‘what is to be explained’, there are four major types of use for the explanation mechanism in Creek. They correspond to the four types of inference tasks listed below, where each inference task is associated with an explanation task which tries to justify an inferred hypothesis:

- Inferring consequences
  - What are the consequences of a proposition A? What does A lead to?
- Matching concepts
  - Is A like B? What are the similarities and differences between A and B?
- Assessing a value
  - Is A a plausible value for B? Are there constraints on B which may exclude A?
- Assessing support
  - Does the existence of A support B? If so, to what extent?

1. Inferring of consequences: What are the consequences of A?
Consequences are inferred by following relational links in the integrated knowledge model. The links may be associative (case remindings, heuristics) or may represent deeper, principled knowledge. Inferring of consequences always starts from a concept (A, above). Thus, inferring consequences may be viewed as a spreading activation process using a particular set of spreading relations, and a spreading control algorithm. The set of spreading relations as well as the spreading control is partly determined by the reasoning method (i.e. case-based, rule-based, or model-based) used to derive the result. A rather thorough explanation will often be needed if an associational reasoning method (case-based or rule-based) has been used, but a consequence inferred by model-based methods may also need justification. The acceptance of a consequence requires that it is coherent with the existing knowledge (see chapter 3.5.4.1). Unless explicitly stated in the value-justification field, the degree of belief in a consequence is determined by how strong the explanatory support is.

Inferring consequences is normally done within the deeper model. The inference task first looks for negative consequences, following negation relations (such as excludes). This is followed by an attempt to derive consequences in general, while looking for other contradictions and possible incoherences. Plausible inheritance rules are used to inherit consequences to additional concepts. Contradictions are detected by constraint violations when entering a value into a slot, and by comparing negative consequences (concepts activated via negative relations) to other activated concepts. When consequences are derived within the deep model, the explanation structure is to a large extent generated as a part of this process. The explanation task then mainly becomes a process of evaluating explanations and selecting the best one. If possible consequences are derived from propagation of rules or retrieval of cases, explanations will have to be constructed afterwards.

Note that the reasoning process which infers consequences follows the general reasoning structure model of ACTIVATE-EXPLAIN-FOCUS: The inferring of possible consequences is the activation of a concept structure, the explanation step is the production of explanations, while the focusing step is the evaluation of these explanations according to the evaluation criteria for this particular explanation task. The same pattern appears in the three other explanation tasks, as described below.

2. Matching of concepts: Is A like (in this context similar to) B?

Concept matching includes the matching of two cases during case retrieval and storage, and - at a more detailed level - the matching of two features of a case (or terms of a rule). The purpose of the explanation process is to assess similarity, not to support a particular inferred fact. The generation of explanations as well as the criteria for evaluation of explanations differ from the

---

1As when, e.g., by-strict-definition is stored in this field.
previous inference task. The inference method to derive a possible match also differs from the inferring of consequences, since a matching process starts out from two concepts in the knowledge network. Explanation structures are generated by propagating from both concepts along a suitable set of relations, while looking for a common 'intersection'. The paths established in this way between the matching candidates may serve as sufficient explanations by themselves. If they are too weak, however, the system should look for explanatory support by trying to find additional explanations for the weakest parts in an explanation chain.

An explanation - or part of an explanation - justifying a match of two features during case matching, typically take one of the following three forms (or a combination of them):

- Explaining 'away' a difference in feature values by showing that the particular feature has a low importance\(^1\) for the retrieved case.

- Showing that the input problem feature does not contradict any strongly relevant features of the retrieved case, its solution or solution constraints.

- Explaining the featural match by searching for an explanatory path in the general knowledge network.

3. **Assessing a value:** Is A a reasonable (plausible) value for B?

Here, A and B are assumed to be a proper value and a concept.relation pair (frame.slot pair), respectively. The most important thing to consider in this explanation task is to check whether the suggested value, A, contradicts other values of B - or their consequences\(^2\). Contradictions are checked by propagating along a particular set of (negative) relations. If a possible contradiction is derived, the system should try to resolve the conflict by generating explanations for each value, i.e. derive consequences of the two facts (inference and explanation task 1). An acceptance level determines the required combined explanation strength needed to confirm the plausibility of a value.

4. **Assessing of support:** Does the existence of A justify or support B?

A and B are propositions, the explanation task is to explain why a fact may lead to - or strengthen - belief in another fact. The first subtask is the checking of contradictions and negative support. If this process does not disqualify A as supportive of B, a search for supportive explanations is performed. Explanations are evaluated according to their

---

\(^1\)Value of the 'importance' key in the feature’s 'miscellaneous' field.

\(^2\)Constraint violations (specified in value constraint facets) are also checked, of course, as part of the attempt to enter a value into a slot. Contradictions are different, however, since such values may be accepted locally, i.e. by the frame or by a submodel, but not within a larger context.
explanatory strength (as described in the following two sections). B is regarded justified by A if there is an explanation from A to B with strength above a certain level.

6.2.2. How to Produce an Explanation.

As indicated above, the production, evaluation and final selection of an explanation in Creek follows the basic ACTIVATE-EXPLAIN-FOCUS cycle. This section describes the first step, while the other two are discussed in the next section. It should be clear from the previous section that production of explanations - as well as their evaluation - depends on the type of explanation task involved. This and the following sections present a general method of explanation generation and evaluation in Creek, without elaborating on details regarding the different explanation types.

Some relations in a Creek knowledge model are classified as explanation relations. The basic set of explanation relations contains the structural, functional, causal, and implicational/correlational relations. This set will to some extent be application dependent, and the basic set should be extended and refined during the initial knowledge analysis and modelling phase. Each explanation task, as described above, should have its own subset of explanation relations. Each explanation task has its own algorithm for propagation control.

An explanation process is always given two concepts (or concept sets) and a task type as input (e.g., B consequence-of A). An explanation structure is produced simply by propagating activation from each of the two concepts along the set of relations defined for the current explanation task. The explanation structure produced is a list of possible explanations, where each explanation represents a path between the two concepts. An explanation chain may contain multiple subpaths between any two of its nodes.

6.2.3. How Explanations are Evaluated

The method for evaluating explanations has some similarities with the way this is done in the Protos system. The differences are related to Creek's use of different evaluation criteria for different explanatory purposes, to the way relations are represented, and to the fact that Protos has a pre-defined set of relations with predefined default strengths, while the basic set of relations used for explanations in Creek is extended and refined as part of the initial knowledge analysis and modelling phase.

---

1 That is, they are instances of the explanation-relation class - a subclass of semantic-relation.

2 Even when the inference task is to infer consequences of a proposition, the explanation process does not start before one or more candidate consequences have been hypothesized.

3 Protos' combination of qualifiers and a general relation (e.g., sometimes strongly implies) would be represented by a special relation (e.g. sometimes-strongly-implies) in Creek.
How good an explanation is, is determined by:

- the default strength of each relation in an explanation chain
- a set of explanation evaluation rules

Each explanation relation has a default explanation strength as the value of its has-explanation-strength slot. A basic set of explanatory relations are part of the ISOPOD knowledge base, for example cooccurs-with (0.7), always-cooccurs-with (0.8), causes (0.9), always-causes (1.0), where default explanatory strengths are indicated in parentheses. Different sets of explanation relations may be defined for the different purposes of explanations (i.e. the different explanation tasks, as described in the previous section). Identifying other explanation relations, and assessing their strengths, is an important subtask of the system design phase, which typically will involve several test&refine cycles. The explanation strengths of all relations in an explanation chain are combined to form a single explanation strength for the total explanation:

The explanation strength of a starting concept in an explanation chain is always assumed to be 1.0. Unless only relations with default strengths equal to 1.0 is followed, the accumulated strength of a single explanation path is weakened as more relations are followed. However, a concept may be explained by several explanation paths, i.e. several explanation paths may lead to a concept node. In this case the explanation strengths of all single paths are combined into a total explanation strength for the concept. The strength of an explanation is computed by taking the strength of the initial concept in the explanation chain and propagate explanation strength values to succedent concepts in the chain, in the following way:

Given a concept, $C_1$, with explanation strength $S(C_1)$, an explanation relation, $R_{12}$, with explanatory strength $S(R_{12})$, and a concept pointed to by this relation, $C_2$, with an existing strength $S(C_{2old})$. The new strength of concept $C_2$ is computed according to the following formula:

$$S(C_{2new}) = S(C_{2old}) + k_{old} \times S(C_1) \times S(R_{12}) \times (1 - S(C_{2old}))$$  \hspace{1cm} (I)

1Note that relation names not starting with a qualifier (always, sometimes, etc.) should be interpreted as qualified by 'typically' or 'usually'.

2The basic formula for calculating a total, combined explanatory strength from a sequence of single relationships, has the same structure as Mycin’s formula for propagating certainty factors along a chain of rules. Hence, an explanation strength cannot exceed 1.0, but should be as close to this value as possible.
The last expression in the formula represents the current ‘weakness’ of the explanation, and ensures that the relative contribution to the total strength is larger the weaker the existing explanation is.

The constant k determines how strong effect the contribution from a new path will have on the old explanatory strength assigned to a concept. The constant has a value larger than 0 and equal to or less than 1. The value of k is set by the system designer, and it is altered automatically by the system if one of the following two situations occur: (1) If the concept C₂ does not have an old strength - i.e. if $S(C_{2\text{old}})$ equals 0 - the constant k is set to 1.0. (2) An explanation often terminates on one or more findings. An explanation is strengthened if more than one finding is explained. The degree of strengthening of an explanation due to additional findings being explained depends on the ratio of current problem findings to the total number of findings which are explainable. If a finding $C₂$, is pointed to by a concept $C₁$, and another finding, $C₂'$, also is pointed to by $C₁$, then the formula for the strength of $C₂'$ takes the following form:

$$S(C₂') = S(C₂) + k_{\text{cover}} * S(C₁) * S(R_{12}) * (1 - S(C₂))$$

That is, the explanatory strength assigned to the first finding is regarded as an 'old' strength when computing the accumulated strength assigned to the second finding, and so on. The constant $k_{\text{cover}}$ is calculated by the system as the ratio of activated findings related to $C₁$, to the total number of findings related to $C₁$.

As a simple example to illustrate the user of formula (I)\(^1\), consider the explanatory strength of the hypothesis \textit{low-battery-voltage} ($C₁$) for explaining the observation \textit{starter-motor-not-turning} ($C₂$). A part of the total explanation structure is a single-step path along a \textit{causes} relation ($R_{12}$). The strength of the assumed hypothesis is, by default, 1.0\(^2\). If the existing strength of the explanation - which may be expressed by $S(\text{starter-motor-not-turning}_{\text{old}})$ - is assumed to be 0.8, the explanatory strength of a \textit{causes} relation is 0.9, and k is 0.5, the resulting explanation strength becomes:

$$S(\text{starter-motor-not-turning}_{\text{new}}) = 0.8 + 0.5 * 1.0 * 0.9 * 0.2 = 0.89$$

In addition to the basic strength computed in this way, Creek has a method for evaluating the total quality (‘goodness’) of an explanation, by applying a set of control rules called \textit{explanation evaluation rules}. There are two types of explanation evaluation rules, corresponding to whether they are triggered \textit{during} the search for explanation paths and computation of basic explanation strengths,

\(^1\)A more extensive example, including the use of formula (II) is shown in chapter 7.
\(^2\)That is, a search for an explanation path starts by assuming that the hypothesis is true, and the resulting explanation may result in a confirmation of this strength value, or a reduction of it.
or after this process has terminated. The first type of rules uses the basic strength of the path up to
the standing node, together with information about the explanation task, the fault context, and the
problem solving goal, to determine which relation is the most promising to further propagate along.
The purpose of this type of rule is to guide and focus the production of explanations, by evaluating
the strengths of sub-paths as they are produced, and direct the search according to context dependent
criteria. The second rule type guides the evaluation of completed explanation chains, in order to
select among competing explanations. These rule sets should be developed as part of an application.
(In Protos, for example, two rule sets are defined to guide the production of explanations to explain
case matching during the knowledge-based pattern matching process (see chapter 4.3.3). One set is
related to the matching of problem features to exemplar (case) features, and the other to matching of
problem features to categories (corresponding to fault classes in Creek). These rules were specified
in Protos through the development of a particular application (auditory diseases), and their generality
as explanation evaluation rules has not been tested).

6.3. The Problem Solving Process

The problem solving process in Creek is based on the three-step model of UNDERSTAND - GENERATE - SELECT as described in the framework chapter, section 3.5.5. In each of the three
problem solving steps, the problem solver runs through a reasoning cycle of ACTIVATE - EXPLAIN - FOCUS steps. The integrated problem solving and learning cycle is therefore based
on the framework’s model of problem solving, reasoning and sustained learning as illustrated in
figure 3.19.

This subchapter describes the problem solving method in Creek, with reference to the
UNDERSTAND-GENERATE-SELECT model, and the Diagnosis and Repair Model of subchapter
6.1. To start with, the Combined Reasoning Model is briefly described. This model controls the
multiple reasoning methods available to the problem solving process.

6.3.1. Combined Reasoning in Creek

The top level method for combined reasoning was described in chapter 5.1.2 (Functional
Architecture). In this section, the reasoning control method is further detailed, with emphasis
on the case-based reasoning process. The case-based method is the primary reasoning paradigm
in Creek, the other methods are used - as separate reasoning methods - only if the case-based
method is unable to suggest a solution.

\[\text{The structure of these rules is similar to the 'explanation heuristics' defined in the Protos system [Bareiss-88b], as}
\text{exemplified in section 4.3.3. In Creek these rules may be regarded as a particular type of plausible reasoning rules.}\]
The high level algorithm for combined reasoning is shown in figure 6.6. The choice of reasoning method is made after the system has gained an initial understanding of the problem. This initial understanding process (described in the next section) results in an activated problem context, including a set of relevant features for describing the problem, a structure of problem solving (sub)goals, and a hierarchy of possible faults.

**Figure 6.6: Combined Reasoning in Creek**

This figure views the problem solving process from a reasoning method perspective. The three reasoning methods of Creek are illustrated by the three 'Attempt XXX' boxes, of which only the Case-Based Reasoning (CBR) box is detailed. All problem solving steps, from the receiving of relevant features until a solution is evaluated and accepted, are contained in these three boxes. (RBR = Rule-Based Reasoning, MBR = Model-Based Reasoning).
First, Creek will attempt to solve the problem by case-based reasoning. The relevant findings are combined into a set of remindings, where each reminding points to a case (or a class of cases) with a certain strength. If some cases are pointed to by remindings with strengths above the reminding threshold, the case most strongly reminded of is retrieved. If no such reminding is produced, the system may trigger its rule-based reasoning method. However, before doing that it will normally try to elaborate on the findings of the case most strongly reminded of. The purpose of this is to improve a weak match by looking for common states, constraints, etc., which will imply a stronger similarity than determined by the basic case retrieval method. Whether the elaboration on a weak match is attempted or not depends on the strength of the strongest reminding and the size and strength of the case base relative to the rule base.

If an acceptable match is found, the case is used in the attempt to solve the problem. If not, rule-based reasoning is attempted\(^1\). If no cases were reminded of in the first place, Creek will also try its rule-based reasoning method, i.e. attempt to solve the problem by a combined forward chaining (from the relevant findings) and backward chaining (from the fault hierarchy) within the rule base\(^2\).

When a case is retrieved, the solution (fault and - possibly - treatment) is evaluated to see if it is acceptable for the current problem. If the system is unable to produce a good enough explanation to accept or reject the solution candidate, it is presented to the user for evaluation. If for any reason the solution is unacceptable, a check is performed to determine whether the solution would be accepted if slightly modified, in which case a modification is attempted. When no more modifications are relevant and no more new cases are available for use, Creek gives up case-based reasoning.

Figure 6.7 illustrates the abstract structure of case-based reasoning in Creek, according to the ACTIVATE-EXPLAIN-FOCUS model. The input to a reasoning process is a problem description. This may be a description of the user's problem, or a partial solution of this problem - for example a set of descriptors which includes a fault hypothesis, given as input to the retrieval of a case containing a suitable repair.

The next three sections describe the UNDERSTAND-GENERATE-SELECT process. The UNDERSTAND process is general, while the GENERATE and SELECT processes are described from a case-based reasoning point of view.

---

\(^1\)This step may also be re-run after the rule-base has been tried, but before the presumably less efficient model-based method is attempted. This is not shown in the abstracted algorithm in the figure.

\(^2\)A rule-based method has not been specified in detail as part of the work reported here. However, Christian Larssen implemented a rule-based reasoner within the CreekL representation system as part of his Diploma Thesis [Larssen-90].
6.3.2. Understanding the problem.

This process attempts to give the problem solving system an initial understanding of the problem. A problem will often not be 'understood' before it is finally solved (and sometimes not even then). So, the term 'understanding' should not be taken too literally, but be interpreted as initial understanding in the sense of trying to make some initial 'sense' of the input data and problem solving goal. The main purpose of this initial understanding step is to gather and/or infer sufficient information about the problem to be able to discriminate potentially relevant problem features from irrelevant data. A second objective is to gather/infer information about the goal of the problem solving process, so that an appropriate goal/subgoal structure may be instantiated. The goal structure is application-dependent, and corresponds to a set of application-specific sub-problems that need to be solved in order to achieve the goals. In the car starting domain, for example, a subgoal could be find-fuel-system-fault, which has been reached via the goal find-fault. In a medical domain a specific goal could be check-for-gastric-ulcer. Each goal has associated a set of fault hypotheses, such as blocked-fuel-filter, or peptic-ulcer. The function of the goal structure, with its associated fault hypotheses, is to guide and constrain the problem solving process¹. A goal structure will be part of the application domain

¹In the diagnostic task model (illustrated in figure 6.4), this is illustrated by the GOAL box placed in the lower right corner of the figure.
model, and the job of the ‘Understander’ is to reduce this general structure into one that
describes the goal structure for the current problem. The understanding process creates an
instance of the general goal structure and fault hierarchy that represents the current problem.
An example fault hierarchy, taken from the car starting domain, is illustrated in figure 6.8. The
figure also shows part of a findings hierarchy and a repair hierarchy. Such hierarchies
constitute an extension of the diagnostic concept model (figure 6.2) for a particular domain.
For clarity, the figure only displays subclass/superclass relations.

Three important questions need to be answered in the understanding process:
- What is the problem that needs to be repaired?
- How does the problem manifest itself in the findings?
- What type of external requirements are imposed on the solution?

The first question is related to establishing the goal structure and fault hierarchy, as just
described. The second question concerns how to select relevant findings. The third question
aims at identifying the external requirements on a solution, e.g. particular properties of a
particular solution required by the user or by environmental circumstances. These requirements

![Diagram of fault hierarchy and findings hierarchy for car-starting domain]

**Figure 6.8:** Example hierarchies of findings, faults, and treatments of the car-starting
domain.
The figure exemplifies a small part of what may constitute an abstraction hierarchy of domain-
dependent diagnostic concepts. Such a model would be integrated into the generic diagnostic
concept model shown in figure 6.2.
set up constraints which may influence the instantiation of a goal and task structure, as well as the identification of relevant findings and diagnostic evidence for the current problem. Such requirements may be a given set of faults to select from, a set of faults to disregard, time constraints, requirements for non-risky solutions, the fact that only a limited set of treatments may be available, a request for a particular type of treatment, etc.

Given a problem description in the form of a list of descriptors and a goal, the UNDERSTAND process runs through the following steps:

UNDERSTAND

U1. Establish problem context by goal-focused spreading activation.
U2. Establish goal structure and fault hierarchy.
U3. Infer a set of relevant findings from those observed.

U1. Establish context by goal-focused spreading activation.

This step determines the general context in which subsequent problem solving tasks will take place. It may be viewed as the initial step of the expansion task (cf. figure 6.4), since both the observed findings and the problem goal gives rise to a broader structure of concepts - with attached cases and heuristic rules. However, the context establishing step may also contribute to the restriction of the finding set, by excluding noisy findings.

A problem description is a list of descriptors that are either facts (propositions) of the form entity-relation-value, or findings defined as concepts (e.g. no-starter-click). Context is determined by spreading from nodes corresponding to findings explicitly represented as concepts as well as target concepts of findings represented as triplets. As previously mentioned, target concepts are concepts which are targets of a description, i.e. concepts intended to be described. The opposite role of a concept is as a value concept, i.e. the concept intended to describe a target concept. For example the finding car-0256 has-color green will spread from car-0256 since it is an instance of car and inherits car's property of being a finding-target concept. It will spread from green only if green, too, is defined as a target concept for a finding, and not if it is only defined as a feature-value within the domain model.

The spreading algorithm that achieves this is shown in figure 6.9. Basically, the spreading process starts from two ends: Findings and goals. The nodes which are activated from both ends are marked. These nodes, and the nodes lying on paths between any of these nodes and a node representing an initial finding, constitute the resulting set of activated nodes. The two relations lists referred to in the algorithm contain predefined sets of spreading relations, i.e. relations to use for propagation of activation between nodes. The relations in these lists will depend on the domain, but *goal-spreading-relations* typically includes relations such as has-
subclass, has-associated-fault, caused-by, implied-by, has-evidence, etc., while *finding-spreading-relations* at least should include has-synonym, subclass-of, part-of, causes, implies, evidence-for.

1. Activate the node, or nodes, representing the goal. Call the resulting node set \( G_I \).
2. Spread activation exhaustively from the nodes in \( G_I \) along the relations that are members of the relation set *goal-spreading-relations*. Call the resulting set of activated nodes \( G_A \).
3. Activate the node, or nodes, representing the observed findings. Attempt to transform quantitative findings into qualitative ones. Call the resulting node set \( F_I \).
4. Spread out from the nodes in \( F_I \) along the relations that are members of the relation set *findings-spreading-relations*. Mark the nodes in \( G_A \) that are hit by such a path, and call the resulting node set \( G_H \). Activate all nodes lying on paths between \( F_I \) and \( G_H \). Call the resulting set of activated nodes \( F_A \).
5. The activated context is established by taking the union of the node sets \( G_A \) and \( F_A \). Call this node set \( C_A \).

**Figure 6.9: Goal Focused Spreading Activation**

The algorithm shown establishes the broad context wherein further reasoning and learning takes place. Basically, the context is determined by spreading from the goal to other goal-relevant terms, and from the observed findings describing the problem. The context is defined by those concepts that lie on paths between goal concepts and findings.

The purpose of establishing this context is to constrain the subsequent derivation of relevant findings, the generation of candidate fault hypotheses, and - more generally - the search for explanations. How broad the resulting context becomes, is determined by the set of spreading relations. Hence, these relations should be selected in such a way that all possibly relevant parts of the knowledge structure gets activated without activating more irrelevant parts than necessary. The context established here is just the initial, high level context. Other focusing methods (e.g. constraint-based and explanation-based methods) are used to determine more specific contexts related to parts of the problem solving process. The context may later also need to be expanded, for example after new (additional) observations are entered during evaluation of an hypothesis. A context is expanded by re-establishing the context 'from scratch', or by incrementing the existing context. The latter method is preferred, and applied unless severe modifications to the initial context have been made.
Figure 6.10 illustrates some important terms referred to in the algorithm. It shows a network of nodes and links, representing concepts and relations of a domain model. A goal concept and three observed findings are given as input (thick circle and boxes, respectively). The activation spreads from the goal concept and from the input features following a specific subset of relations. All concepts encountered by spreading from the goal are activated, while spreading from findings only activates those concepts that are on paths leading to an activated goal concept. The dashed boxes show concepts spread to from findings that did not get activated. The thick dashed box represents an observed finding which is not related to any concept of the goal structure, and which is therefore regarded as noise and deactivated. A Creek system will always ask the user for confirmation before it deactivates an observed finding.

**Figure 6.10: Activation of the Broad Context.**

The thick-lined boxes in the lower part of the network are input terms describing observed findings. The thin-lined boxes shows concepts (i.e. nodes) that have been activated from input findings, and the circles are goals and goal-activated concepts. Findings-activated concepts that do not lie on paths leading to a goal-activated concept, are deactivated.
The two remaining steps of the initial understanding process, briefly described below, are mainly contained within the restriction task of the task model. They start with the established context and attempt to reduce the goals and findings activated by the context mechanism by making use of more focused methods.

**U2. Establish goal structure and fault hierarchy**

This step starts the process of refining and focusing the context. The context established by the spreading activation method provides a goal structure and a set of target concepts for inferring relevant findings and intermediate diagnostic states for the problem. These sets will typically be too large to be useful as operational goals, relevant diagnostic hypotheses and relevant features describing the actual problem instance. The established context should be regarded as a context for a class of problems, exemplified by the present one.

The purpose of this step is to specialize and instantiate a goal structure and fault hierarchy for the present problem. During the establishing of context, the relations used to infer the context were pre-set, and applied unconditionally. A goal structure for the current problem case, with an associated fault hierarchy, enables further inferencing to be performed only if certain goal-related conditions are met.

To instantiate the goal structure and fault hierarchy, the system looks for particular constraints related to subgoals and faults (such as only to consider a particular set of faults or fault types, not to consider a particular fault set, etc.). These constraints are user-specified requirements on a solution or a way to achieve a solution. They may be stated as absolute or conditional requirements, for example dependent on certain observed findings being present. If no constraints are given, the entire goal and fault structures, as established by the context mechanism, is instantiated.

**U3. Infer a set of relevant findings from those observed**

The purpose of this step is to increase the system’s understanding of the present problem, by using the observed, non-noisy findings to infer other findings relevant for describing the problem. Some inferred findings may result from - or be confirmed by - the deep model (for example, compartment-light.has-status.off inferred from no-dashboard-lights and knowledge of the wiring connections between these lights). Other inferred findings are regarded as expected findings, to be confirmed by the user.

---

1For example, a control rule may state: "If the fault type assumed is \( \text{elsystem-fault} \), then a valid plausible inheritance rule is: \( \text{If } A \text{ connected-to } B \text{ and } A \text{ has-state.abnormal } \Rightarrow B \text{ has-state.abnormal} \)."

2The instantiation process creates instances by appending a number to the name of a goal or fault type. The number is increased each time a new instance of a class is created by the reasoning process.
This step contains two substeps, corresponding to parts of the Expansion and Restriction tasks, respectively (see the task model, figure 6.4.). The result of this step will be a set of findings assumed to be relevant for the present problem, containing observed as well as inferred findings. The previous two steps activated a set of possibly relevant findings, and established a hierarchy of relevant goals and fault types. This step uses stronger knowledge-based methods to infer additional information about the problem and to limit the set of findings activated in the spreading process to those assumed to be relevant for the current problem instance. Findings defined as facts (concept-relation-value triplets) may also lead to derivation of other findings. Two methods are used to expand the set of findings, based on those observed:

1. Application of a set of feature derivation rules.
2. Generation of expected findings by retrieving past cases with similar problem descriptions.

**Feature derivation rules** are particular control rules for inferring additional features from existing ones. They are a special type of plausible inference rules that may infer findings from other findings, intermediate states from findings, as well as intermediate states from other states. In this step only the findings-to-findings rules are used. An example of such a rule is (all concepts are assumed to be within the context):

   If a finding cooccurs-with another finding, and possible constraints for the cooccurrence are satisfied, then include the other finding in the set of derived findings.

The second method attempts to derive additional findings by following case remindings to previous cases. A combined reminding is computed from the observed findings, in a similar way as in PROTOS. The criterion for a case to be retrieved is that the combined reminding is above a certain strength value (e.g. 0.9)\(^1\). The cases retrieved in this way are first checked to see whether their solutions are within the current fault hierarchy. Then their relevant findings are appended to the list of findings derived by method 1.

The findings derived by these two methods should be regarded as expectations which need some kind of justification or confirmation before they can be added as actually relevant findings for the current problem. The methods used to derive additional findings are rather optimistic, and the justification process will typically lead to a restriction of the findings set. Two methods are used to assess the relevance of the derived features:

1. Attempt to explain the relevance of each derived finding
2. Ask the user for confirmation of unexplained findings

\(^1\)If a censor (negative reminding) to a case is present among the findings, the combined reminding gets the value 0.0, irrespective of how strong positive remindings other findings may generate.
The explanation process first checks for contradictions among the findings. The absence of contradictions is regarded as a weak support for a finding’s relevance. Next, an attempt is made to produce an explanation chain which relates a derived finding to a fault in the fault hierarchy. To be considered relevant, this explanation needs to be above a certain strength threshold. It can be assumed that the context mechanism has already established one or more paths between the finding being assessed and the hierarchy of goals and faults. The explanation process searches this path structure, and looks for additional relations and combinations of relations in an attempt to construct as strong an explanation as possible in support for the features relevance. Note that this process does not attempt to explain the plausibility of each fault; the faults are regarded as equal with respect to explaining the relevance of the finding. Hence, this explanation task is simple compared to other kinds of explanations in Creek. The explanation process uses a set of control heuristics which determine the relations to look for, their individual strengths, and how they combine into a total strength for the explanation.

If an explanation is not strong enough for a derived finding to be included in the relevant finding set, it may still be strong enough to treat the derived finding as an expectation. In this case, the user is prompted for confirmation or rejection of the finding.

6.3.3. Generating a set of candidate solutions

This process starts out from a set of relevant findings, with the goal of producing a set of likely solution hypotheses. The description given here emphasizes the case-based reasoning method, where the more general knowledge types (deeper models, heuristics) are used to support the case retrieval and adaption steps.

The generation of candidate solutions consists of two types of subprocesses: Finding a set of cases that sufficiently match the current problem description, and - once this is achieved - deriving a set of plausible solutions by evaluating the applicability of the solutions in the matched cases to the current problem. If no solutions seem applicable, the SELECT process is triggered in order to select the strongest candidate, after which GENERATE again gains control and uses this solution to generate one or more modified solutions. The modified solution must be justified by the underlying deeper model as being a plausible solution to the current problem.

Generating candidate solutions in Creek is a two step process:

**GENERATE**
- G1 Retrieve a set of matching cases
- G2 Derive a set of plausible solutions
**G1. Retrieve a set of matching cases**

This subprocess contains three major steps, where the first is to combine the relevant findings and the current goal into a reminding. The current goal limits the candidate cases to those useful for solving the current (sub)problem. In the description given here, the top level goal has already been chosen, namely to find a fault. However, since the case retrieval step may be performed for other goals (e.g. finding a proper treatment) as well as subgoals (e.g. choosing a measuring technique in order to acquire more data), the goal is an important part of the reminding process. The idea of a reminding is as a shortcut from a problem description to a solution of a (sub)problem, and it is used here as a metaphor based on the role of remindings in human problem solving (see, e.g., [Schank-82]). In Creek, a reminding from a set of problem features (findings, faults, treatments, problem solving goals) to a case is derived by looking up the slot name relevant-feature-in of each feature concept. This slot lists the cases for which a feature concept is a relevant feature, and a numeric relevance factor is attached to each case in the list (see section 5.2.5.2). A maximal strong reminding to a case is derived if there exists a case which is pointed to by all the features.

A combined reminding is computed from the set of relevant features to each of the cases reminded of by a single feature. A minimal requirement for a case to be reminded of is that its context is the same (expressed by the problem solving goal and the general fault class), and that there are no censors from any of the problem findings to the case. A relevance factor of 1.0, for a problem finding, will then always give a reminding strength of 1.0. If no finding is found to have this relevance factor, an additional requirement comes into play: More than 1/3 of the relevant problem findings have to be present in a case, in order for a reminding to be generated. The reminding strength is then calculated for each such case, as the ratio of relevant problem findings multiplied by a combined relevance factor, calculated from all the relevant findings pointing to the case. The combined relevance factor is simply the average (the arithmetic middle) of the relevance factors for all findings pointing to the case. The combined relevance factor has to be above a certain threshold value for the case matching process to continue.

The second major step retrieves the cases which are pointed to by remindings whose strength is above a certain threshold level - the remindings threshold\(^1\). This level is adjusted according to the behavior of the system, as described in chapter 5.1.2.

\(^1\)As noted before, this description focuses the case-based processes, and the rule-based and model-based reasoning resulting from all reminding strengths being below this threshold is not discussed at this level of detail.
The third step tries to justify the similarity of cases reminded of, based on a more thorough analysis of the similarity in findings. The reminding process will retrieve cases that, after closer examination, may turn out to have significant dissimilarities with the current problem. This is due to the rather syntax-oriented criteria for matching implicit in the method of deriving remindings: The use of a percentage of the findings present, without considering the possible impact of unmatched findings. This step examines the non-similar findings, and attempts to explain their similarity - or their irrelevance - in the current context. If this explanation process does not succeed in producing an explanation with an acceptable strength, the case is removed from the list.

**G2. Derive a set of plausible solutions**

While the step described in the previous paragraph retrieved a set of cases by matching the problem description, this step tries to improve the match by evaluating the degree of fit for each solution of the cases that match on findings.

The checking of whether solutions from the retrieved cases may apply to the current problem involves examining the requirements that each solution candidate put on the problem findings, followed by an explanation process to justify a candidate as a plausible solution to the problem:

If a piece of evidence is necessary for a fault, that particular evidence should be in an activated state, otherwise the fault hypothesis is rejected. For the remaining solution candidates, an explanation process checks for consequences of a solution, tries to explain expectations in findings and intermediate states inferred, and - if necessary - asks the user for additional information or confirmation/rejection. This process may end up with a set of plausible solutions to the present problem, one solution, or none. In the first situation the control is handed over to the SELECT process, the second is trivial, and the third leads to an attempt to modify one of the previous solutions.

The principle for modification of a solution is simple: As previously described, the explanation of a solution is stored with the solution in the case structure. The system backtracks from the case solution, within the explanation structure, until it reaches an intermediate state which is shared by the current problem. Then a forward propagation within the current problem solving context, along explanation relations, is started from that node. A set of domain dependent control rules is used to decide what successive combinations of relations to propagate along. The inferences made in this process are automatically justified, since the control rules ensure that only those combinations of relations that produce sufficiently strong inferences are allowed.
An example: When trying to start a car, it is observed that the engine is turning, but it will not fire (Findings: engine-turns and engine-does-not-fire). A previous, similar problem was explained by a jammed carburettor valve, which caused too rich a gas mixture to enter the cylinders, which in turn lead to no ignition of the gas mixture (Normal physical state: fuel-reaches-cylinder-chambers. Abnormal physical state: no-ignition-in-chambers caused by the Abnormal physical state: too-rich-gas-mixture, caused by the Fault: carburettor-valve-jammed). This fault is suggested for the problem case, but the carburettor is ok. The solution is then modified by searching the explanation path of the previous solution. This is done by starting with the fault concept and following the explanation path until a common abnormal state is reached. The closest common abnormal state is no-ignition-in-chambers. The propagation controller then looks for a strong explanation relation extending from this concept (e.g. caused-by, subclass-of, subprocess-of) and finds that a lack of ignition in the cylinders also is caused by water in the gas mixture. This causal relationship is conditioned to apply only within a specific context, expressed by a particular physical state:

\[
\text{no-ignition-in-chambers} \\
\text{caused-by value } ((\text{too-rich-gas-mixture} \\
\text{... context fuel-reaches-cylinder-chambers}) \\
(\text{water-in-gas-mixture} \\
\text{... context fuel-reaches-cylinder-chambers}))
\]

The context is confirmed, and the system follows the link to water-in-gas-mixture, which - it turns out - has a caused-by link to moisture-in-gas-tank, which then is suggested as the modified solution.

6.3.4. Selecting the best solution

The third, and final step of the solution generation process is to select the best solution candidate. As explained in the previous section, this process may be interleaved with the GENERATE process.

When this stage has been reached, substantial effort has been put into understanding the problem, and in generating a plausible set of solutions. The strategy of selecting the best solution is comparatively simple. The following steps reflect the evaluation criteria, in descending order of importance:

SELECT
S1 Assess each candidate’s explanatory support
S2 Assess the predictivity of relevant features for each remaining candidate
S3 Assess how well the remaining candidates fulfil external requirements
**S1. Assess each candidate’s explanatory support**

The primary criterion will favor a solution which has a strong explanation within the domain model. The calculated, numeric values of explanatory strength of each candidate hypothesis are compared, and the strongest one selected. If two or more candidates have explanation strengths which are identical or do not differ significantly\(^1\), the next step is triggered.

**S2. Assess the predictivity of relevant features for each remaining candidate**

The predictive strength of a feature with respect to a previous case (and, hence, a solution) is involved in the computation of a combined reminding from the set of relevant features to a case. This criterion has therefore implicitly been used for case retrieval, but it is considered important enough to be re-used in the discrimination of candidate solutions which have equal explanatory support.

**S3. Assess how well the remaining candidates fulfil external requirements**

The third criteria uses the external constraints - given as part of the problem description - to discriminate better solutions from less appropriate ones. Some of these criteria may have been used earlier in the process, but this part of the selection step ensures that all relevant criteria are taken into account.

If it turns out that Creek is unable to discriminate between two or more solution hypotheses, they are presented to the user for final selection.

### 6.4. The Learning Process

The primary machine learning method in Creek is the learning of problem specific knowledge by retaining experience in the form of cases. However, a Creek system is also able to take part in the refinement of its general knowledge model. This ability is due to its knowledge-intensive methods for generating and evaluating explanations, which also has the side-effect of checking coherence and integrity in the knowledge-base, as well as suggesting possible ways to resolve conflicts. Unresolvable contradictions discovered during problem solving or case learning is reported to the user, who decides proper actions to take. This refinement process is basically a manual process, however, while the research reported here focuses methods for sustained learning in which the computer plays a much more active part. This subchapter is therefore devoted to the case learning method in Creek.

---

\(^1\)The actual criteria - i.e. the degree of difference of explanation strength - for when to apply the 'predictivity measure' has to be tested and refined during implementation.
6.4.1. The Learning Algorithm

A Creek system learns by retaining cases, either representing single problem solving experiences or merged into slightly generalized cases. The criteria for when to store a new case, when to modify an existing case and when to update indices only, are as follows:

A new case is created either if no similar past case was found, if a past solution needed modification, or if the new case has a set of relevant findings significantly different from those of the matching case. ‘Significantly different’ in this context means that the two sets differ in the names of the findings, or that the different values of the corresponding finding names are not generalizable. If both cases have the same findings names, and their values are close enough to be generalizable, the two cases are merged. If their values are close enough to be regarded equal, the only updating made is adding of ‘book-keeping’ information to the case and adjustment of the relevance factors: If a past case leads to a successful solution to the new problem, this leads to a strengthening of the relevance factor for each finding. If a past case leads to an unsuccessful solution, an attempt is made to identify one or more findings responsible for the mismatch, leading to a weakening of their relevance factors.

As the system gains experience, the frequency of problem cases which lead to storing of a new case will gradually decrease, implying a flattening of the knowledge base growth curve over time.

The Creek learning algorithm is shown in figure 6.11. Only higher level processes and branching points are shown, in order to capture the main principles of the algorithm. The conditions and actions for modifying case descriptions and relevance factors of findings (referred to as ‘remindings’ in the figure) correspond to the description given above. When referring to the EXTRACT-CONSTRUCT-STORE model of sustained learning (chapter 3.5.7), the condition boxes to the upper left are part of the EXTRACT process, the ellipses in the middle of the figure - representing creation or modification of cases - are part of the CONSTRUCT process, while the STORE step is represented by the bottom level (elevated) process and the modifications of remindings.

The process named rerun the problem in the figure, represents Creek’s attempt to evaluate the operationality of a modified case structure: Before a modified case is permanently stored, the new case memory is automatically tested by re-running the problem. If the new or modified case is retrieved as the best match to the problem, the test is successful; if not, the remindings are adjusted until the case is retrieved.
The learning described so far is based on whether the user regards the suggested problem solution acceptable (the first condition of the algorithm). An acceptable solution may be a solution which seems reasonable and which the user therefore wants to apply to the actual problem, or it may be a solution which has been tested on the real application with a positive result. As previously mentioned, it may take some time to test out the effects of a suggested solution.

Figure 6.11: Learning in Creek
The figure shows the main steps of the learning algorithm, grouped into the phases of EXTRACT, CONSTRUCT, and STORE. Rounded boxes are processes while the eight-sided polygons are branching points.
solution (a particular diagnosis or treatment, say), but a case-based reasoner should be able to learn from a problem solving experience even if the suggested solution has not yet been confirmed by real world testing. The learning algorithm of Creek, as described here, regards a solution accepted by the user as a confirmed solution to be stored in a case and reused in future problem solving. The confirmation status is held in a slot and updated as soon as the result of applying the solution is reported back to the system. When the problem solver tries to retrieve a case that matches a new problem, it will retrieve a non-confirmed case only if there are no confirmed cases with strength above the reminding threshold.

If a solution turns out not give the expected results when tested on the real application, the system (and the user) should learn from the mistake. In Creek this is done by entering a dialogue with the user in order to identify and explain the mistake. The mistake may have been caused by errors or incompleteness in the general knowledge model (leading to erroneous explanations), or by errors in indices or relevance factors. Possible weaknesses in the general knowledge model need to be fixed first. A Creek system will be able to act as a 'discussion partner' in such an error recovery process, since it can perform coherence checks on the general knowledge, provide examples from the case base, etc. The subsequent updating of the case memory is performed according to the learning algorithm. An attempt is made to solve the problem again. The only difference is that the criterion for an acceptance of a solution is clearly defined this time.

6.4.2. Indexing of cases

The case indices provides the 'hooks' that connect the case structures to the general knowledge model, as well as the links that relates cases to one another. As described in chapter 5.2.5.2, a Creek case contains the following information (grouped according to type):

<table>
<thead>
<tr>
<th>Relevant findings</th>
<th>Successful diagnosis</th>
<th>Expl. of successful diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Differential features</td>
<td>Successful treatment</td>
<td>Expl. of successful treatment</td>
</tr>
<tr>
<td>Differential cases</td>
<td>Failed diagnosis</td>
<td>Expl. of failed diagnosis</td>
</tr>
<tr>
<td>Prototypicality of case</td>
<td>Failed treatment</td>
<td>Expl. of failed treatment</td>
</tr>
</tbody>
</table>

In addition, a case also contains 'book-keeping' information like its status in the problem solving and learning process, time reference, the number of times it has been used to solve new problems, etc.

---

1This assumes that the actual problem was solved manually. If not, the system will use its information about the failed solution to exclude that case, and otherwise attempt to solve the problem in the regular way.

2Note that, in addition their role as pointers to cases for the purpose of problem solving from matching cases, indices may also be used to generate example cases, for example during manual (or semi-automated) refinement of the knowledge base.
A case is indexed by:

- **Relevant findings.**
- **Differential findings.**
- **Faults and treatments:**

When storing a case, the relevant findings are evaluated in order to determine their importance and predictive strength with respect to the case's solution. These values are stored with the finding in the 'has-relevant-finding' slot of the case. The numerical relevance factor of a finding, computed from the predictive strength and importance, is stored within the case index (see chapter 5.2.5.2). The predictive strength and importance are derived on the basis of:

- The strength, \( S_{\text{expl}}(\text{Finding}) \) of the explanation which was generated in order to justify the finding as relevant (see chapter 6.3.2, description of task U3).
- The finding's importance, \( I_{\text{oldc}}(\text{Finding}) \), and predictive strength, \( P_{\text{oldc}}(\text{Finding}) \), in the previous case (if such a case exists).
- The number of other cases pointed to by the finding, \( N_{\text{cases}}(\text{Finding}) \).
- The number of other findings pointing to the solution of the case, i.e. pointing to both the diagnosis and the treatment of the case, \( N_{\text{finds}}(\text{Solution}) \).
- The strength of the explanation generated to justify the plausibility of the solution, \( S_{\text{expl}}(\text{Solution}) \).
- The strength of the explanation generated to justify the plausibility of the solution *when a particular finding has been removed*, \( S_{\text{expl-finding}}(\text{Solution}) \).

(The subscript "expl-finding" should be read "explanation minus finding").

The suggested predictivity strength is determined by the following simple algorithm:

\[
\text{IF solution of new case = solution of old case} \\
\quad \text{IF } P_{\text{oldc}}(\text{Finding}) = \text{sufficient} \\
\quad \quad \quad \text{SET } P_{\text{newc}}(\text{Finding}) = \text{strongly-indicative} \\
\quad \quad \quad \text{SET } P_{\text{oldc}}(\text{Finding}) = \text{strongly-indicative} \\
\quad \text{ELSE} \\
\quad \quad \quad \text{SET } P_{\text{newc}}(\text{Finding}) = P_{\text{oldc}}(\text{Finding}) \\
\text{ELSE} \\
\quad \text{IF no existing case-pointers from finding} \\
\quad \quad \text{SET } P_{\text{newc}}(\text{Finding}) = \text{sufficient} \\
\quad \text{ELSE} \\
\quad \quad \text{BRANCH} \left[ \frac{N_{\text{case}}(\text{Finding})}{N_{\text{cases}}(\text{All-findings})} \right] \\
\quad \quad \quad < k_1 \quad \text{--> SET } P_{\text{newc}}(\text{Finding}) = \text{strongly-indicative} \\
\quad \quad \quad >= k_1, < k_2 \quad \text{--> SET } P_{\text{newc}}(\text{Finding}) = \text{indicative} \\
\quad \quad \quad >= k_2 \quad \text{--> SET } P_{\text{newc}}(\text{Finding}) = \text{spurious} \\
\quad \text{IF } S_{\text{expl}}(\text{Finding}) > k_9 \\
\quad \quad \text{BRANCH} \left[ P_{\text{newc}}(\text{Finding}) \right] \\
\quad \quad \quad = \text{spurious} \quad \text{--> SET } P_{\text{newc}}(\text{Finding}) = \text{indicative} \\
\quad \quad \quad = \text{indicative} \quad \text{--> SET } P_{\text{newc}}(\text{Finding}) = \text{strongly-indicative}
\]
The algorithm first checks whether the solution of the two cases are the same. If so, the predictive strength is copied, but a sufficient feature for the old case is reduced to a strongly indicative one before the copying takes place. If the solution was modified, the predictivity of the old case cannot be used to compute the value for the new case. A feature with no previous case pointers is regarded as a sufficient feature for retrieving the new case. If there are existing case pointers, the ratio of cases pointed to vs. the total number of cases is calculated. Each predictive strength value is assigned a threshold value ($k_1$, $k_2$) - a set of initial threshold values may be, e.g., $k_1 = 0.01$, $k_2 = 0.1$). The resulting predictive strength of the feature is determined by relating the case ratio to these threshold values. This is expressed in the algorithm by testing the ratio $N_{\text{case(Finding)}} / N_{\text{cases(All-findings)}}$ against the threshold values. Finally, an attempt is made to strengthen the weakest values by examining the explanation which was generated when the relevance of the feature was justified. If the explanation strength is above a certain threshold value ($k_9 = 0.95$, say), a spurious or indicative predictivity strength is replaced by an indicative and strongly indicative strength, respectively.

The algorithm for computing the importance of a finding is somewhat different, since a more knowledge-intensive method is needed to get even a slight estimate of the importance of a finding to a new solution. If a solution concept is linked to the finding via a strong causal or premiss-setting relation, such as always-caused-by, or always-requires, the importance value is set to necessary. Otherwise, the suggested method is based on removing the finding from the explanation structure previously generated in order to justify the solution. A new explanation for the plausibility of the solution is constructed, and its strength is evaluated. A large reduction of the explanatory strength after removing of a finding indicates that this finding is an important one. This criterion is reflected by the branching test in the last part of the algorithm:

```plaintext
IF solution of new case = solution of old case
  SET \text{I}_\text{newc(Finding)} = \text{I}_\text{oldc(Finding)}
ELSE
  IF no other findings point to the new case
  OR
    if finding is linked to case via premiss or strong causal relation
    SET \text{I}_\text{newc(Finding)} = \text{necessary}
  ELSE
    BRANCH \left[ \text{Sexpl(Solution)} - \text{Sexpl-finding(Solution)} \right]
    \begin{align*}
      \geq k_{11} & \quad \rightarrow \quad \text{SET } \text{I}_\text{newc(Finding)} = \text{characteristic} \\
      < k_{11} \quad & \quad \rightarrow \quad \text{SET } \text{I}_\text{newc(Finding)} = \text{informative} \\
      < k_{12} & \quad \rightarrow \quad \text{SET } \text{I}_\text{newc(Finding)} = \text{irrelevant}
    \end{align*}
```

1The k-thresholds are parameters, to be tuned during an actual implementation.
Suggested initial values for the k-factors are: \( k_{11} = 0.1 \), \( k_{12} = 0.01 \).

To summarize the case learning process in Creek, the main steps are shown in figure 7.3., with reference to the generic EXTRACT-CONSTRUCT-STORE model. As indicated in the figure, the Extract process will necessarily extract different items from a problem solving process involving case-based reasoning than it would if rule-based or model-based reasoning has been used. The subsequent process description assumes that the problem has been solved by use of a previous case. The process steps shown are exemplified in subchapter 7.5.2.

![Figure 6.12: Structure of the Learning Process in Creek](image)

The major steps of case-based learning in Creek, according to the EXTRACT-CONSTRUCT-STORE model. The user may assist in the construction of a new case, and the indexing and evaluation processes.

6.4.3. Case generalization

It is not a goal of Creek’s learning method to perform case generalization whenever a possible opportunity exists. Creek is a 'lazy generalizer' where the retention of past cases - not induction - is the primary learning paradigm. The purpose of the case generalization method in Creek is partly to improve the matching process, partly to reduce the growth of the case base. Cases are only generalized if they match exactly on feature names, and where some feature values have common parents in an inheritance hierarchy. The only relation used for generalization is the \emph{subclass-of} relation. For a generalized value to be acceptable, it may not have a specialization that is a feature value of another case which is on the current case’s difference list. The specific
values may be retained - in a particular facet of the finding slot - in case the generalized value later turns out to be an over-generalization. The specific values may later be removed, for example after the generalized case has been successfully used to solve a certain number of problems. Criteria for when to keep and when to remove such specific values needs to be defined for a given type of application.

6.4.4. Acquiring general domain knowledge

General domain knowledge is learned in a Creek system as a consequence of incoherences discovered in the knowledge base, of unsatisfactory explanations being generated, or unknown terms being entered into the system. The user/expert is always the active part when new terms shall be defined or modifications to the general knowledge model shall be made. Creek can support the acquisition of new concepts or modifications of existing structures by checking consequences and generating expectations from the updates made. Creek's case base may be utilized in the testing of extensions and updates to the knowledge base. The Creek representation system is coupled to a graphical knowledge editor, which enables visualization and graphical editing of concepts and relations, as described in chapter 5.2.6.
Chapter 7

Mud Creek - An example in the domain of oil well drilling

This chapter illustrates the functionality of CREEK, by describing an example of a system design for diagnosis and treatment of faults during oil well drilling. The example application addresses the specific problems of identifying erroneous properties or constituents of the drilling fluid, and suggesting repair actions. After a brief overview of the problem domain, the structure of the knowledge base is described and exemplified. This is followed by an example session describing characteristic properties of a running system. The emphasis is on demonstrating the knowledge-intensive case-based method of CREEK. The subtasks focused are case retrieval, case matching, solution transfer, and learning of a new case.

7.1. Overview of the problem

While drilling for oil and gas, a liquid is pumped down the drill string and upwards through the annular space between the drill string and the wall of the borehole. This fluid - the drilling fluid, also called mud - serves several purposes: Cleaning of the borehole by removing cuttings and cavings, control of the pressure in the well, lubricating and cooling the drill string and bit, maintaining mechanical stability of the drilling process, bringing constituents of the rock formation to the surface for analysis. Mud engineering is a separate discipline of well drilling and are most often handled by special mud companies. A mud engineer is continually supervising the mud properties during drilling, adjusting the properties mechanically (e.g. filtering) and chemically (e.g. adding chemicals) if necessary. Maintaining the right pressure is particularly important since inflow of fluid from the surrounding formation or loss of mud to the formation always is a danger when drilling into a new formation layer.

Diagnosis and subsequent treatment of mud is a complicated classification problem from an AI point of view. There are four main reasons for this:
First, diagnosis and treatment are not two separate problems, the latter succeeding the former. The process is rather a merging of partial diagnosis and treatment activities. Typically, some initial observations lead to formation of a diagnostic hypothesis (best guess) for which a treatment is initialized. During treatment more observations and measurements are gathered, possibly leading to a refusal (or specialization) of the current hypothesis and to the formation of a new one. Since the only way to gather more information about the problem is to start treatment (add chemical additives) based on an initial hypothesis, this treatment must be carefully done in order not to create dangerous situations if the hypothesis should fail. This means that unless the diagnostic hypothesis is pretty certain, other possible hypotheses must also be taken into consideration when a treatment is applied.

Second, the mud itself is a complex liquid containing roughly 20 important constituents. They give the mud its properties described by approx. 40 observable or measurable descriptors. Some of these descriptors are simple property-value pairs (e.g. density 1.4), while others are complex structures (e.g. filter cake description). There are about 20 or so significant single causes for non-normal mud properties (different types of contaminants, formation fluid influx, etc.), and a lot of relevant combinations. A mud engineer supervising the process typically monitors 10-12 properties continually or periodically during normal operation.

Third, the values of mud properties are (of course) highly interdependent. Their normal ranges also depend on variables like the depth of drilling, and properties of the surrounding geological formation - including local geology at the drilling site. A knowledge model describing the relations between normal and abnormal values of single mud properties, groups of mud properties, properties of the formation, and causes and consequences of failure situations, is time-consuming and difficult to acquire.

Finally, a problem may have more than one cause, i.e. multiple faults may occur simultaneously. Diagnostic problem solving in AI has primarily focused on single fault problems. Multiple faults introduces a level of complexity difficult to handle with traditional methods, which reason from generalized knowledge only. The CREEK approach, on the other hand, relates co-occurring faults to each other by storing them together in the same problem case.

The emphasis on reusing previous specific cases (or only slightly generalized cases), guided by knowledge-intensive procedures for case matching and solution transfer, provides a proposed method for dealing with the type of real world applications described here. Due to the complexity of the problem, it is not possible to build a complete domain model of the problem. In order to continuously adapt and improve the system, proper routines and methods for
updating and refining the knowledge base during regular system operation is needed. In addition to manual methods, techniques which enable the system itself to automatically learn from its own problem solving activities seems crucial to maintaining its successful operation.

7.2. Designing a Creek Application System

In order for an implemented Creek system to behave according to its intentions, different types of control rules and parameters need to be defined, as described in the previous chapters. Below, a stepwise procedure is proposed, which ensures that all the necessary pieces of knowledge and information get defined. It does not suggest how to elicit and construct the knowledge models, it only points out what type of knowledge and control parameters are needed:

1. Top level architecture
   - Identify the roles of the Conceptual Knowledge Fundament, the Object Knowledge Model and the Diagnosis and Repair Model.
   - Design initial submodels of general problem concepts (cf. figure 6.2: "Diagnosis and repair concepts"), the problem solving process (cf. figure 6.3: "A top level, generic procedure for diagnosis and repair"), and problem descriptors, solutions and intermediate states (cf. figure 6.7: "Example hierarchies of findings, faults, and treatments of the car-starting domain").

2. The Object Knowledge Model
   - Determine the level of the object knowledge depth suitable for the application.
   - Extend the model of problem findings, solutions and intermediate states with concepts and relationships that justify (explain) them.
   - Identify the domain-dependent relations, in addition to the relation set defined by the ISOPOD knowledge base (figure 5.9).
   - Determine which type of relationships to define explicitly (as single concepts), and which to express as concept.relation.value triplets.
   - Determine whether the existing inference methods are sufficient. Add rules for plausible inheritance (e.g. general inheritance rules, feature derivation rules).

3. The Combined Reasoning Model
   - Determine the set of relations to use for spreading activation.
   - Determine the initial value of the reminding threshold, and the threshold values related to the matching of cases, to estimation of predictive strength and importance of findings, and to the merging of cases.
• Determine the sets of explanation relations (chapter 6.2.1).
• Determine the set of explanation evaluation rules.

4. The Diagnosis and Repair Model
• Identify the different tasks (cf. figure 6.4: "Diagnostic tasks").
• Determine the degree and extent of explicit representation of control knowledge related to the tasks.

5. The Sustained Learning Model
• Determine the domain-dependent modification rules.
• Determine a manual procedure to handle system questions which the user is unable to answer.

The next section illustrates parts of the domain knowledge, using the CreekL frame language. The knowledge describes part of the envisioned system, showing pieces of the general object level knowledge and a previous problem solving case, as well as control level concepts.

7.3. The knowledge base - Object level

A conceptual/relational domain model describes what drilling mud is, its functionality, components, mud properties and effects of mud additives. All relevant inter-relationships between concepts - like hierarchical structures, functional relations, causalities, constraints, contextual dependencies - are explicitly described.

The purpose of this model is to give the expert system as deep and wide an understanding of the domain as possible. The model constitutes the knowledge fundament for the whole system. It will be the basis for plausible reasoning, generation of explanations, error-checking of input data, generation of expectations from input data, etc.

Parts of example frames defining geological concepts are shown below. (The mud system being described is a water-based, gel/lignosulphonate system):

```
FRAME rock-salt
  subclass-of value contaminant substance
  part-of value rock-formation
  has-state-change value-set increased-rock-salt decreased-rock-salt unchanged-rock-salt

FRAME increased-rock-salt
  always-causes value increased-chloride-content
  value-class state-change
```

\(^{1}\)The model is nevertheless a simplified one, of course.
always-causes value decreased-pH
value-class state-change

sometimes-causes value increased-funnel-viscosity
increased-yield-point
increased-gel-strength

does-not-change value mud-volume
value-class mud-property

FRAME salt-water
subclass-of value contaminant water mud
part-of value natural-environment
main-substance-of value sea-water
has-state-change value-set increased-salt-water decreased-salt-water
unchanged-salt-water

FRAME increased-salt-water
subclass-of value state-change intermediate-state
always-causes value increased-chloride
increased-volume
value-class state-change
always-causes value decreased-ph
decreased-density
decreased-solids-content
value-class state-change
sometimes-causes value increased-funnel-viscosity
increased-yield-point
increased-gel-strength
value-class state-change

While rock salt is regarded as a contaminant of the mud, salt water may be the liquid basis of
the mud as well as a contaminant. The domain also involves definition of processes, for
e.g.: A particularly important type of concept related to findings are those describing qualitative
value changes with respect to a set of reference values. The reference values may be a set of
predefined 'normal' values, or a set of values measured at a particular point of time. Typically,
measurements are taken at various points of time during drilling, and the reference set for state
to enable
decisions based on comparing a sequence of previous tests).

FRAME salt-water-influx
subclass-of value liquid-flow natural-process water-into-mud
has-effective-component value salt-water
has-subclass value excessive-salt-water-influx

FRAME decreased-ph
subclass-of value state-change finding
state-change-of value ph
measured-by value ph-test time-stamp
occurs-when value (and (< ph.has-measured-value
ph.has-reference-value))
caused-by value salt-water rock-salt lignite
lignosulphonate gypsum
implies value decreased-total-hardness
relevant-feature-in value (case-055 0.8) (case-068 0.95)
The 'relevant-feature-in' slot is an index to the cases for which the frame concept is a relevant feature. Each case pointer is associated with the relevance factor of the feature - case pair. This factor expresses how relevant the feature is to the particular case pointed to. Decreased pH, in the example above, is a more relevant finding for case-068 than it is for case-055\(^1\). The definition of what pH is, is shown below:

```plaintext
FRAME ph
  subclass-of value chemical-measure
  has-state-change value-set decreased-ph increased-ph unchanged-ph
  measured-by value ph-test
  has-measured-value value-dimension none
  normal--when value (and (>= self.has-measured-value 9.8)
                   (<= self.has-measured-value 10.2))
  high--when value (> self.has-measured-value 10.2)
  low--when value (< self.has-measured-value 9.8)
  has-reference-value if-needed (f-get input-value-norm.has-ph)
  measure-of value acidity
```

Both diagnosis and treatment of the mud is based on how much a set of measured parameters (i.e. findings) deviates from their normal values. The set of normal values is determined by the type of mud being used and other drilling conditions. They are defined by the mud engineering team, and kept in a separate frame:

```plaintext
FRAME input-value-norm
  instance-of value gold-standard
  has-funnel-viscosity value 48
    value-dimension sec
  has-plastic-viscosity value 20
    value-dimension cp
  has-yield-point value 10
    value-dimension lb/100 ft\(^2\)
  has-gel-strength value 5/14
    value-dimension lb/100 ft\(^2\)
  has-fluid-loss-api value 5
    value-dimension ml
  has-fluid-loss-hpht value 12
    value-dimension ml
  has-chloride value 15000
    value-dimension mg/l
  has-solids-content value 14
    value-dimension percent
  has-cation-exch-capacity value 25
    value-dimension meq/ml
  has-ph value 10
    value-dimension none
  has-alcalinity value 1.3/3.2
    value-dimension none
  has-calcium value 40
    value-dimension mg/l
  has-sand value 0.25
    value-dimension weight-%
  has-density value 1.35
    value-dimension g/cm\(^3\)
```

\(^1\)Relevance factors, and how their values are derived, is described in chapter 5.2.5.2.
Relations not defined in ISOPOD need to be explicitly defined in their own frames. For example:

```plaintext
FRAME does-not-change
  instance-of value modifying-relation
  used-to-describe value finding state-change treatment
  has-entry-type value finding physical-state
  evaluation-depends-on value entry-1 entry-2
  occurs-when value (cond ( (and (= entry-2 entry-1) (> (time-stamp entry-2) (time-stamp entry-1))))
```

In addition to conceptual definitions, as illustrated above, general knowledge also include heuristic rules. They may express rules-of-thumb kind of experiential knowledge, i.e. surface level knowledge not captured by the deeper domain model. It may also be knowledge that exists at a deeper level, but is compiled ('cached') to a surface level in order to improve performance.

```plaintext
FRAME salt-water-influx-rule
  instance-of value domain-rule rule-of-thumb
  has-antecedent value ph.has-state-change 'decreased-pH
  has-consequent value diagnostic-hypothesis.has-instance.salt-water-influx
```

The object level knowledge also include the cases. An example case is shown below (the annotation fields are left out):

```plaintext
FRAME case-068
  instance-of value solved-case
  has-process-status value diagnosis-confirmed-by-user
  treatment-confirmed-by-testing
  has-input-time value 11/10/88 18:20
  has-relevant-finding value (increased-funnel-viscosity 62 48)
  (increased-yield-point 18 10)
  (increased-plastic-viscosity 23 20)
  (increased-chloride 25000 15000)
  (increased-gel-strength 12/32 5/14)
  (decreased-ph 9 10)
  (increased-tank-volume)
  (increased-calcium 60 40)
  has-successful-diagnosis value (salt-water-influx
  (0.95
  (increased-yield-point.caused-by.chemical-reaction
  chemical-reaction.implied-by.increased-chloride
  increased-chloride.implies.increased-salt-content
  increased-salt-content.result-of.salt-water-influx)
  (increased-tank-volume.caused-by.fluid-influx
  fluid-influx.has-subclass.salt-water-influx)))
  has-contaminant value salt-water
  has-successful-additive value (caustic-soda lignosulphonate
  (1.0
  (goal-state.has-instance.increased-ph
  increased-ph.caused-by.caustic-soda)
  (goal-state.has-instance.decreased-viscosity
  decreased-viscosity.caused-by.lignosulphonate))))
```

```plaintext
same-diagnosis-in value case-004 case-279 case-115 case-051
same-treatment-in value case-095 case-101 case-005 case-131
```
The list of relevant findings contains a finding’s qualitative value (simplified, see next paragraph), the observed value, and the reference value. This enables a user to go back to the numerical source of the qualitative values.

Note that a simplification has been made in the examples above: The rate of change, e.g. whether a value is weakly-increased, moderately-increased or strongly-increased is highly significant for the consequence of a set of changes. As suggested at the end of chapter 5.2.5.1, the degree of decrease or increase of a measured value is not represented as a separate concept (e.g., strongly-increased-ph) in this knowledge model, but kept in the miscellaneous field of a value expression under the key rate. A moderate change is the default, and the rate annotation is used only for strong and weak values. This field is omitted, for clarity, in the sample concepts shown. Their use will be illustrated in the example session later in this chapter.

7.4. The Knowledge Base - Control Level

A task model of mud diagnosis and treatment (corresponding to the Diagnosis and Repair Model) guides the problem solving process from the indication of an abnormal situation through fault finding and treatment. This model defines control level concepts (observation, measurement, diagnosis, hypothesis, evidence, cause, etc.), and a hierarchy of actions (collect-more-data, apply-non-risky-additive, etc.) controlled by an overall strategy for mud diagnosis&treatment (when to do what, what to look for, etc.). Example frames:

```
FRAME strategy-rule  
  subclass-of value    heuristic-rule  
  has-instance value    stepwise-interpretation-rule check-expectations-rule

FRAME stepwise-interpretation-rule  
  instance-of value    strategy-rule  
  has-antecedent value    (and    
                           possibly-abnormal-state    
                           (not all-data-available))  
  has-consequent value    stepwise-interpretation

FRAME stepwise-interpretation  
  subclass-of value    strategy  
  action-tree value    
                       ((test-for-viscosity-change)  
                       ((measure-yield-point)  
                       (measure-plastic-viscosity) )  
                       ((test-for-chemical-reaction)  
                       (measure-ph)  
                       (check-for-unbalance)  
                       (check-for-contamination)  
                       ((test-for-solids-contamination)  
                       (measure-solids-contents)  
                       (measure-particle-size)  
                       (measure-solids-volume)  
                       (measure-mud-density) )
```
The control knowledge frames illustrate the explicit definition of a problem solving strategy within this domain. There are two types of strategies, one to perform a stepwise interpretation of certain important data, the other to derive and check expected values from existing findings and hypotheses. The strategy for stepwise interpretation is first to look for general changes in viscosity, either by direct observation or by measuring the Funnel Viscosity (FV). If there are significant viscosity changes, a more detailed test is performed by measuring the Yield Point (YP) and Plastic Viscosity (PV). If YP has increased more than PV, then the cause of the viscosity change is likely to be a chemical reaction in the mud. If PV has increased more than YP, then contamination by solid material is likely to be the cause.

7.5. An Example Session

This section illustrates the main parts of the problem solving and learning cycle in Creek, using the knowledge base just described. The purpose of the example is to demonstrate how the knowledge structures and methods of the Creek architecture works together in solving a concrete problem, and learning from this experience. Since only a small subset of the methods are implemented\(^1\), the example is constructed based on the description of the Creek methods given in chapter 6, and experience in mud diagnosis gained in an earlier project [Nordbø-89]. To the extent that detailed methods are exemplified, e.g. for calculating strength values of parameters such as case remindings, the degree of matching between cases, etc. the methods shown in this example should be regarded as first approximations. The quality and effect of these detailed algorithms can only be determined through a cycle of implementation, testing, and refinement. Some of the methods will depend on the structure of the knowledge model (e.g. the granularity and explicitness of relationships, the detailed structure and growth rate of the case base, etc.). Tuning, extending and refining these methods beyond the initial versions suggested here should therefore be regarded as part of the application development process.

\(^1\)The knowledge representation system, its basic inference processes, and the main parts of the spreading activation method have been implemented (see chapter 5.2.6).
7.5.1. Problem solving

To increase readability, a combination of textual and graphical forms is used in this subchapter to describe problems, cases and general object level knowledge.

The problem: An abnormal change in drilling fluid viscosity has been observed by routine check. The set of findings which describe the problem case - Input-case-1 - is listed below.

The problem solving steps are described in detail in chapter 6.3. This example illustrates the operation of the problem solving and reasoning algorithms, with focus on the case-based reasoning process.

7.5.1.1. Understanding the problem

The system reads the problem description, with the accompanied goals of suggesting a diagnosis and a treatment. First, the input values are transformed into qualitative values, expressing a decrease, increase or unchanged value with respect to the reference set (a set of normal input values).

Input-case-1

<table>
<thead>
<tr>
<th>Finding name</th>
<th>Reference value</th>
<th>Measured value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density</td>
<td>1.82</td>
<td>1.82</td>
</tr>
<tr>
<td>Apparent Viscosity</td>
<td>38</td>
<td>32</td>
</tr>
<tr>
<td>Funnel Viscosity</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Plastic Viscosity</td>
<td>34</td>
<td>30</td>
</tr>
<tr>
<td>Yield Point</td>
<td>14</td>
<td>11</td>
</tr>
<tr>
<td>Gel Strength</td>
<td>6/14</td>
<td>5/14</td>
</tr>
<tr>
<td>Fluid Loss API</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Fluid Loss HPHT</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>Chloride</td>
<td>18000</td>
<td>18000</td>
</tr>
<tr>
<td>Solids</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>CationExchange Capacity</td>
<td>22</td>
<td>18</td>
</tr>
<tr>
<td>pH</td>
<td>10.2</td>
<td>10.3</td>
</tr>
<tr>
<td>Alcalinity</td>
<td>1.4/3.0</td>
<td>1.5/3.1</td>
</tr>
<tr>
<td>Calcium</td>
<td>380</td>
<td>370</td>
</tr>
<tr>
<td>Watercontent</td>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>Oilcontent</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

This transformation process may be viewed as the initializing spreading step, activating a set of inferred findings. This set of findings constitute the starting-points for the main spreading
activation process\(^1\). The set of spreading relations are defined by the parameters *goal-spreading-relations* and *finding-spreading-relations*, where the first set includes relations such as *has-subclass*, *has-part*, *causes*, *has-effective-component*, etc., and the relations that spread out from findings include *subclass-of*, *part-of*, *caused-by*, *effective-component-of*, *measure-of*, *measured-by*, *occurs-when*, *state-change-of*. Figure 7.1 gives a view into the system’s model of general domain knowledge, and illustrates partial concept structures of goals and findings, and their their interrelations.

![Diagram of Mud Creek's Model of Generalized Domain Knowledge](image)

**Figure 7.1: A View into Mud Creek's Model of Generalized Domain Knowledge**

The figure illustrates a small part of the search space delimited by the problem findings and goal states. Only a few of the relationships are named, since the purpose of the figure is to illustrate the basic structure of the 'knowledge network', not the details. The vertical, dashed lines indicates the part of the network (small or large) that connects the upper and lower structures.

The objective of the spreading activation process is to determine the problem solving context by spreading out from the problem findings as well as the goal concepts of the problem solving process - as illustrated in the figure. The spreading process is constrained by the subsets of

\(^1\)Activation of a finding is marked by adding the following slot to its frame definition:

```
<finding concept>
  has-activation-state  value  activated
```
relations along which the spreading will take place, as defined in the *goal-spreading-relations* and *finding-spreading-relations* lists. As described in chapter 6.3.2, the spreading activation process results in an activation of the set of concepts that are linked to both a finding and a goal concept via the relations specified in the lists just mentioned. The activated knowledge structure provides a focused environment (context) for later generation and evaluation of explanations, as well as other subprocesses of problem solving.

This completes the initial step of problem understanding (U1). In our example case, no transformed finding is removed by the spreading process, i.e. none of the input findings are considered totally irrelevant.

The next step (U2) refines the hierarchies of activated goals and treatments. No goal concepts are removed, since no explicit goal constraints has been given. An attempt is then made (U3) to infer other relevant findings from the (transformed) input findings, within the context defined by the activated concepts. The following two feature derivation rules are triggered:

```
FRAME derivation-rule-013
  instance-of    value    feature-derivation-rule
  has-antecedent value    (and decreased-plastic-viscosity
                          .has-activation-status.activated
                          decreased-yield-point
                          .has-activation-status.activated)
  has-consequent value    decreased-funnel-viscosity
                          .has-activation-status.activated
```

Since the rates of changes are not mentioned in the rule, the weakest of the change rates for the 'source' findings - i.e. decreased plastic viscosity and decreased yield point - is assumed for the change rate of the derived finding. That is, the decreased funnel viscosity gets the change rate moderate.

```
FRAME derivation-rule-021
  instance-of    value    feature-derivation-rule
  has-antecedent value    (and fluid-loss.has-state-change.
                          .increased-fluid-loss-api
                          fluid-loss.has-state-change.
                          .increased-fluid-loss-hpht)
  has-consequent value    increased-fluid-loss
                          .has-activation-status.activated
```

The first rule infers the feature decreased-funnel-viscosity from the facts that the apparent viscosity, plastic-viscosity and yield point has decreased. In the second rule, the two different measures of fluid loss are merged into one finding. The resulting set of relevant findings, with samples of previous case pointers, is shown below. Each case pointer consists of a case name and a relevance factor (Rf) expressing the relevance of the particular finding for the particular
case. A feature representing a state change also contains the rate, or degree, of change for the input case (rate moderate being the default):

<table>
<thead>
<tr>
<th>Feature</th>
<th>Case-1</th>
<th>Case-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>unchanged-density</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decreased-apparent-viscosity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decreased-plastic-viscosity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decreased-yield-point</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decreased-gel-strength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>increased-fluid-loss</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unchanged-chloride</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unchanged-solids</td>
<td></td>
<td></td>
</tr>
<tr>
<td>increased-ph</td>
<td></td>
<td></td>
</tr>
<tr>
<td>increased-alkalinity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decreased-calcium</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decreased-funnel-viscosity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decreased-cec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unchanged-water-content</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unchanged-oil-content</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This constitutes the candidate set of relevant findings. The next step (within U3) is to justify that all these findings are relevant within the current problem context. Remember that the findings have been selected or inferred by a rather weak method, which is why supporting explanations are generated to justify them. An explanation evaluates the chain of relationships linking one or more features to the goal hierarchies, and its strength is determined. In our example, the explanation process removes the decreased-apparent-viscosity. This is done because there is an explanation evaluation rule saying that an apparent viscosity finding should be removed (i.e., regarded as superfluous) if at least two other viscosity measures exist, and their changes are in the same direction as the apparent viscosity change. In our case, the decreased plastic-viscosity and the decreased yield-point leads to exclusion of the decreased apparent viscosity. This completes the Understanding phase of the problem solving process.

7.5.1.2. Generating a set of plausible solutions

The next main phase contains the generation of a set of candidate faults. The relevant features are combined into case remindings, based on their relevance factors and pointers to common cases (as described in section 6.3.3, G1).

First, it is checked whether a finding has the relevance factor 1.0 for a case. This test fails, and the number of findings pointing to common cases are identified. As seen from the above list (when decreased-apparent-viscosity is excluded), there are 5 findings pointing to case-321. At least 1/3 of the relevant findings must point to a case for the computation of a reminding strength to be made. Since there are 14 relevant findings, 5 pointers to the same case is a

1 An explanation process is illustrated later in this example.
sufficient number of pointers. No other cases are being pointed to by a sufficient number of findings. The combined relevance factor for the findings with respect to case-321 is 0.74, and the threshold value for continuing the matching process has been set to 0.7. Thus, a combined reminding strength is calculated as $\frac{5}{14} \times 0.74 = 0.26$, which is rather weak. Nevertheless, a potentially relevant case has been found, and the case matching process continues based on the following matched case:

**Case-321**

<table>
<thead>
<tr>
<th>Relevant findings</th>
</tr>
</thead>
<tbody>
<tr>
<td>weakly decreased density</td>
</tr>
<tr>
<td>weakly decreased plastic viscosity</td>
</tr>
<tr>
<td>strongly decreased yield point</td>
</tr>
<tr>
<td>decreased gel strength</td>
</tr>
<tr>
<td>strongly increased fluid loss</td>
</tr>
<tr>
<td>strongly increased chloride</td>
</tr>
<tr>
<td>weakly increased solids</td>
</tr>
<tr>
<td>weakly decreased pH</td>
</tr>
<tr>
<td>decreased alkalinity</td>
</tr>
<tr>
<td>increased calcium</td>
</tr>
<tr>
<td>decreased funnel viscosity</td>
</tr>
<tr>
<td>decreased cec</td>
</tr>
</tbody>
</table>

On the next page, the measured findings of the retrieved case - with reference values - are shown, together with the solution (fault and treatment) to the problem represented by this case. The most significant differences between the two cases are the changes in chloride, calcium and fluid loss. The qualitative findings are related as illustrated below the table of measured values. A ‘==’ link between two findings indicate a perfect featural match, a ‘++’ link indicates a weaker, but acceptable, match, a ‘..' link indicates an apparent mismatch that needs some explanation, while a ‘--’ link is a definite mismatch, needing a strong explanation. The use of rate assessments to state changes which is inherent in the knowledge model of our domain, is reflected in the way matching strengths are determined. For example, only features which match in their direction of change as well as their rate of change constitute a perfect match. Features differing in their rates in the smallest possible way (e.g. decreased-plastic-viscosity and weakly-decreased-plastic-viscosity) are regarded as acceptable - although not perfect - matches. Features with a larger 'rate distance' are regarded as apparent or definite mismatches, as shown.

Note that the important thing here is not the particular matching criteria just described. It is the ability - within the Creek architecture - to model and express the domain in a suitable way, as exemplified in our application by the change rates of problem findings, and to define a set of matching criteria appropriate for the domain model. Another domain may take another modelling approach, leading to different criteria for evaluating the degree of matching between feature pairs.
Case-321(cont.)

<table>
<thead>
<tr>
<th>Finding name</th>
<th>Reference value</th>
<th>Measured value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density</td>
<td>1.35</td>
<td>1.34</td>
</tr>
<tr>
<td>Funnel Viscosity</td>
<td>48</td>
<td>42</td>
</tr>
<tr>
<td>Plastic Viscosity</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>Yield Point</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Gel Strength</td>
<td>5/14</td>
<td>3/12</td>
</tr>
<tr>
<td>Fluid Loss API</td>
<td>5</td>
<td>18</td>
</tr>
<tr>
<td>Fluid Loss HPHT</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>Chloride</td>
<td>15000</td>
<td>25000</td>
</tr>
<tr>
<td>Solids</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>Cation Exchange Capacity</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>pH</td>
<td>10.0</td>
<td>9.8</td>
</tr>
<tr>
<td>Alkalinity</td>
<td>1.3/3.2</td>
<td>0.7/2.5</td>
</tr>
<tr>
<td>Calcium</td>
<td>40</td>
<td>80</td>
</tr>
</tbody>
</table>

Fault

Salt water influx

Treatments
Add bentonite
Add CMC

Below, the degree of match between pairs of findings is illustrated:

<table>
<thead>
<tr>
<th>Input-case-1</th>
<th>Retrieved case-321</th>
</tr>
</thead>
<tbody>
<tr>
<td>unchanged density</td>
<td>weakly decreased density</td>
</tr>
<tr>
<td>decreased plastic viscosity</td>
<td>weakly decreased plastic viscosity</td>
</tr>
<tr>
<td>strongly decreased yield point</td>
<td>strongly decreased yield point</td>
</tr>
<tr>
<td>decreased gel strength</td>
<td>decreased gel strength</td>
</tr>
<tr>
<td>weakly increased fluid loss</td>
<td>strongly increased fluid loss</td>
</tr>
<tr>
<td>unchanged chloride</td>
<td>strongly increased chloride</td>
</tr>
<tr>
<td>unchanged solids</td>
<td>weakly increased solids</td>
</tr>
<tr>
<td>weakly increased ph</td>
<td>weakly decreased ph</td>
</tr>
<tr>
<td>weakly increased alkalinity</td>
<td>decreased alkalinity</td>
</tr>
<tr>
<td>weakly decreased calcium</td>
<td>increased calcium</td>
</tr>
<tr>
<td>decreased funnel viscosity</td>
<td>decreased funnel viscosity</td>
</tr>
<tr>
<td>decreased cation exch. capacity</td>
<td>weakly decreased cation exch. capacity</td>
</tr>
<tr>
<td>unchanged-water-content</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>unchanged-oil-content</td>
<td>&lt;none&gt;</td>
</tr>
</tbody>
</table>

On the basis of the pairwise matching of findings, as illustrated above, the degree of apparent match is assessed. This is done by taking a weighted sum of the numbers of perfect and
acceptable matches (giving a perfect match a higher weight), and divide this sum by the total number of findings in the input case. Since the apparent match between the two cases turns out to be rather weak, the system looks for difference links from the retrieved case to other cases, in an attempt to retrieve a better matching case. For example, the differential links of case 321 (i.e. cases differing from case 321 in only one significant finding; empty findings - denoted <none> in the list above - are excluded) include the following finding - case pairs:

Case-321

<table>
<thead>
<tr>
<th>Difference link</th>
<th>Case-191, case-221</th>
<th>Case-112</th>
<th>Case-037</th>
</tr>
</thead>
<tbody>
<tr>
<td>unchanged-chloride</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>increased-alcalinity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>increased-water-content</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The system assesses the apparent match for the cases pointed to by difference findings found in the input case, such as unchanged-chloride and increased-alcalinity. For the example problem, however, it turns out that none of the differential cases appears to match better than case 321.

The assessment of similarity between the input case and an existing case applied so far has been referred to as an apparent match. This notion reflects that the match criteria used, to a large degree have been based on syntactical, rather than semantical, criteria. If this procedure had produced a case with a strong apparent match, this match would also have been accepted as an acceptable real match. However, when the apparent match is weak, as in the example described here, the apparent mismatches need to be explained. A description of such an explanation process follows:

A window into a part of the knowledge base is shown in figure 7.2. As an example of how an explanation process may resolve an apparently serious mismatch, a matching of 'unchanged chloride' with 'strongly increased chloride' is described below. This example shows Creek's ability to perform a context-dependent matching when a generally valid match is unobtainable. The concepts referred to in the explanation structures below are included in figure 7.2.

The following explanation is created in an attempt to explain the chloride difference:

- Increased chloride is caused by Salt water influx (fault of the case)
- Salt water influx is subclass of Salt water into mud (domain model)
- Salt water into mud is subclass of Water into mud (domain model)
- Water into mud has subclass Fresh water into mud (domain model)
Fresh water into mud does not change chloride (plausible inference)

So, strongly increased chloride and unchanged chloride 'matches' if their causes are abstracted to the entrance of water into the mud. This explanation is too weak, however, since it is not known whether 'Fresh water into mud' is a plausible state for the input case. If this could be justified, the above chain of relationships would constitute a coherent and strong explanation of the chloride mismatch. The two cases would then match on the abstracted feature 'Water into mud' and differ in the type of water entered (salt water vs. fresh water).

An attempt to explain the plausibility of the state 'Fresh water into mud' for the input case is then made, leading to the following explanation:

---

1This proposition is inferred from the facts that no state-change link exists between the fresh water concepts and chloride. A plausible inference rule is used, saying that if no state-changes follow from the model, then assume that no relevant state-change takes place.
Fresh water into mud is subclass of Water into mud  (domain model)
Water into mud always causes Mud dilution  (domain model)
Fresh water into mud always causes Mud dilution  (inheritance)
Mud dilution causes decreased Funnel Viscosity   (domain model)
Mud dilution causes decreased Yield-point   (domain model)

This explanation states that there is a strong causal relationship between the entrance of water into the mud and decreased viscosity values. Following this explanation, the decrease in cation exchange capacity (which indicates a decrease in clay content), and the increased fluid loss (i.e. filtration rate) are explained in a similar manner. All these explanations support the hypothesis that entrance of fresh water into the mud is a plausible state of the mud being diagnostized. The retrieved case is then marked as matching the input case via a modified solution, and the physical state which made the match possible, 'Fresh water into mud' is added as a solution constraint of the input case.

Another mismatch which needs an explanation is the 'weakly decreased calcium' vs. 'increased calcium'. The system is not able to generate a sufficiently strong explanation for this difference. However, the system finds that this finding has a low importance in the retrieved case and it is therefore disregarded. The same is true for alkalinity and pH. The water and oil content findings are not relevant findings in the retrieved case, and after attempting to explain their relevance for the input case, the system suggests to the user that they might be disregarded. The suggestion is accepted by the user.

The next step of Generate (G2) is to derive a set of plausible solutions, i.e. the chemically or mechanically related faults of the drilling fluid system. Since one and only one matching case is found, the solution of Case-321 is used to derive a solution for the input case. The general procedure is to take the fault of the retrieved case and attempt to explain its validity for the current problem. The fault of Case-321 is 'Salt water influx'. When justifying a solution, the solution constraints are checked first. One constraint states that the solution must be related to entrance of fresh water into the mud. If this constraint is not satisfied, the assumed match will be considered invalid, and the retrieved case will be rejected. Hence, a valid solution must be able to explain Fresh water into mud. Creek therefore attempts to modify the retrieved fault by searching for generalizations or causes of this fault which are explainable by the input case1. That is, the system follows the 'subclass-of' link (i.e. the inverse of the 'has-subclass' link shown in figure 7.2.) from 'Salt water influx' to 'Salt water into mud'. A search is made for the most specific specializations or effects of this concept which qualify as a fault hypothesis for

1In this example, Creek could have started with the state 'Fresh water into mud' and searched for subclasses or other fault concepts strongly related to this state. Since there, generally, may be several goal constraints, the general strategy is to start from the fault concept of the retrieved case, infer possible solution, and then compare each suggested solution with the constraints.
the input case. The attempt fails, and the system climbs one generalization level up, to 'Water into mud'. The search for specializations is repeated, this time leading to 'Fresh water into mud', with the possible specializations 'Added fresh water' and 'Fresh water influx'. Added fresh water means that fresh water has entered the system from other sources than the geological formation. The system checks the treatment history list for the current drilling operation, and finds that water recently has been added to the mud in order to dilute it. There are no indications of formation fluid influx, so the system hypothesizes the fault: 'Added fresh water'. The solution constraint (Fresh water into mud) is checked again and validated. The system is not able to infer other plausible modifications to the retrieved solution.

### 7.5.1.3. Selecting the best solution

In general, all plausible fault candidates are kept in a list ordered by calculated explanatory strength. Since only one good candidate is found in our example, the selection process is trivial. In order to illustrate this process in general, however, the evaluation steps corresponding to the subprocesses S1, S2, and S3 are exemplified in the following.

The first substep (S1) selects the fault with the highest explanatory strength. The explanatory strength of a fault is evaluated partly by propagating and combining strength values according to the basic explanatory strength formula:

\[
S(C_{2\text{new}}) = S(C_{2\text{old}}) + k_{\text{old}} \cdot S(C_1) \cdot S(R_{12}) \cdot (1 - S(C_{2\text{old}}))
\]

partly by applying explanation evaluation rules which take the calculated explanatory strength as input. Using the formula, the basic strength of the explanation for Added fresh water is 0.93, as shown below (\(k_{\text{old}} = 1.0 \) if \(S(C_{2\text{old}}) = 0\), otherwise \(k_{\text{old}} = 0.5\), \(S(\text{causes}) = 0.9\), \(S(\text{always causes}) = 1.0\), \(S(\text{subclass-of}) = 0.95\)).

<table>
<thead>
<tr>
<th>Relationship in explanation path</th>
<th>Accumulated explanation strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added fresh water (assumption)</td>
<td>---(\rightarrow) 1.0</td>
</tr>
<tr>
<td>Added fresh water is subclass of Fresh water into mud</td>
<td>1.0 * 0.95 * 1.0 --(\rightarrow) 0.95</td>
</tr>
<tr>
<td>Fresh water into mud is subclass of Water into mud</td>
<td>0.95 * 0.95 * 1.0 --(\rightarrow) 0.9</td>
</tr>
<tr>
<td>Water into mud always causes Mud dilution</td>
<td>0.9 * 1.0 * 1.0 --(\rightarrow) 0.9</td>
</tr>
<tr>
<td>Fresh water into mud always causes Mud dilution</td>
<td>0.9 + 0.5 * 0.95 * 1.0 * 0.1 --(\rightarrow) 0.95</td>
</tr>
<tr>
<td>Mud dilution causes decreased Funnel viscosity</td>
<td>0.95 * 0.9 * 1.0 --(\rightarrow) 0.86</td>
</tr>
<tr>
<td>Mud dilution causes decreased Yield point</td>
<td>..</td>
</tr>
<tr>
<td>Mud dilution causes decreased Plastic Viscosity</td>
<td>..</td>
</tr>
<tr>
<td>Mud dilution causes decreased Gel strength</td>
<td>..</td>
</tr>
<tr>
<td>Mud dilution causes increased Fluid loss</td>
<td>0.86 + 5/8 * 0.95 * 0.9 * 0.14 --(\rightarrow) 0.93</td>
</tr>
</tbody>
</table>

\(^1\text{See chapter 6.2.3.}\)
Note that the expressions involving findings of the input problem are calculated in a different way than the others, as explained in chapter 6.2.3. Multiple findings related to a concept in the explanation chain via the same relation, are combined in one operation. The resulting strength is calculated by use of a modified version of the $S(C_{2\text{new}})$ formula (corresponding to formula (II) in chapter 6.2.3). The differences are that $S(C_{2\text{old}})$ - in the expression shown above - in this situation represents the existing strength of combining all findings which are related to $C_1$ (Mud dilution) by the relation $R_{12}$ (causes). Therefore, 0.86 is the 'old' strength assigned to the findings of Mud dilution, as computed by the expression previous to the last one. The value of $k_{\text{cover}}$ (corresponding to $k_{\text{old}}$) is 5/8, since the total number of findings related to Mud dilution via a causal relationship is 8, and the number of findings used in the current explanation is 5.

The basic explanatory strength of the fault candidate "Added fresh water" is 0.9, as shown. Selecting the strongest fault candidate among competing hypothesis would be done by using this basic strength measure together with the set of explanation evaluation rules (not elaborated in this example).

If the result from the candidate evaluation process is more than one solution, the next step of the Select process (S2) tries to discriminate between them by use of the combined predictive strength of the relevant problem features: For each candidate case, the set of features matching relevant features of Input-case-1 is retrieved. For each such feature set, the values of the predictivity-strength fields for the features in the set is combined into a total predictivity strength for the set. The solution most strongly predicted is then chosen.

The final step of the Select process (S3) is triggered if the predictivity strength is unable to discriminate between competing hypotheses. This step relates the candidate solutions to possible external requirements. No such requirements are specified for the example discussed here, but relevant external requirements for the application could be based on information about the current drilling process, such as the type of formation being drilled.

Going back to the actual example, the suggested fault - Added fresh water ("The addition of too much fresh water into the mud") - is presented to the user, who confirms it as a plausible fault hypothesis, and asks for treatment suggestions.

Suggesting a treatment is performed by 'replaying' the steps G2 through S3. The task is different, but the basic method is the same. An attempt is made to justify the treatment of the retrieved case as a treatment for the problem case. Since both cases involve mud dilution, the adding of bentonite is explained as a useful treatment. Due to the treatment strategy, however, CMC treatment is postponed until the results of the bentonite treatment is evaluated. The user

---

1 Note that only two findings are shown in the partial model illustrated in figure 7.1.
2 It has to be an acceptable match, corresponding to the ‘==’ or ‘+++’ links previously described.
acknowledges this suggestion. If the treatment had not been acknowledged, the case-based method may have been applied in an attempt to retrieve a previous case with the same fault as the current problem. The set of relevant features used for case retrieval would have been extended with the feature representing the fault, which would have been considered a necessary feature. In our example, the result of problem solving is the following solution to the input case:

```
Input-case-1
...
Fault
  Added fresh water
Treatment
  Add bentonite
Possible additional treatment
  Add CMC
```

7.5.2. Learning

The input problem has been solved by retrieving a past case and modifying its solution. A new case therefore has to be constructed, either as an additional case or as a merging of the two cases. The learning algorithm in Creek is re-displayed in figure 7.3, and the path through the algorithm for the present case corresponds to the shaded part of the figure.

7.5.2.1. Extracting relevant problem solving descriptors

The Extract process uses the above criteria to select the pieces of information that are given as input to the subsequent learning step, the Construct process. The case descriptors extracted are the relevant findings of the two cases, their faults and treatments, and the associated explanations. The explanation which in this process gets assigned to the input problem case, is the explanation which represents the strongest path between a solution and the set of relevant findings. The explanations extracted in order to explain the fault and treatment, for example, is:

```
New-case
...
Fault
  Added fresh water
Fault explanation:
  Added fresh water is subclass of Fresh water into mud
  Fresh water into mud is subclass of Water into mud
  Water into mud causes Mud dilution
  Mud dilution causes decreased Funnel Viscosity, decreased Plastic Viscosity,
    decreased Yield Point, decreased Gel Strength,
    increased Fluid Loss
Explanation strength: 0.93
```
**Treatment**  
Add bentonite

**Treatment explanation**  
Add bentonite always causes increased Bentonite  
Increased Bentonite causes increased Funnel Viscosity, increased Plastic Viscosity, increased Yield Point, increased Gel Strength, decreased Fluid Loss

*Explanation strength: 0.95*

**Treatment strategy**  
Add small amount and observe changes

**Possible additional treatment**  
Add CMC

---

**Figure 7.3. Learning in the Current Example Session**  
The figure re-displays figure 6.10. In addition, the learning path followed in the example discussed here is marked with a gray zone. The learning algorithm is divided into three parts, corresponding to the Extract, Construct, and Store processes. The subsequent testing and refining/repair of a solution is included in the Store process, although it may involve other subprocesses as well.
### 7.5.2.2. Constructing a new or modified solution

The first task of the Construct process is to decide whether the modification made to the previous solution was significant enough to lead to the addition of a new case to the case memory, or whether the two cases are mergable. The degree of similarity between the two cases is not strong enough for them to merge (according to a pre-defined 'merging threshold'). A new case therefore has to be built. The construction process consists of building a new case structure, and involves three steps:

First, a new case description is assembled from the components gathered by Extract. The result - described in the CreekL frame representation language - is the following frame:

<table>
<thead>
<tr>
<th>FRAME case-501</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance-of</td>
<td>solved-case</td>
</tr>
<tr>
<td>has-process-status</td>
<td>diagnosis-not-confirmed</td>
</tr>
<tr>
<td></td>
<td>treatment- not-confirmed</td>
</tr>
<tr>
<td>has-input-time</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>12/07/90 14:10</td>
</tr>
<tr>
<td>has-relevant-finding</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>((unchanged-density 1.82 1.82))</td>
</tr>
<tr>
<td></td>
<td>((decreased-funnel-viscosity))</td>
</tr>
<tr>
<td></td>
<td>((decreased-plastic-viscosity 30 34))</td>
</tr>
<tr>
<td></td>
<td>((decreased-yield-point 11 14) :rate strong)</td>
</tr>
<tr>
<td></td>
<td>((decreased-gel-strength 5/14 6/14))</td>
</tr>
<tr>
<td></td>
<td>((unchanged-chloride 18000 18000))</td>
</tr>
<tr>
<td></td>
<td>((increased-fluid-loss 12/11 24/23) :rate weak)</td>
</tr>
<tr>
<td></td>
<td>((unchanged-solids 30 30))</td>
</tr>
<tr>
<td></td>
<td>((increased-ph 10.3 10.2) :rate weakly)</td>
</tr>
<tr>
<td></td>
<td>((increased-alkalinity 1.5/3.1 1.4/3.0) :rate weak)</td>
</tr>
<tr>
<td></td>
<td>((decreased-calcium 370 380) :rate weak)</td>
</tr>
<tr>
<td></td>
<td>((decreased-cec 18 22))</td>
</tr>
<tr>
<td></td>
<td>((unchanged-water-content 69 69))</td>
</tr>
<tr>
<td></td>
<td>((unchanged-oil-content 0 0))</td>
</tr>
<tr>
<td>has-suggested-diagnosis</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>(added-fresh-water 0.93)</td>
</tr>
<tr>
<td></td>
<td>(added-fresh-watersubclass-of. fresh-water-into-mud 0.93)</td>
</tr>
<tr>
<td></td>
<td>(fresh-water-into-mudsubclass-of.water-into-mud 0.93)</td>
</tr>
<tr>
<td></td>
<td>(water-into-mudalways-causes.mud-dilution 0.93)</td>
</tr>
<tr>
<td></td>
<td>(mud-dilution.causess(decreased-funnel-viscosity 0.93)</td>
</tr>
<tr>
<td></td>
<td>(decreased-yield-point 0.93)</td>
</tr>
<tr>
<td></td>
<td>(decreased-gel-strength 0.93)</td>
</tr>
<tr>
<td></td>
<td>(decreased-plastic-viscosity 0.93)</td>
</tr>
<tr>
<td></td>
<td>(increased-fluid-loss 0.93)</td>
</tr>
<tr>
<td>has-contaminant</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>fresh-water</td>
</tr>
<tr>
<td>has-suggested-additive</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>bentonite 0.95</td>
</tr>
<tr>
<td></td>
<td>(goal-state.has-instance.(increased-funnel-viscosity 0.95)</td>
</tr>
<tr>
<td></td>
<td>increased-yield-point 0.95)</td>
</tr>
<tr>
<td></td>
<td>increased-gel-strength 0.95)</td>
</tr>
<tr>
<td></td>
<td>increased-plastic-viscosity 0.95)</td>
</tr>
<tr>
<td></td>
<td>decreased-fluid-loss 0.95)</td>
</tr>
<tr>
<td></td>
<td>(bentonite.causes.(increased-funnel-viscosity 0.95)</td>
</tr>
<tr>
<td></td>
<td>increased-yield-point 0.95)</td>
</tr>
<tr>
<td></td>
<td>increased-gel-strength 0.95)</td>
</tr>
<tr>
<td></td>
<td>increased-plastic-viscosity 0.95)</td>
</tr>
<tr>
<td></td>
<td>decreased-fluid-loss 0.95)</td>
</tr>
<tr>
<td>same-diagnosis-in</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>case-037 case-179 case-015 case-061</td>
</tr>
<tr>
<td>same-treatment-in</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>case-055 case-111 case-155 case-101</td>
</tr>
</tbody>
</table>
Next, the importance and predictive strength of the relevant findings with respect to the new case are assessed. The system suggests values for these parameters, which are presented to the user for confirmation or modification. Since our diagnosis is different from that of the retrieved case, assessment of predictive strength for a finding is made on the basis of the number of other cases pointed to by that finding. Using the algorithm presented in chapter 6.4.2, predictive strengths of all relevant findings with respect to the fault added-fresh-water is computed. In our application, the rate of change of a finding is taken into account. Thus, whether a decrease or increase of some value is weak or strong is regarded significant when the predictivity of a finding for a particular fault is to be assessed.

An example: The decreased-yield-point finding points to, say, 91 cases. By examining each finding, it turns out that the rate of change is ‘strong’ in 18 of these cases. Assuming that the total number of cases in the case base is 500, the ratio \( N_{\text{case(Finding)}} / N_{\text{cases(All-findings)}} \) is set to 0.04. According to the algorithm, this gives the predictive strength value: indicative. The last part of the algorithm checks the explanation of the relevance of the finding to the case, in order to see whether the predictive strength should be increased. This does not turn out to be the case for the yield point finding.

The estimation of a finding’s importance is done according to the corresponding algorithm, as described in chapter 6.4.2. Unless particular conditions apply (first part of algorithm), the finding being assessed is removed from the explanation chain justifying the diagnosis, and the explanation is re-evaluated. The outcome of this process determines the importance of the finding for the case (actually, the diagnosis of the case). For example, removing the decreased yield point finding from the explanation (stored in the has-suggested-diagnosis slot of case 501), results in the following structure:

\[
\text{(added-fresh-water.subclass-of.fresh-water-into-mud)} \\
\text{(fresh-water-into-mud.subclass-of.water-into-mud)} \\
\text{(water-into-mud.always-causes.mud-dilution)} \\
\text{(mud-dilution.causes.(decreased-funnel-viscosity \\
\quad \text{decreased-gel-strength}} \\
\quad \text{decreased-plastic-viscosity} \\
\quad \text{increased-fluid-loss))}
\]

The strength of this explanation is computed as previously shown (subchapter 7.5.1.3), the only difference being that the causal relationship between mud dilution and decreased yield point has been removed. Thus, the ‘causal finding ratio’ of 5/8 is reduced to 4/8. The strength of the explanation without the yield point finding then gets the value:

---

*The system uses the explanations that previously were generated (the last step of the understanding process) in order to justify the relevance of each finding.*
S_{expl-finding}(Solution) = 0.86 + 5/8 * 0.95 * 0.9 * 0.14 = 0.92

The reduction in explanatory strength is 0.01, which according to the algorithm, and the values of its k-factors suggested in chapter 6.4.2, gives the finding decreased-yield-point the importance informative.

After having had these values computed and presented to the user for confirmation or adjustment, the system updates the has-relevant-finding slot of the case description, as exemplified below (note that the default values, 'characteristic' importance and 'strongly-indicative' predictive-strength, are not stored):

FRAME case-501
...
has-relevant-finding  value
  ((unchanged-density 1.82 1.82))
  ((decreased-funnel-viscosity ))
  ((decreased-plastic-viscosity 30 34) :importance necessary
    :predictive-strength indicative)
  ((decreased-yield-point 11 14) :rate strong
    :importance necessary)
  ((decreased-gel-strength 5/14 6/14))
  ((unchanged-chloride 18000 18000) :importance necessary
    :predictive-strength indicative)
  ((increased-fluid-loss 12 11) :rate weak)
  ((unchanged-solids 30 30))
  ((increased-ph 10.3 10.2) :rate weak)
  ((increased-alcainity 1.5/3.1 1.4/3.0) :rate weak)
  ((decreased-calcium 370 380) :rate weak)
  ((decreased-cec 18 22))
  ((unchanged-water-content 69 69))
  ((unchanged-oil-content 0 0))
...

7.5.2.3. Storing, indexing and evaluating the new case

The Store process creates indices to the new case, updates the existing case structure according to experiences gain during the present problem solving session, and tests whether the new case will solve the input problem the next time it is encountered.

7.5.2.3.1. Storing the new case by integrating it into the case base

Indexing the new case is done by updating the case pointers defined for the findings, solutions and treatments which describes the case. For the relevant findings, relevant factors are determined by the importance and predictive strength values (see chapter 5.2.5.2). For example:
increased-yield-point

relevant-feature-in value (case-100 0.9) (case-71 0.9) ... (case-501 0.95)

added-fresh-water

suggested-solution-in value (case-71 ... case-501)

Difference links are created between the new and previous case. This leads to the addition of difference links related to chloride content, pH, alkalinity, and calcium between the new case and case 321:

FRAME case-501

has-difference-link value ((increased-chloride case-321)
(decreased-ph case-321)
(decreased-alkalinity case-321)
(increased-calcium case-321)

Updating the existing case-structure is done by modifying case-321 in the following way:

• Increasing the importance of chloride to the case
• Decreasing the predictive strengths of the other relevant factors to the case
• Adding difference links to case 501 from the cases involved (i.e. generating inverses of case-501's difference links to case-321)

The modifications of importances and predictive strengths leads to a new value for the relevance factor, which are then updated in the indices to case 321. The increments and decrements of these values are very small, typically in the order of one per cent.

7.5.2.3.2. Evaluating the solution

Two subprocesses now remain, before the new case structure can be physically stored in the permanent memory:

1. The problem solving process is re-run, starting from the subprocess of initial case retrieval. If the new case is retrieved, and its solution is recommended by the system, step 2 is performed. If the new case is not retrieved as the best matching case, the reminding are adjusted. If the system is unable to justify that the solution of the new case is a plausible solution to the input problem, a problem in the model of general knowledge is indicated (needs manual intervention).
2. The entire new case and all the modifications are presented to the user for information and acknowledgement\textsuperscript{1}.

The input problem has now been solved and a new case has been integrated into the knowledge base. The remaining part of the learning process is to evaluate the suggested - and user-confirmed - solution by applying it to the actual problem. In the example domain, this is done immediately, and the effects of the treatment will usually be ready within a short time. A successful treatment only leads to updating the processing status of the case, while an unsuccessful treatment will lead to an attempt by the system to explain the failure, given the input of what went wrong. Such a situation will typically involve interaction with a highly competent user.

\textsuperscript{1}This step may be controlled by a switch, and turned off when the learning in the system has reached a sufficient degree of reliability.
This chapter starts by summarizing important properties of the Creek approach, with reference to the set of critical factors defined in chapter 4.2. The summary is followed by a discussion of the differences and similarities between Creek and the four systems described in chapter 4. The problem of how to evaluate artificial intelligence research is an important one, and crucial to the advancement of AI as a scientific field. A separate subchapter is devoted to this problem, related to the research reported in this dissertation. A list of questions representing focusing points for evaluating this type of research is presented and discussed. The final section of this chapter suggests some topics for future research.


Based on the requirements of the framework presented in chapter 3, chapter 4.2 defines the following set of critical factors for assessing the architecture and functionality of the type of systems addressed in this research.

- Thoroughness of the knowledge model
- Active role of explanations
- Combined use of reasoning methods
- Combined learning methods

This list was used to summarize properties of the PROTOS, CASEY, JULIA and CHEF systems, and is used here to highlight important properties of CREEK. In the following summary, a "+" sign indicates a significant positive property, while a "-" sign indicates a

---

1This not a problem particular to AI, however, it is shared by a lot of scientific disciplines, including the disciplines which AI 'rides on the back of': Computer Science, Cognitive Psychology, Philosophy.
limitation of the Creek approach with respect to the critical factors. A brief discussion of the limitations follows each of the property listings.

**CF1: Thoroughness of knowledge model**

+ Four knowledge submodels (an object-level knowledge model, a diagnosis and repair task model, a model of combined reasoning and a model of sustained learning) are integrated into a unified knowledge environment, in which the concepts of all four models are defined through intra- and inter-model relationships.
+ Three knowledge forms (cases, deep models, rules) are represented within a unified representation language.
+ A comprehensive set of relations are defined, with the option to define other relations as explicit concepts in terms of already defined relations and/or Lisp functions.
+ Quantifiers, qualifiers and conditionals are used to specify scopes, strengths and contexts of relationships.
+ An open knowledge model is assumed, which - in principle - never should be assumed complete and finished, but rather be viewed as a basis for discussions and modifications by a competent user.

- The emphasis on expressiveness and flexibility of the representation language may lead to reduced semantic clarity.

The means provided by Creek to maintain semantic clarity is the explanation mechanism, the underlying inference methods, and the fact that all relations of the modelling language have to be explicitly defined. Coherence within the knowledge model is 'enforced' by requiring that updates do not contradict with the existing facts and constraints unless there is an explainable reason for it. Semantic interpretations of the knowledge constructs are defined by the basic inference methods of default inheritance, constraint propagation, and activation spreading. Above this level a set of plausible inference rules control and guide the inference processes, which in turn is controlled and used by the explanation mechanism in order to infer a best possible result within a given context.

**CF2: Active role of explanations**

+ Explanations support the extraction of relevant features from the problem description, the matching of cases, the assessment of solution candidates, the merging of cases, the modification of solutions, the selection of relevant features to be stored, and the assessment of their degree of relevance for the case.
+ The explanation mechanism takes advantage of the expressive representation language by explicitly defining particular groups of explanation relations for particular explanation tasks, and a set of control rules which operate on these relation sets.
Chapter 8 - Discussion

+ Control heuristics for generating and evaluating explanation paths.
+ Explanation paths are stored within cases, enabling the reusing of past explanations in the case matching, case merging and solution modification processes.
+ Rating of explanation strengths by computing the combined strength of an explanation chain from the explanatory strength of single relations (for a particular explanation task).

÷ Explanations are not used in the spreading activation process.

Spreading activation in Creek is a rather weak method, since it spreads along a pre-defined set of relations, without dynamically considering consequences and other effects. Using explanations to guide the spreading activation process may lead to a more focused context, and, hence, improved problem understanding. On the other hand, it may lead to limiting the context too much in this early phase.

CF3: Combined use of reasoning methods
+ A spreading activation method is used to activate concepts within a broad problem context, thus limiting the subsequent reasoning to a subpart of the (presumably large) knowledge base.
+ Model-based reasoning supports various phases of case-based reasoning.
+ Rule-based and model-based reasoning methods to fall back on if case-based reasoning fails.
+ Control level reasoning using explicit knowledge of strategies and tasks.

÷ Although a part of the top level architecture, methods for solving problems without the use of cases has not been elaborated and detailed.
÷ The combined reasoning model is to a large extent specified as an algorithm, rather than as a fully explicit knowledge model at the control level.

The focus of this research is on case-based reasoning. As a consequence, the case related methods have been given the most detailed treatment. The other reasoning methods have mainly been viewed from a case point of view, i.e. as support methods and joint partners for the case-based methods. Within the architecture, however, the basic method for solving problems by reasoning within the general knowledge model only, is a plausible reasoning method which combines the spreading activation and explanation mechanisms. Hypotheses are generated by a controlled spreading activation process, and explanations are produced in order to assess the plausibility and relevance of each hypothesis.
The combination of cases and rules was studied by Christian Larssen in his master thesis research [Larssen-90], and a rule-based reasoner based on the CreekL language was specified and implemented.

**CF4: Combined learning methods**

- A case-based learning method, with explanation-based support for various phases of the learning process.
- A learning apprentice approach, i.e. learning through active interaction with the user - by asking for, receiving and explaining information from the user.
- Explanation-based generalization of features during the merging of similar cases.
- No explanation-based learning of generalized concept definitions.

The potential of using the model of general knowledge and the explanation mechanism in Creek for explanation based generalization is interesting. However, this would be a different research topic than the one addressed here. A study of combining the learning of specific cases with learning of generalized concept definitions was done by Velitchko Koulitchev in his thesis research [Koulitchev-90]. An algorithm for combined learning, using CreekL as the representation language, was implemented.

Creek acquires general domain knowledge through interaction with the user. The explanation mechanism is the basis for this facility, since it identifies contradictions, ambiguities, errors, and missing parts in the general knowledge model as it attempts to produce explanations in the course of normal problem solving.

The Creek architecture represents an integrated and knowledge-intensive approach to problem solving and sustained learning, with an emphasis on case-based reasoning. Figure 6.11 summarizes the interplay between the basic reasoning and learning methods in Creek.

### 8.1.2. Comparing Creek to the Systems Previously Analyzed.

As previously remarked, the case-based methods of the Protos system has formed a basis for developing the case methods in Creek. However, there are several significant differences. The similarities and differences with Protos, as well as with CASEY, CHEF and JULIA, are summarized below, with reference to the five descriptive dimensions defined for the framework (chapter 3.3), i.e. the problems addressed, the knowledge model, the problem solving process, the reasoning methods, and the learning.
1. The Problems Addressed

The problem addressed by Creek is competent problem solving and sustained learning for real world knowledge-based systems within weak theory domains:

**Type of Problem to be Solved**

- **Problem type:** Diagnosis and treatment in natural and technical domains.
- **Application focused:** Diagnosis and treatment of mud during a drilling operation.
- **Type of input:** Quantitative and qualitative values of observed symptoms, measurements, test results, etc.
- **Type of output:** One or more faults and one or more treatments, with justifying explanations.
- **External environment:** User interaction throughout the problem solving and learning phases. The system is an active assistant. Open architecture, aimed to interact with other computing systems.
**Type of Learning**

Learning type: Learning to assign diagnostic classes and repair classes to a set of problem descriptors.

Type of input: A set of relevant problem features, the assigned diagnosis and treatment, a matching past case. After evaluation: Failed solutions.

Type of output: A new or modified case integrated into the model of general domain knowledge. Modified general knowledge model.

External interaction: The user provides input in order to resolve knowledge conflicts or to confirm/reject proposals by the system.

Creek represents an integrated problem solving and learning environment, where case-based reasoning is the main part. The four other systems address the more specific problems of concepts learning (Protos), learning more efficient problem solving by compiling deep knowledge into cases (CASEY), learning by recovering from failure (CHEF), and mixing of case-based and constraint-based problem solving (JULIA).

The type of problem solving task addressed by Creek is the same as Protos’ and CASEY’s: Classification type problems, or more specifically, diagnosis and treatment of faults in complex, open, real-world systems. CHEF and JULIA addresses planning problems, also in open domains. From the case-based reasoning research reported in the literature, reasoning from past cases seems to be applicable and suitable to planning and design as well as classification.

### 2. The knowledge model

Creek’s models of knowledge dimensions, expertise components, and representational primitives are illustrated in figures 3.2, 5.2, and 5.5, respectively. The representation of general knowledge, explanations, and cases are described in the sections of chapter 5.2.5, while knowledge integration aspects are described in chapter 5.2.11.

Creek’s major difference with respect to the other four systems is the role played by the general knowledge model. Creek represents a qualitatively different level of knowledge-intensive support to the case-based reasoning and learning processes, compared to any of the four systems. Closest in this respect is CASEY, which is the only other system with a sufficiently strong domain model to be utilized for tasks such as modifying a solution, and solving a  

---

9The pointers are included in order to make this summary description consistent with the descriptions of chapter 4, where the topics listed are subheadings used in describing the knowledge models of the four systems.
problem from scratch. The problem with CASEY’s approach is its blind reliance on its causal knowledge model. The model is regarded as complete, and the system is not able to interact with a competent user in its problem solving or learning processes. This may be a sound approach for closed problem domains, but not for the open type of domains addressed by the Creek architecture. JULIA does not have a conceptual model, its general knowledge is modelled as a set of constraints. CHEF’s general knowledge is basically a network of states and actions, closer to a set of inference rules than to an explicit conceptual model. Protos has a rich set of relations in its modelling language, and it exhibits a potential for developing a thorough and strong conceptual model. However, the role of the general knowledge model in Protos is limited to supporting the case-based method on a subset of problem solving and learning tasks.

The active role of Creek’s knowledge model is enabled by an expressive representation language, emphasizing explicit definitions of all terms to be reasoned about. Protos has the most expressive representation of the four other systems, although it is not able to represent structured features. None of the other systems treat relations as explicitly defined concepts.

3. The problem solving process

The spreading activation method (chapter 6.3.2) as a means to initially focus the problem solving context is a new approach.

Creek has an explicit model of problem solving tasks and strategies, which it instantiates for a particular problem. The problem solving strategy of Creek is the DRM submodel described in chapter 6.1, which also includes a task model of diagnosis (chapter 6.1.3 and figures 6.4 and 6.5). CASEY and Protos do not reason with control knowledge. JULIA has a goal scheduler which sets up the problem solving tasks, but it is not an explicit model of problem solving concepts, actions, etc. CHEF is the prominent example of a strategic reasoning system among the four systems. It has a model of repair strategies and a set of modification rules which control the recovery from a failed solution. CHEF’s control model addresses the repair of a failed solution, not how to solve a problem in the first place.

An important property of the Creek approach is its basis within a view to knowledge acquisition, problem solving and sustained learning as constructive and cooperative processes, in which a human user is an integrated part (cf. chapter 1.2.1 and figure 1.1). No part of the domain knowledge is defined as fixed and unchangable, i.e. any concept within the total network of object and control knowledge may be re-defined by the knowledge engineer and/or

1Actually, CASEY applies its closed model approach to a highly open problem domain: Medical diagnosis.
the user\(^1\). A Creek system should be viewed as a learning apprentice. It is able to solve - or participate in solving - problems at a continually increasing level of success. It learns partly by compiling its general model knowledge, partly by observing how a competent user solves problems, and integrating this information into its knowledge base. The explanation mechanism is the crucial part in this integration process. The Protos system also fits within this constructive and cooperative view, while CASEY does not. JULIA and CHEF may have a potential in this direction, although they are not described as such systems.

4. The reasoning

Creek’s algorithm for multi-paradigm reasoning, and the generic case-based reasoning structure, are summarized in figures 6.6 and 6.7, respectively (chapter 6.3.1).

Protos, JULIA and CHEF all combine case-based reasoning with reasoning from general knowledge, but general knowledge is not used for problem solving on its own. CASEY is able to solve problems within its causal model. Creek is a three-paradigm reasoning architecture, with case-based reasoning as the main paradigm.

5. The learning

All five systems learn from experience by extending their stores of past cases. In addition, both CASEY and Protos are able to generalize feature descriptions by following generalization links in their general knowledge models. CHEF learns from a failure by explaining the cause of the mistake, and generalizing this cause. Thus, CHEF comes close to an explanation-based learner as well as a case-based one. Creek does not generalize to the extend that CHEF does, it is more similar to Protos in this respect, being a 'lazy generalizer'. Creek and Protos share the assumption that specific knowledge is a type of knowledge valuable for problem solving in open, complex domains. Hence, the particular features that make specific knowledge specific should not be ‘generalized away’.

A major difference between Creek’s learning model and Protos’ model - at the abstraction level illustrated by figures 6.12, 4.6, and 4.13 - is that the EXTRACT step in Creek does not involve any explanation of features, since this has already been done during problem solving. In Protos, on the other hand, unmatched features may be disregarded during problem solving, as long as the total matching strength is above a certain threshold. In such situations, the unmatched features are evaluated during the extraction step of the learning process. Further,

\(^{1}\)What type of knowledge various type of users should be allowed to modify is, of course, a matter to be decided by organizational requirements of the operational system environment.
Creek - like CASEY - retains within a new case the explanation path which justifies the solution and treatment. A difference between Creek and CASEY is that Creek stores the qualitative findings (if necessary transformed from quantitative measurements) within cases, while CASEY also retains the most general intermediate states for which there is evidence in the findings (the so-called causal features). CASEY also retains all input findings in its learned case, without evaluating them. So does JULIA and CHEF, as well, but these two systems use a restricted set of pre-defined problem descriptors, which are always assumed relevant.

8.2. Evaluating AI Research Results.

This subchapter addresses the problem of evaluating AI research results. The problem is introduced as a general one, then related to Creek in particular.

8.2.1. A set of significant questions.

Unfortunately, there exist no methods - or set of criteria commonly agreed upon - by which a system architecture such as Creek’s can be systematically evaluated with respect to its achievements, limitations, and contributions to the advancement of science. The problem of evaluating AI research in general, and case-based reasoning research in particular, has been discussed in the literature [Cohen-88b, Cohen-89, Koton-89b, Bareiss-89]. Cohen [Cohen-89] presents a list of 6 questions which should be answered by researchers when presenting their AI methods:

"1. What are the metrics for evaluating the method (e.g. cognitive validity)?

2. How is the method an improvement, an alternative, or a complement to existing technologies? Does it account for more situations, or produce a wider variety of desired behaviours, or is it more efficient in time or space, or model humans better?

3. What are the underlying architectural assumptions? Are all design decisions justified? Does the method rely on other methods (e.g. do we assume an indexing method when developing a learning method)?

4. What is the scope of the method? How extensible is it? Will it scale up? Does it exactly address the task, or portions of the task, or a class of tasks? Could it or parts of it be applied to other problems? Does it subsume some other method?

5. Why does the method work (or not work)? Under what circumstances won’t it work. Are the limitations of the method inherent, or simply not addressed?

6. What is the relationship between the class of tasks, of which the current task is an example, and the method? Can this relationship be stated clearly enough to support a claim that the method is general to the class of tasks?"
8.2.2. Evaluating the Creek approach.

The result of the research reported here has two major parts: The first part is the theoretical framework for knowledge modelling, problem solving, reasoning and learning presented in chapter 3. The second major part is the architecture - including the underlying method for representation of multiple forms of knowledge, the diagnostic problem solving strategy, the combined reasoning and learning method with emphasis on reusing and learning specific cases. The two parts are jointly referred to as the 'Creek approach'.

Below, the Creek approach is discussed with reference to the questions listed above. This does not constitute an evaluation of Creek, of course, it is rather a discussion of topics relevant to such an evaluation.

Parts of Creek has, in a sense, been validated through its use in other projects: A few minor adaptations to the CreekL representation system has enabled it to be used as a general frame representation system within the Acknowledge project (named SFL - for Sintef Frame System [Aakvik-90]). A reduced version of the CreekL language, and an extended version of Creek's spreading activation mechanism, has been implemented as part of the KNOWIT system - a fully implemented prototype system for knowledge-based information retrieval [Sølvberg-91].

**Question 1 - Are there metrics for evaluating the Creek approach?**

The Creek framework and architecture is aimed at improving a knowledge-based system's ability to solve real world problems through interaction with a human being. The assessment of a system design such as Creek - and hence its underlying methods - can only be based on qualitative factors. The only 'real test' of such a architecture and system design is whether its implementation will lead to systems that are being accepted and regularly used in the type of environments addressed. All other evaluation efforts will only be partial.

In chapter 4 the premisses and analytic properties of the framework were used to select four existing systems and discuss their pros and cons. Thus, one evaluation 'metric' is the degree to which Creek fulfil the requirements R1 - R3 listed in chapter 3.1. This was discussed in the previous subchapter.

Another relevant 'metric' is the cognitive plausibility of the approach, i.e. to what degree the system design reflects psychological models of knowledge representation, problem solving and learning. This metric is relevant for two reasons: First, models of human cognition may provide ideas and guide-lines for how to achieve intelligent behavior in a computer system. The history of AI clearly shows a strong connection between AI and cognitive sciences.
Second, the primary type of systems aimed at by the Creek architecture are interactive, knowledge-based decision support environments. Hence, the systems will interact with human beings for development and refining of their knowledge models, as well as to solve problems and learn from experiences during normal operation. The closer the system's knowledge modelling approach, reasoning methods, etc., are to human models, the easier it is likely to be to obtain a productive, user-friendly, and cooperative decision support system.

There is still an active debate among psychologists and philosophers concerning the nature of human knowledge, reasoning, and learning, which indicates that such a metric can only be used to assess general properties, not specific solutions. For example, this metric may be applied at the level of system requirements (R1-R3), rather than at the level of system design. Psychological arguments in favor of knowledge intensive case-based methods are briefly reviewed in appendix A1 and chapter 1.1.

**Question 2 - Improvement over existing approaches**

Throughout the presentation of Creek - the framework, the architecture, as well as the more specific system design - the active role of a thorough domain model in producing explanations for supporting the various reasoning and learning steps have been emphasized. This kind of knowledge-intensive support is not met in any other known case-based systems. Neither is the combined, three-component method of case-based, rule-based, and model-based reasoning. The case-based reasoning method itself should be regarded as an alternative to existing methods, while the spreading activation method used to establish an initial broad context for problem solving and learning, is a new suggestion.

The four-component architecture of explicit object-level and control-level models is an improvement compared to other related approaches. The underlying, frame-based representation system is designed to facilitate explicit representation of all types of relevant knowledge. The expressiveness of the CreekL language, and the fact that its internal structure and representational primitives are explicitly represented, is an improvement over the typically very simple representation schemes used for most problem solving and machine learning systems.

The combined reasoning method enables a Creek system to solve a wider range of problems than single reasoning systems do. The spreading activation method 'filters' out a part of the knowledge base assumed to contain the concepts relevant for subsequent problem solving, and has the potential of making elaborate, deep reasoning procedures tractable even if the
knowledge base is large. Further studies are needed in order to give more precise statements about the advantages, problems, and limitations of this method.

**Question 3 - Underlying assumptions in the architecture and methods.**

The design decisions made in Creek are motivated by the need for more robust and competent knowledge-based systems, which are able to continually learn from solving problems. The methods proposed are justified by the current state of art in AI, and by the needs to combine several methodological paradigms in order to achieve the system properties wanted.

In order to build the necessary models of generalized knowledge used in Creek, this knowledge has to exist, and be expressible in conceptual models involving deep as well as shallow relationships. The case-based method assumes that there exists a means by which two problem situations can be compared and their degree of similarity determined.

**Question 4 - Scope of the method**

The integrated, knowledge-intensive approach of Creek relies on the existence of a general knowledge model for producing explanations. The Creek approach addresses real world problems in open and weak theory domains\(^1\).

The method of learning by storing experiences as specific cases, depends on the availability and frequent addition of a certain amount of cases. Further, the problems encountered need to have some degree of regularity, and there has to be some practically useful criteria for assessing the similarity of two cases. A case-based method is hardly suitable for applications which will assist in solving a few complex problems over a wide period of time\(^2\).

Although the type of problems addressed here are classification problems, there are no obvious arguments disfavoring a similar knowledge-intensive, integrated approach - with CBR as its primary paradigm - for other classification problems or planning and design problems (with the exceptions stated in the previous paragraph). Design problems, for example, are typically more complex and 'open' than diagnosis problems, which should indicate an even larger gain by reasoning from past cases, compared to other approaches. Of the four knowledge modules of the Creek architecture, only the Diagnosis and Repair model is specific to the type of classification task addressed.

\(^1\) Problem solving and learning in domains with a strong domain theory (i.e. where most relationships are certain), should use more stringent methods, for example methods based on mathematical logics, to represent the knowledge and to prove properties of problem instances.

\(^2\) The extreme in this sense is one-time problems, for example putting a man on the moon...
Question 5: Why does the method (presumably) work?

The method of combining the three reasoning paradigms is based on the fact that rule-based reasoning, case-based reasoning, and model-based reasoning ‘works’. A significant number of systems have been built - within each of these paradigms - and shown to work. As previously argued (chapter 1) all these single paradigms have limitation as separate methods, and the basic idea of Creek is to integrate them into a common framework and system architecture. The major argument in favor of a success for this integration effort, is the role of the thorough and deep model of conceptual knowledge as the common ‘glue’ that keeps the various parts and modules together. Since all types of knowledge are explicitly represented within a common representation system, the principle of justifying an hypothesis generated by any of the system modules by an explanation within the deeper model, ensures a unified ‘understanding’ by the total system. The crucial part, then, is the quality of the method by which explanations are generated and evaluated. Creek’s method of assigning default strengths to the set of relations, and moderating the resulting accumulated strength of an explanation path by explanation evaluation rules has similarities with Protos’ method. Protos has been successfully evaluated with respect to correctness of classification after learning [Bareiss-89]. This suggests that the Creek approach is unlikely to be off track.

Question 6: Generality of the method to the class of task

This question is partly answered under question 4. The Creek architecture and system design constitutes a hierarchy of methods, ranging from the high-level notions of knowledge-intensive case-based reasoning/learning, via the three reasoning paradigms, the learning paradigm, the spreading activation idea, the four knowledge modules, down to the specific methods exemplified in the mud application for explanation, case-matching, construction of learned cases, etc. Obviously, some components are task dependent, while others are unlikely to be. A thorough analysis would be needed to identify the dependency of each method to the various levels of tasks and subtasks.

8.3. Suggestions for further research

Several suggestions for further studies related to the proposed Creek approach have already been indicated in this and previous chapters. In this subchapter some particularly interesting investigations and extensions related to the existing system design are briefly discussed.
The complexity of an integrated system such as Creek is hard to evaluate without a more extensive implementation. An implementation should start with those parts that are specified in most detail, e.g. by implementing the Mud Creek system. Some of the methods will probably turn out to be too simple or to need an improvement for other reasons. The purpose of the implementation should be to identify strong and weak points of the current system design.

As already mentioned, the important role of the explanation mechanism suggests that the explanation method should be a topic for further studies. In particular, the explanation evaluation rules have been presented in this report more as an idea than as well worked-out mechanism. Does the combination of numeric default strengths of relations, and a rule-controlled method for evaluation of explanations as they are constructed, provide the intended effect of ensuring that meaningful and useful conclusions are inferred? In the KNOWIT project [Sølvberg-91] conducted at ELAB-RUNIT, a method for assessing the relevance of a concept with respect to another concept was developed. Here, the relevance of a concept is determined on the basis of the preceding relation-concept-relation triplet leading to the concept in question. It should be investigated whether the explanatory power of Creek would improve by adopting this method.

Although the CreekL representation language has the potential of expressing various knowledge types, no higher level language constructs have been defined to facilitate representation of complex structures like rules containing variables, processes and procedures, temporal relationships, spatial models, etc. A study of how such structures are expressed in other frame systems should be undertaken, with the aim of extending the existing CreekL implementation. Corresponding inference procedures would need to be implemented, and explicitly modelled within the ISOPOD knowledge base.

An interesting extension to the case based learning method would be to learn generalized knowledge, in the form of heuristic rules, from the more deep knowledge represented in the conceptual knowledge model. Operational problem solving heuristics could be 'chunked' by 'compiling' relevant explanation chains (chains of plausible arguments) guided and focused by the problem solving goal. Such methods have been studied as a single learning method, but within the Creek context this method should be combined with case learning - for example by attempting to generalize into a rule and learn the case if rule generalization fails (or learn rules in addition to cases, since rules and cases have different properties). This combined learning method should also lead to the development of an explicit control level model of sustained learning - a learning strategy model.

A problem related to the learning of cases which barley has been touched in this dissertation is that of forgetting. The size of the case base will eventually grow very large unless some
precautions are made to prevent this. A very large case base may not only represent a storage problem, but it will lead to decrease of efficiency in the case retrieval and matching processes. There are several ways in which this problem could be handled. One is related to the previous topic of the paragraph, namely generalizing cases into heuristic rules. The disadvantage of this is that the benefits of reasoning by re-using specific past problems when solving problems in weak theory domain are lost. Another option is therefore to merge cases more extensively when the size of the case base grows. A third option - called forgetting - is to systematically delete cases in order to keep the case-base at a stable size. For example, cases which has not been used since a particular date may be removed. A more semantic criteria may serve equally well, such as removing the cases related to problems which are obsolete in equipment based on new technologies.

No studies has been undertaken regarding the performance efficiency of a Creek application. The main goal of Creek has been to develop a knowledge-based system architecture that supports knowledge-intensive and case-based methods for reasoning and sustained learning. Highly expressive representation formalisms and complex reasoning schemes are, in general, computational expensive. The study of these problems may take at least three directions: One is a computational approach, like investigating low level data access methods, parallel processing, etc. Another is a representational approach, i.e. to constrain the representation of knowledge and reasoning so it fits more readily to the underlying computational processes. The third approach emphasizes machine learning methods for improving performance and stabilizing the size of a knowledge base. This may be done, e.g., through continually learning of generalized and operationalized - purpose directed - knowledge as more experience is gained.

8.4. Conclusion

There is still a large number of important and challenging problems to be addressed in order to improved the quality and usefulness of expert systems for practical, real world problems. The research reported here have addressed the problem of how to achieve, and continually maintain, a higher level of competence and robustness in such systems than what they possess today. The problem has been approached from two sides:

- Strengthening of the problem solving capability by combining several reasoning paradigms within a knowledge-rich environment, focusing on case-based reasoning as the major method.
• Enabling a continually improvement of an incomplete knowledge base by learning from each problem solving experience, using a knowledge-intensive, case-based learning method.

The resulting framework, architecture, system design, and representation platform - i.e. the Creek approach - has been motivated and supported by relating it to strengths and weaknesses of other approaches. Its core methods has been demonstrated through a knowledge model and an example session in the domain of mud diagnosis and treatment.
APPENDIX
Appendix 1

An Introduction To Case-Based Reasoning

A1.1. General

When presented with a new problem, a person is often reminded of a previous problem similar to the one at hand. For example:

- A physician - after having examined a particular patient in his office - gets a reminding to a patient that he treated two weeks ago. Assuming that the reminding was caused by a similarity of important symptoms (and not the color of the patient's sweater, say), the physician uses the diagnosis and treatment of the previous patient to determine the disease and treatment for the patient in front of him.
- A financial consultant working on a difficult credit decision task, uses a reminding to a previous case, which involved a company in similar trouble as the current one, to recommend that the loan application should be refused.
- A drilling engineer, who have experienced two dramatic blow out situations, is quickly reminded of one of these situations (or both) when the combination of critical measurements matches those of a blow out case. In particular, he may get a reminding to a mistake he made during a previous blow-out, and use this to avoid repeating the error once again.

The examples above illustrate that reasoning by re-using or modifying past experiences is a powerful and frequently applied paradigm for human problem solving. This claim is also supported by results from cognitive psychological research. Schank [Schank-82] has developed a theory of learning and reminding based on retaining of experience in a dynamic, evolving memory structure. Anderson [Anderson-83] shows that people use past cases as models when learning to solve problems, particularly in early learning. Later, according to Anderson, concrete experiences get generalized and operationalized ('compiled') into production rules. Other results (e.g. by W.B. Rouse, as described in [Kolodner-85]) indicate
that the use of past cases is a predominant problem solving method among experts as well. Studies of problem solving by analogy (e.g. [Gentner-83], [Carbonell-83]) also shows the frequent use of past experience in solving new and different problems. Analogy reasoning is closely related to case-based reasoning, but the study issues focused are different in the two disciplines. While a main research issue in analogy ([Kedar-Cabelli-86], [Hall-89], [Burstein-89]) is the mapping of a new problem description (called the target) to a known problem (called the base) in a different domain, case-based methods focus on indexing and matching strategies for single-domain cases. Although their methods typically are different, analogy and case-based research problems have several meeting points, and the two paradigms should be regarded as partially overlapping rather than being completely different.

An important feature of case-based reasoning (CBR) is its coupling to learning. The driving force behind case-based methods in AI comes from the machine learning community, and case-based reasoning is regarded a subfield of machine learning. Thus, the notion of case-based reasoning does not only denote a particular reasoning method, irrespective of how the cases are acquired, it also denotes a machine learning paradigm that enables sustained learning by updating the case base after each problem solving session. In CBR, learning occurs as a natural ‘by-product’ of problem solving. When a problem is successfully solved, the experience is saved in order to solve a similar problem more efficiently the next time. When an attempt to solve a problem fails, the reason for the failure is identified and remembered in order to avoid the mistake in the future. Effective learning from a problem solving experience requires a sophisticated set of knowledge based methods in order to extract relevant knowledge from the experience, integrate a case into an existing knowledge structure, and index the case for later matching with similar cases [Rissland-89, Kolodner-85].

While in the earliest case-based approaches, a previous case was retrieved based on superficial, syntactic similarities among problem descriptors (e.g. the CYRUS system [Kolodner-83a, Kolodner-83b]), more recent approaches attempt to match cases based on relevant features that are semantically similar (as in the PROTOS [Bareiss-88a], CASEY [Koton-89] and GREBE [Branting-89] systems). In order to match cases based on semantic similarities and relative importance of features, an extensive body of general domain knowledge is needed to produce an explanation of why two cases match and how strong the match is.

---

1 Anderson’s domain was geometry problems, a well defined domain with a complete domain theory, while Rouse observed trouble-shooting by steam plant operators, a more open domain with an incomplete domain theory. Different characteristic of the domains may explain the different results of these experiments.
The CBR paradigm covers a range of different methods for organizing, indexing, retrieving and utilizing the knowledge retained in past cases. Cases may be kept as concrete experiences, or a set of similar cases may form a generalized case. Cases may be stored as separate knowledge units, or split up into subunits and distributed within the knowledge structure. Cases may be indexed by a prefixed or open vocabulary, and within a flat or hierarchical index structure. The solution from a past case may be directly applied to the present problem, or modified according to differences between the two cases.

A note on terminology:
The term *case* is a general notion. It may refer to a problem that is to be solved (frequently called *problem case, new case, input case, current case, or just problem description*), as well as to a previous, stored experience (often referred to as *previous case, stored case, retained case, memorized case, past case, solved case or just past experience*). An input case generally differs from a stored case in having a larger number of features and no solution. In a stored case, typically, only features that have turned out to be relevant for solving the current problem are stored. A stored case may also contain parts of the derivation steps which lead to the solution, or some other explanation of the solution. Some approaches also retain unsuccessful problem solving paths and failed suggestions, in order to avoid similar mistakes later. The term *memory* is often used to refer to the storage structure that holds the previous cases. A memory, thus, refers to what is remembered from previous experiences. Correspondingly, a *reminding* is a pointer structure to some part of memory.\(^1\)

The next section presents the fundamental steps undertaken in CBR as a cycle of four steps, corresponding to four main tasks. The third section outlines two important models for indexing and storing of cases, while the fourth and final section lists some application systems where case-based reasoning have been used.

**A1.2 The Cbr Cycle**

At a high level of abstraction, a CBR process may be described by the following four steps:

1. Understanding and remembering.
   
   Understand the problem and find the best matching previous case.

2. Transfer and adaption.
   
   Build a solution to the new problem using the previous case.

3. Testing and evaluation.

\(^1\)Note that this is different from the interpretation of the term 'memory' in computer science in general (i.e. as internally stored and fast accessible data structures, as opposed to external data stored on peripheral units like disks).
Attempt to solve the new problem, and evaluate the result.

4. Learning.

Learn by extracting and storing experience from the problem solving steps.

The first two are problem solving steps, the third step applies the suggested solution to the problem and evaluates the result, while the fourth step learns from the experience.

1. Understand the problem and find the best matching previous case.

Problem understanding involves finding relevant problem descriptors, checking of values, etc. The process may infer other problem descriptors than those given as input. By using a model of general domain knowledge, or retrieving a similar problem description from the case base, the system may derive consequences and generate expected findings from the ones entered. Once the problem is 'understood', the task of finding a good match is typically split into two subtasks. First, a set of potentially relevant cases is retrieved. Second, these cases are further checked and analyzed in order to find the best - or a sufficiently good - match.

Retrieving a set of relevant cases is done by using the problem descriptors (input features) as indices to the case memory. Cases may be retrieved solely from input features, or also from features inferred from those entered. Cases that match all input features are, of course, good candidates for matching, but - depending on the retrieval strategy - cases that match a given fraction of the problem features (input or inferred) may also be retrieved. Some tests for relevance of a retrieved case is often executed, particularly if cases are retrieved on the basis of a subset of features. For example, a simple relevance test may be to check if a retrieved solution conforms with the assumed or pre-defined solution type of the new problem. Initial case retrieval may also be more knowledge-based than described so far, for example by trying to understand the problem more deeply, and using the goals, constraints, etc. from this elaboration process to guide the retrieval process. Such a strategy is able to weigh the problem descriptors according to their importance in characterizing the problem, and retrieve a set of past cases based on characteristic features.

From the set of similar cases, a best match is chosen. This subtask is usually a more elaborate one than the retrieval task, although the distinction between retrieval and elaborate matching is not distinct in all systems. The matching process typically generate consequences and expectations from each retrieved case, and attempts to evaluate consequences and justify expectations. This may be done by using the system's own model of general domain knowledge, or by asking the user for confirmation and additional information. The cases are eventually ranked according to some metric or ranking criteria. Knowledge-intensive matching methods typically generate explanations that support this ranking process, and the case that has the strongest explanation for being similar to the new problem is chosen. Other
properties of a case that are considered in some CBR systems include relative importance and discriminatory strengths of features, prototypicality of a case within its assigned class, and difference links to related cases.

2. Build a solution to the new problem using the previous case.
This task constructs a solution to the new problem based on the solution of the matching previous case. The solution may be copied directly, or adapted according to significant differences between the two cases. Additional cases may also be used in this process, for example cases that represent previously failed solutions. Past failures may be stored as separate failure cases, indexed by remindings to failed solutions, or they may be parts of cases that represent the entire problem solving experience.

A solution derived from case-based reasoning is, in general, not reliable without subsequent justification. Before a solution is applied, it is therefore checked to see if it violates important constraints, contradicts given assumptions, leads to potentially risky conditions, etc. Modifying an existing solution by constructing a new - not previously seen - solution is in general difficult. If it is done automatically (as in CASEY, see chapter 4.4), the system needs a strong and reliable knowledge model. An alternative approach is that the system suggests a modification and leaves the justification to the user (as in PROTOS, see chapter 4.3). A third approach - and the one taken in the research described in this dissertation - is to combine the two extremes by allowing the system to ask the user only if it is not able to produce a sufficiently strong justification on its own.

3. Attempt to solve the new problem and evaluate the result.
The solution built in the previous step is applied to the current problem, and the result is evaluated. The solution may be evaluated by simulation in a separate model, or by being applied in the real world. In the latter case there may be a significant delay between the time the solution is proposed and the time the result has been evaluated. The case-based reasoner should know whether a solution has been successful, whether it has been a failure or whether it has not been evaluated yet. This fact is important when it later comes to assessing the usefulness of a case for solving a new problem. Hence, the evaluation part is crucial to the system’s learning process: If a solution turns out to be successful, the system must be told so. It may then update the case information accordingly, so the confirmation of success may be used in assessing this and similar solutions in the future. If a solution fails, the system must learn from its failure in order not to make the same mistake the next time.

Some case-based reasoners are also able to recover from a failure (e.g. CHEF, see chapter 4.5). When feedback about a failed solution is received, these systems attempts to explain the
reason for the failure. This may cause the system to change its general knowledge - or at least suggest such changes to the user. In this way, not only the actual case is updated, the learning also involves refinement of a system’s more general background knowledge.

4. Learn by extracting and storing experience from the problem solving steps.
This is the process of integrating the new problem solving episode into the existing knowledge. It involves selecting which information from the case to retain, in what form to retain it, and how to index the case for later retrieval from similar problems. In CBR the case base is updated no matter how the problem was solved. If it was solved by use of a previous case, a new case may be built or the old case may be generalized to subsume the present case as well. If the problem was solved by other methods, including asking the user, an entirely new case will have to be constructed. In knowledge-intensive CBR, modifications may also be made within the general conceptual knowledge model. Such modifications may result from inadequate or erroneous explanations, or from a solution that failed due to errors in the general knowledge model. Thus, with a proper interface to the user (whether a competent end user or an expert) a system may incrementally extend and refine its general knowledge model, as well as its memory of past cases, as more problems are solved (as done, e.g., in the Protos system).

The learning from success or failure of the proposed solution is triggered by the outcome of the evaluation performed in step 3. The fastest and most effective learning takes place when a solution proposed by the system turns out to fail. A knowledge-intensive approach to correcting the failure is that the system tries to explain why the solution failed by identifying possible weak steps in the reasoning process leading to the solution. In this credit or blame assignment task, a CBR system may utilize its particular advantage compared to other machine learning methods: It may retrieve a similar failure made in the past and use or adapt the explanation of the past failure to explain and correct the present failure (as done, e.g., in the CHEF system).

Figure A1.1, illustrates a CBR process, as presented in [Simpson-85]. With reference to the four steps just described, Step 1 corresponds to the UNDERSTAND PROBLEM and the FIND MATCHING CASE boxes in the figure. The figure illustrates that the retrieval of similar problems may be used to increase problem understanding (upper right box), for example by setting up expectations that need to be checked for the present problem. Step 2 is contained within the GENERATE SOLUTION box, constructing a solution to the new problem from the solution of a previous case. Step 3 is covered by the TEST PREDICTIONS box. This task takes feedback from the surrounding environment (e.g. the user). In Step 4 the system learns by trying to understand a failure, if there is one, suggesting an improved solution based on this improved understanding, and testing the improved solution by attempting to solve the
problem again (as indicated by the arrow from Step 4 back to Step 1). The system learns from a successful solution by retaining the case and updating the case index structure. Permanent storing of learned knowledge takes place in the two UPDATE MEMORY boxes, i.e. each time a failure is encountered or when a successful solution has been generated. As the lower left box indicates, failures may be indexed and stored as cases (either as separate failure cases or within total-problem cases). When a failure is encountered, the system may get a reminding to a previous similar failure, and use the failure case to improved understanding of - and correct - the present failure.

A1.3 Organization of the Cases in Memory

A case-based reasoner is heavily dependent on the structure (and content, of course) of its case memory. Since a problem is solved by recalling a previous experience suitable for solving the new problem, the case search and matching processes need to be both effective and reasonably time efficient. Further, since the experience from a problem just solved has to be retained in some way, these requirements also apply to the method of integrating a new case into the memory. How the cases should be organized and indexed, e.g. how cases should...
be described and represented, how they should be inter-related and how they should relate to entities like input features, solutions, and problem solving states, have been - and still is - a major research topic. In the following subchapter, two influential case memory models are briefly reviewed: The dynamic memory model of Schank and Kolodner, and the category-exemplar model of Porter and Bareiss.

### A1.3.1 The Dynamic Memory Model (Schank/Kolodner)

One of the first systems to be called a case-based reasoner was built by Janet Kolodner. The system, CYRUS, was based on Schank's "dynamic memory" model [Schank-82], and was basically a question-answering system with knowledge of the various travels and meetings of former US Secretary of State Cyrus Vance. The case memory model developed for this system has later served as basis for several other case-based reasoning systems (including MEDIATOR [Simpson-85], CHEF [Hammond-86b], JULIA [Kolodner-87, Hinrichs-88], CASEY [Koton-89], ARC [Plaza-90]).

The case memory is a hierarchy of interlinked structures, called "episodic memory organization packets" (E-MOPs), or "generalized episodes" (GEs). The basic idea is to organize specific cases which share similar properties under a more general structure. A generalized episode contains three different types of objects: Norms, indices, and cases. Norms are features common to all cases indexed under a GE. Indices are features which discriminate between a GE's cases. An index may point to a more specific generalized episode, or directly to a case. An index is composed of two terms: An index name and an index value. Figure A1.2 illustrates this structure.

The entire case memory is a single discrimination tree where the nodes are generalized episodes (containing the norms), index names, index values and cases. An index value may only point to a single case or a single generalized episode. If - during storing of a case - two cases (or two GEs) end up under the same index, a new generalized episode is automatically created. Hence, the memory structure is dynamic in the sense that similar parts of two case descriptions are dynamically generalized into a GE, and the cases are indexed under this GE by their difference features.

A case is retrieved by finding the GE with most norms in common with the matching problem. Indices under that GE are then traversed in order to find the case which contains

---

1 Other models include Rissland and Ashley's HYPO system [Rissland-87] in which cases are grouped under a set of domain-specific dimensions, and Stanfill and Waltz' MBR model, designed for parallel computation rather than knowledge-based matching.

2 This work was part of her Ph.D. thesis at Yale University, supervised by Roger Schank. The Schank/Kolodner memory structure is based on Schank's MOP theory of human problem solving and learning (which in turn is a refinement of his Script theory).
most of the additional problem features. Storing of a new case is performed in the same way, with the additional process of dynamically creating generalized episodes, as described above. Since the index structure is a discrimination tree, a case (or pointer to a case) is stored under each index that discriminates it from other cases. This may easily lead to an explosive growth of indices with increased number of cases. Most systems using this indexing scheme therefore put some limits to the choice of indices for the cases. In CYRUS, for example, only a small vocabulary of indices is permitted.

---

The norms part of a generalized episode contain abstract general information that characterize the cases organized below it.

---

The primary role of a generalized episode is as an indexing structure for matching and retrieval of cases. The dynamic properties of this memory organization, however, may also be viewed as an attempt to build a memory structure which integrates knowledge from specific episodes with knowledge generalized from the same episodes. It is therefore claimed that this
knowledge organization structure is suitable for learning generalized knowledge as well as case specific knowledge, and that it is a plausible - although simplified - model of human reasoning and learning.

A1.3.2 The Category & Exemplar Model (Porter/Bareiss)

The PROTOS system, built by Ray Bareiss¹, proposes an alternative way to organize cases in a case memory. The psychological and philosophical basis of this method is the view that 'real world', natural concepts should be defined extensionally. Several researchers (e.g. Smith and Medin [Smith-81]) have shown that natural concepts (concepts that are part of the natural world - like bird, orange, chair, car) are polymorphic. That is, their instances may be categorized in a variety of ways, and it is extremely difficult to come up with classical definitions (i.e. sets of necessary and sufficient features) of such concepts. An answer to this problem, therefore, is to represent a concept as a category defined by its set of instances - called exemplars (or cases). Further, different features are assigned different importances in describing an exemplar's membership to a category. Any attempt to generalize a set of exemplars should - if attempted at all - be done very cautiously. This fundamental view of concept representation forms the basis for this memory model.

The case memory (the collection of exemplars) is embedded in a network structure of categories, cases, and index pointers. Each case is associated with a category. An index may point to a case or to a category. The indices are of three kinds: Feature links (called remindings; features pointing to a case or a category), case links (pointers from a category to its associated cases), and difference links (pointers from a case to another case with a number of features in common, and with a set of different features suitable for discriminating between the cases). A feature is, generally, described by a name and a value. A category's exemplars are sorted according to their degree of prototypicality in the category. The case most representative of the category (i.e. the most typical case) is ranged highest.

Figure A1.2 illustrates a part of this memory structure, focused around a single category. Within this memory organization, the categories are inter-linked into a semantic network, which also contains the features and intermediate states (e.g. subclasses of goal concepts) referred to by other terms. This network represents a background of general domain knowledge. However, all knowledge associating problem features directly with solutions are contained in the cases.

¹The PROTOS system is part of his Ph.D. dissertation at The University of Texas, Austin, supervised by Bruce Porter.
A past case is retrieved by combining the input features of a problem case\(^1\). The cases or categories that most closely match the problem features are selected (according to some match criteria). If a category is most strongly reminded of, the links to its most prototypical cases are traversed, and these cases are returned. The best matching case is then determined by evaluating the degree of match more closely. This is done by an attempt to generate explanations to justify non-identical features, based on the knowledge in the semantic network. If a match turns out not to be strong enough, an attempt to find a better match by following difference links to closely related cases is made. An input case is classified as an instance of the same category as its closest matching case.

![Figure A1.2: The Structure of Categories, Features and Exemplars](image)

The figure illustrates the linking of features and cases (exemplars) to categories in the Porter/Bareiss model. The unnamed indices are remindings from features to a category.

A new case is stored by searching for a matching case, and by establishing the appropriate feature indices. If a case is found with only minor differences to the input case, the new case may not be retained or the two cases may be merged by following taxonomic links in the semantic network.

### A1.4. Concluding Remarks

Case-based reasoning represents a different paradigm both to reasoning and learning than what the AI community traditionally have been focusing. Reasoning from past cases is a major process of human reasoning, and the rapid increase in interest for exploring it as a AI method may give useful results both for the construction of knowledge-based computer systems and for the study of human reasoning and learning.

---

\(^1\)If a feature is unknown to the system, the semantic network needs to be updated with a description of the new feature which relates it to the existing terms in the network.
Appendix 2

A Summary of the CreekL Knowledge Representation Language

A2.1. General

The CreekL language is a flexible frame-based representation language, implemented as a set of CommonLisp procedures. To enable a thorough representation of knowledge, CreekL facilitates explicit definitions of relations, and symbolic values. For example, if the user want to introduce a new slot, called has-color, on the frame car, the system automatically creates the frame has-color and gives it a slot called used-to describe with the value car. The user may enter additional slots on the has-color frame in order to describe what it means to have color. The system also automatically creates a frame for each symbolic values\(^1\) entered in a slot. These mechanisms are illustrated later.

The inference methods operating on this semantic network of frames are typical frame language methods like property inheritance, frame matching (concept recognition), and constraint enforcement (constraint propagation).

Basic representation paradigm:

- Concepts are represented as a network of frames
- Concepts are either entities or relations
- Entities are physical or abstract objects of a domain
- Relations are named associations (links) between concepts

Corresponding to the two types of concepts, there are two types of frames in CreekL: 
Entity-frames and Relation-frames.

---
\(^{1}\)A symbolic value is a value that is not a number, a text string or a lisp-function.
A *frame* is a structure containing a framename and set of slots.

A *slot* is a structure containing a slotname and a set of facets.

A *facet* is a structure containing a facetname and a set of value expressions.

A *value expression* is a structure containing the value itself (called 'proper-value'), a justification for the value, a source of the value, a timestamp, and an optional parameter:

\[
\begin{align*}
\text{<frame>} & := \text{<framename \{slot\>}}, \\
\text{<slot>} & := \text{<slotname \{facet\}>}, \\
\text{<facet>} & := \text{<facetname \{value-expression\}>}, \\
\text{<value-expression>} & := \text{<proper-value value-justification value-source value-timestamp <optional>>}
\end{align*}
\]

A slot, then, represents a relation that associates a concept (frame) with another concept (value). Some relations are pre-defined, for example the following four structural relations:

*subclass-of*  *instance-of*  *part-of*  *member-of*

They represent subclass specialization, instantiation, decomposition and set membership, respectively. The inverse relations are defined as well, e.g. has-subclass, etc.

Some relationships are generated automatically when a frame is created and when a value is entered:

- **created-by** - sets the value to the registered user name when a frame is created
- **used-to-describe** - used for relation-frames as a pointer to concepts described by the particular relation
- **value-of** - (see paragraph below)

When a concept-relation-concept relationship is established by a frame-slot-value triplet, an inverse relationship is stored on the frame representing the value in the triplet. If a relation has an inverse relation defined, the inverse relation is used. If not, a general relation named *value-of* is used, which stores the frame.slot pair (still referring to the initial triplet). For example, if the following triplet is entered:

\[
\text{my-car has-color red}
\]

a frame called *red* is defined. If the *has-color* frame has an *has-inverse* slot, the value of that slot (e.g. *color-of*) becomes the slotname for the value *my-car* on the *red* frame. If no inverse is
defined for the has-color relation, the value-of relation is used. Thus, red will get one of the following slots:

red  color-of  my-car
red  value-of  my-car.has-color

As the last line illustrates, a dot - "." - is a general delimiter between substructures of a frame. A particular value may be refereed to by: <framename>.<slotname>.<facetname>.<value>

In the examples so far, only one type of slot values have been used, i.e. the 'actual value' of a slot. Facets enables us to express other type of slot contents, like value constraints, deamons - i.e. functions to be called when a slot is referred to, etc. A facet, thus, describes the type or role of a relationship between a framename and the contents of a slot. In the examples above, only one facet was used, namely the value facet. Further only the first part of a value-expression was used, namely what has been called the 'proper value', i.e. the value without annotations.

The following facets are defined:

value    - the actual value
default   - a default value
value-reference   - a value.reference-concept pair that a qualitative value is compared to (e.g. a dog is big compared to pet-animals, but small compared to circus-animals)
value-dimension  - a concept or string expressing the dimension of a quantitative value (e.g. mg-per-l, "seconds")
value-constraint   - a procedure that defines a constraint on the slot's values; values of this facet are general lisp expressions
value-class   - a list of concept classes that any value has to be a specialization of, i.e.
value-set   - a list of legal slot values
inheritance-type   - control the inheritance of a slot.value pair
if-needed, if-added, if-removed, if-modified - procedural attachments that are executed when certain operations are performed on a slot's value

The proper-value part of a value-expression may have one of four value types:
concept - atomic term
number - a numeral
string - a string of letters contained within quotes
lisp-expression - a lisp function

The value may be a single value or a list of values.
The basic frame operation functions are described below. They all operate on frames in
general, i.e. both entity- and relation-frames.

Structural relations between representational terms are defined within the system itself. There
is a separate frame description of the system internal concepts like frame, slot, has-part, value-
expression, etc. Frame, e.g., has subclass case-frame, which is a particular frame intended for
storing problem solving cases. All these internal- structure-and-organisation concepts are
subclasses or instances of iso-thing. Iso-thing is a subclass of the top level concept: thing. The
other subclass of thing is called pod-thing. Concepts of the domain are either subclasses or
instances of pod-thing. Examples of iso-things:

facet
part-of value slot
subclass-of value iso-thing
has-part value value-expression
has-subclass value constraint-facet value-facet daemon-facet

has-instance
used-to-describe value book-keeping-frame constraint-facet daemon-facet transitive-relation
value-facet knowledgebase
instance-of value transitive-relation
has-inverse value instance-of

if-needed
instance-of value daemon-facet

A2.2. Implementation

The system is written in CommonLisp on a TI Explorer, and has been ported to Sun Common
Lisp. It should be runable under any CommonLisp system. The Explorer implementation has
interface routines to the METATOOL graphical structure editor, enabling graphical
construction and visualization of concept structures connected by the four structural relations
(listed above). CreekL is a modification and extension of METATOOL’s representation
language.

A frame is implemented as a nested association list, and placed on the property frame of the
symbol equal to the frame’s name.
For some functions, macros are defined to make terminal input more convenient. Macros only accept atomic parameters; if a value expression argument is a list, the function form should be used.

A2.3. Loading and saving a knowledge base

The function \(\text{(load-kb } \text{kb-name)}\) loads the named knowledgebase from the knowledgebase directory (preset by the user).

\(\text{(load-kb } \text{isopod)}\) load the system knowledgebase.

The function \(\text{(save-kb } \text{kb-name)}\) saves the knowledgebase. This function call will not save the isopod knowledge base. An optional third argument may be set to T (true) if the isopod base should be save together with the domain knowledge base.

A2.4. Building of concepts

**Functions**

\(\text{(f-put framename slotname facetname value)}\)

The basic function for storing data in a frame’s slot. Adds a new value, and creates the frame if it does not already exist.

\(\text{(f-set framename slotname facetname value)}\)

Same as f-put, except that the value overrides any existing values.

\(\text{(f-define-frames)}\)

Interactive prompting for definition of frames

\(\text{(f-define-or-merge-frame frame)}\)

The argument is an entire frame structure, i.e. a nested list of slots, facets, and value expressions. When a knowledge base is saved to a file, it is stored as a set of \(\text{(f-define-or-merge-frame .....) calls.}\)

**Macros**

\(\#>\text{framename.slotname.facetname.value} \quad \text{- same as f-put}\)

\(\#>\text{framename.slotname.value} \quad \text{- f-put with facetname = value}\)

**Examples:**

\(> \text{(f-put } \text{’car ’subclass-of } \text{’value } \text{’vehicle)}\)

VEHICLE
(f-put 'car 'has-number-of-wheels 'default 4)
4

(f-put 'car 'has-number-of-seats 'value-constraint '((and (> value 0) (< value 9))))
(AND (> VALUE 0) (< VALUE 9))

The constraint language is CommonLisp - making it expressive and efficient, but not necessarily user friendly. The term VALUE in the expression above is a place marker to be replaced by the actual value before the constraint expression is evaluated (see example in the Constraints Enforcement section).

(f-define-frames)
framedefinition

(f-define-or-merge-frame (bodywork (part-of (value (car nil agnar 2835469663 nil)))
(created-by (value (agnar nil agnar 2835469663 nil))))
BODYWORK

A2.5. Removing frame contents

Functions:

The following functions remove a concept - or parts of it - from the knowledgebase.

(f-remove-frame framename)
(f-remove-slot framename slotname)
(f-remove-facet framename slotname facetname)
(f-remove framename slotname facetname value)

Macros:
Appendix 2 - The CreekL Language

#<framename - removes a frame from the knowledgebase
#<framename.slotname - removes a slot
#<framename.slotname.facetname - removes a facet
#<framename.slotname.facetname.value - removes a value

Note that the a.b.c form for the #< macro refers to a facet, while the a.b.c form for the #> macro refers to a value for the value facet. The reasons are practical: The value facet is by far the most used facet for value input, hence the a.b.c form of #>. The #< macro is a 'dangerous' operation, since it irreplacably deletes something. Hence, this should be used with caution, and by explicitly specifying the facet.

A2.6. Changing frame contents

Functions:

(f-change-framename oldframename newframename)
(f-change-slotname framename oldslotname newslotname)
(f-change-facetname framename slotname oldfacetname newfacetname)
(f-change framename slotname facetname oldvalue newvalue)

No macros are defined for the modification functions.

A2.7. Displaying concepts

Functions

(f-get-frame framename) - displays the list structure of a frame

Parts of a frame structure may be retrieved by the functions:

(f-get-slot framename slotname) - displays the list structure for a slot
(f-get-facet framename slotname facetname) - displays the list structure for a facet

Macros

The macro #P<expression> pretty-prints a lisp expression.

#?framename - identical to (f-get-frame framename)
#?framename.slotname - retrieves evaluated value content, see below (f-get)
#?framename.slotname.facetname - identical to (f-get-facet framename)
#@ - identical to #P#, i.e. pretty-prints the structures described above
#Lframename - lists a frame’s local contents in a formatted way

The whole knowledge base may be displayed by:
(show-kb) - displays list structure
(list-kb) - displays formatted frames
(list-pod-kb) - displays domain related knowledge, includes all relations

Examples:

> #P(f-get-frame 'car)

(car (subclass-of (value (vehicle nil agnar 2836323995 nil))))
 (created-by (value (agnar nil system 2836323995 nil))))
 (has-number-of-wheels (default (4 nil agnar 2836324041 nil))))
 (has-number-of-seats (value-constraint ((and (> value 0)
                                         (< value 9))
                                         nil agnar 2836325691 nil))))
 (has-function (value (transportation-of-people nil agnar 2836326140 nil))))
 (has-subclass (value (sports-car nil system 2836326630 nil))))
 (has-part (value (bodywork nil agnar 2836408272 nil))))
 (value-of (value (sports-car.has-size))))

> #Lcar

car subclass-of value vehicle
has-number-of-wheels default 4
has-number-of-seats value-constraint (and (> value 0) (< value 9))
has-function value transportation-of-people
has-subclass value sports-car
has-part value bodywork

What we have entered in the examples so far, together with knowledge-base information and predefined relations, may be viewed by calling:

> (list-kb)

ENTITY CONCEPTS:

bodywork part-of value car

car subclass-of value vehicle
has-number-of-wheels default 4
has-subclass value sports-car
value-of value sports-car.has-size
has-part value bodywork
has-number-of-seats value-constraint (and (> value 0) (< value 9))
has-function value transportation-of-people
small
  value-of value sports-car.has-size

sports-car
  subclass-of value car
  has-size value small
  value-reference car

transportation-of-people
  value-of value car.has-function

vehicle
  has-subclass value car

RELATION CONCEPTS:

caused-by
  instance-of value transitive-relation
  has-inverse value causes

causes
  instance-of value transitive-relation
  has-inverse value caused-by

function-of
  instance-of value transitive-relation
  has-inverse value has-function

has-activated-kbs
  used-to-describe value knowledgebase

has-filename
  used-to-describe value isopod

has-function
  instance-of value transitive-relation
  has-inverse value function-of
  used-to-describe value car

has-instance
  used-to-describe value book-keeping-frame constraint-facet daemon-facet transitive-relation
  value-facet knowledgebase

instance-of value transitive-relation
  has-inverse value instance-of

has-member
  instance-of value transitive-relation
  has-inverse value member-of

has-name
  used-to-describe value current-kb

has-number-of-seats
  used-to-describe value car

has-number-of-wheels
  used-to-describe value car

has-part
  used-to-describe value facet frame slot value-expression knowledgebase car
A2.8. Constraints Enforcement

The constraint enforcement mechanism checks the value-constraint and value-class facets of the slot involved. Constraints are inherited down has-subclass and has-instance links:

> #>my-car.instance-of.sports-car
SPORTS-CAR
> #>my-car.has-number-of-seats.25

--> The value entered is not a legal value, as specified in the
value constraint expression. It has therefore been rejected.

  Framename: MY-CAR Value: 25
  Constraint-expression: ((AND (> (QUOTE 25) 0) (< (QUOTE 25) 9)))

25

More information about the conflict is stored in the conflict-state frame:

> #Lconflict-state

conflict-state
message-p       value t
last-conflict-time value November 22nd 1989 at 17:20:30
last-instance-number value 63
current-conflict value general-constraint

T

The value of the message-p slot determines whether the message above shall be printed or not. The actual conflict-instance may be viewed by concatenating the name conflict- with the number in the last-instance-number slot:

> #Lconflict-63

conflict-63
conflict-type       value general-constraint
frame-in-conflict  value my-car
slot-in-conflict   value has-number-of-seats
facet-in-conflict  value value
value-in-conflict  value 25
constraint-expression value (and (> (quote 25) 0) (< (quote 25) 9))
T
> #>my-car.has-number-of-seats.5

5

The value 5 is accepted.

A2.9. Retrieval of value content

The basic content retrieving function is f-get. A first attempt is made to find a value under the value facet. If not successful, the default facet is tried, followed by the if-needed facet - whose function gets evaluated.

>(f-get 'car 'has-number-of-wheels)
(4)

Macros

#?framename.slotname    - equals (f-get framename slotname)
#@framename.slotname    - equals #P#?framename.slotname
A2.10. Value inheritance

f-get may also be asked to traverse inheritance links in order to look for values. There are three types of f-get’s that work along subclass and instance links:

- **f-get-z** - first look for a *value facet*, then for a *default facet*, then move a level up and repeat the process, etc.
- **f-get-n** - first look for a *value facet* on the local frame, then move upwards looking for value facets, then down again to the local frame and repeat the process for the *default facet*.

These functions return the first value found. The most useful function for retrieving inherited values, however is f-get-i:

f-get-i searches all instance-of and subclass-of relations recursively, and concatenates the inherited values. It only includes the most specific concept along each sub-branch. It performs multiple inheritance, and it excludes values inherited along one branch that violate constraints inherited along another branch.

f-get-frame-i retrieves a frame with all its slots and slot values, including inherited slots and values.

**Macros:**

- `#Iframename.slotname` - formatted output of (f-get-i framename slotname)
- `#Vframename` - formatted output of (f-get-frame-i framename)

**Inheritance examples:**

- `> #>vehicle.has-function.transportation` : TRANSPORTATION
- `> #>truck.subclass-of.vehicle` : VEHICLE
- `> #>truck.has-function.transportation-of-goods` : TRANSPORTATION-OF-GOODS
- `> (f-get-i 'my-car' 'has-function)` : (TRANSPORTATION-OF-PEOPLE TRANSPORTATION)

We now make transportation a superclass of transportation-of-people:

- `> #>transportation-of-people.subclass-of.transportation` : TRANSPORTATION
- `> (f-get-i 'my-car' 'has-function')` : (TRANSPORTATION-OF-PEOPLE)

Only the most specific concept is inherited.
An example of multiple inheritance:

> #>sports-car.subclass-of.sporting-gear
SPORTING-GEAR

> #>sporting-gear.has-function.sporting-assistance
SPORTING-ASSISTANCE

> (f-get-i 'my-car 'has-function)
(TRANSPORTATION-OF-PEOPLE SPORTING-ASSISTANCE)

Inherited values are subject to the same constraint checking as done when a value is explicitly entered into the local frame. In case of conflicts between two inherited values, an implemented system would typically look for additional evidence or support to decide which one to choose, depending on the goal and state of the system at the point of conflict. If unable to decide for itself, a typical action would be to ask the user.

The inheritance mechanism described above is the general method that operate if no particular inheritance restrictions is specified. Particular inheritance restrictions may be specified in the inheritance-type facet. Two facet values are recognized by the system:

- not-inheritable - never inherited, used for local values
- forced-inheritance - always inherited, may not be over-written by subclasses or instances

### A2.11. Traversing Relational Links

A set of functions are defined to retrieve concept structures and to check properties of concepts. The most general is:

\[
(f\text{-get-structure} \text{ conceptname relationname})
\]

The function returns a nested list structure of sub- and super-concepts according to the relation specified.

\[
(f\text{-get-super-structure} \text{ conceptname}) \text{ traverses both instance-of and subclass-of links.}
\]

For example:

> (f-get-super-structure 'my-car)
(MY-CAR (SPORTS-CAR (CAR (VEHICLE)) (SPORTING-GEAR)))
(f-get-sub-structure conceptname) correspondingly picks up concepts along has-subclass and has-instance relations.

A2.12. Spreading of activation

A portion of the concept structure may be marked as activated by spreading of activation from a start concept. Activation is spread along a user-specified set of relation (held in the value-facet of activating-relation.has-instance). The default set of spreading relations are the instances of transitive-relation, i.e:

subclass-of, instance-of, part-of, member-of, function-of, caused-by

and their inverses. Spreading is done exhaustively along the defined set of relations by calling:

(spread-activate start-concept)

Example:

> (spread-activate 'sports-car)
(TRANSPORTATION-OF-GOODS TRUCK TRANSPORTATION VEHICLE BODYWORK TRANSPORTATION-OF-PEOPLE SPORTING-ASSISTANCE SPORTING-GEAR CAR MY-CAR SPORTS-CAR)

> #Ltruck

truck

subclass-of value vehicle
has-function value transportation-of-goods
state value active

Activated concepts get the value activated in their has-activation-status.value facets. All active concepts are held in the *activated-concepts* list.

De-activation is done in a similar fashion as activation:

(spread-deactivate start-concept)

Deactivated concepts get the value deactivated in their has-activation-status.value facets, and are removed from the *activated-concepts* list.
Appendix 3

A Description of Creek Terms

Some frequently occurring terms referring to types of knowledge and inference are described below:

**case**
An instance of a problem description, i.e. the description of a concrete problem to be solved (input case), in the process of being solved, or having been solved (past case).

**case base**
The permanent store of past cases, i.e. the memory.

**concept**
A concept in Creek is an explicit containment describing a part of the real world. A concept is defined by a concept name and a list of typical relationships with other concepts.

**conceptual knowledge**
Descriptive, definitional knowledge. Definitions of entities in the real world by various relationships with other entities. A significant number of relationships expresses theoretical or principled, fundamental knowledge. Conceptual knowledge is synonymous to deep knowledge.

**deep knowledge**
Synonymous to conceptual knowledge. This implies that deep knowledge not exclusively expresses 'deep' relationships. Deep knowledge always has a level of relationships of this kind, although the total set of relationships often represents a continuum between deep and shallow knowledge.
fact
A unit of knowledge and information. A fact may be expressed as a relationship, or explicitly defined as a concept. Fact is synonymous to proposition.

feature
In general, a concept descriptor, a property - i.e. what is represented as a slot in a frame. In Creek features usually refer to descriptors of problems and cases (e.g. problem findings, solution hypotheses, intermediate states, the contents of case slots).

finding
A finding is a type of feature which refers to input descriptors of a problem (or a case). A finding may have been directly observed (observed finding) by measurements or human perception, or it may have been derived or transformed from observations (inferred finding).

general knowledge
Also referred to as generalized knowledge. Knowledge which is general and abstract in the sense that it refers to concept classes rather than instances. For example, "a car has four wheels" is general knowledge, while "my car has four wheels" is not. The latter is called special knowledge. General knowledge include deep, conceptual knowledge as well as more 'shallow' and operational knowledge, e.g. heuristic knowledge.

generalized knowledge
See general knowledge.

general domain knowledge
See general knowledge.

heuristic knowledge
A shallow type of general knowledge. Expresses knowledge acquired by generalizing from experiential knowledge and/or from conceptual knowledge.

inference type
A paradigm for deriving a proposition from another proposition in an intelligent agent. There are three inference types: abduction, induction and deduction. They may be used separately or combined in an inference method.
inference method
An basic method for deriving a new proposition based on existing ones. An inference method is based on one or more inference type. Examples of inference methods are forced inheritance, plausible inheritance, forward-chaining of rules, backward-chaining of rules, spreading activation, concept matching.

inference structure
See reasoning model.

memory
A structure of previously experienced cases, their links to the general knowledge model, and their feature indices. (Note that this use of the term memory is different from its use in some areas of computer science, where it denotes an internal, fast-access data store, as opposed to the data store on peripheral units like disks.)

proposition
See fact.

reasoning method
A method for doing reasoning according to a particular paradigm, based on one or more reasoning types. The reasoning methods in Creek are combinations of case-based reasoning, model-based reasoning and rule-based reasoning.

reasoning model
A model containing a reasoning structure (cf.) realized by a reasoning method.

reasoning structure
A high level, sequential structure of a reasoning process. A reasoning structure combined with a reasoning method that enables a realisation of the reasoning structure, is referred to as a reasoning model. For example, Clancey’s model of heuristic classification is basically a reasoning structure, but it becomes a reasoning model when supplied with a rule-based reasoning method.

reasoning type
A paradigm for performing reasoning tasks in an intelligent agent. There are three reasoning types in Creek: Case-based reasoning, model-based reasoning and rule-based reasoning. One or more reasoning types are realized into operational procedures called reasoning methods.
relationship
An expression of a fact as a triple containing a target concept, a relation, and a value. All relationships in Creek are binary relationships.

reminding
A single reminding is a pointer to a case from a finding. A single reminding consists of a finding name, a case name and a relevance factor expressing the strength of the reminding with respect to the case. Single remindings may be computed into a combined reminding if several findings point to the same case.

shallow knowledge
Knowledge expressed by relations that associate problem descriptors (findings or closely related intermediate states) with other problem descriptors or problem solutions. The relationships of shallow knowledge are abstracted to the level that they express mere associations, and not the underlying theories. Shallow knowledge may be general (as expressed by, e.g., heuristic rules) or special (as expressed by previously solved cases).

special knowledge
Also referred to as specialized knowledge. Knowledge which is special in the sense that it refers to particular instances rather than concept classes. A collection of cases, for example, represents points of special knowledge in the knowledge space, unlike the areas of general knowledge represented by, e.g., heuristic knowledge.

specialized knowledge
See special knowledge.

target concept
Denotes the role of a concept as the object being described, i.e. the subject or target of a definition, as opposed to a value concept.

value concept
Denotes the role of a concept as the object contributing to the description of another concept. The other object (the one being described) is referred to as the target object.
References

Aakvik-88

Aakvik-89

Aamodt-87

Aamodt-89a

Aamodt-89b

Aamodt-90a

Aamodt-90b

Achinstein-83

Allen-83

Althoff-89

Anderson-83

Anderson-85
REFERENCES

Bareiss-88a

Bareiss-88b

Bareiss-89

Barletta-88

Bennett-85

Bergheim-88

Blix-89

Bobrow-77

Bobrow-84

Bonissone-85

Brachman-79

Brachman-83
Brachman-83b

Brachman-85

Brachman-85b

Branting-88

Branting-89

Brazdil-90

Breuker-87

Breuker-89

Brown-82

Buchanan-84
Bruce Buchanan, Edward Shortliffe (eds.): *Rule base expert systems; the MYCIN experiment of the Stanfor Heuristic Programming Project*. Addison-Wesley, 1984.

Buchanan-86

Bundy-85

Burstein-89
REFERENCES

Carbonell-83

Carbonell-86

Carbonell-89

CBRW-89

Chandrasekaran-83

Chandrasekaran-86

Chandrasekaran-87

Chandrasekaran-89

Charniak-84

Clancey-84a

Clancey-84b

Clancey-85

Clancey-86

Clancey-89
REFERENCES

Cohen-85

Cohen-88

Cohen-88b

Cohen-89

Collins-85

Davis-82a

Davis-82b
Randall Davis: Expert systems: Where are we? And where are we going from here? MIT, AI Laboratory, AI Memo No. 665, June 1982.

Davis-83a

Davis-83a

DeJong-86

DeJong-88

Dietterich-86

Doyle-86

Doyle-79
Dreyfus-86

Duda-79

Duran-88
Model of Diagnosis. Internal note, Department of Computer Sciences, University of Texas at Austin, January 20, 1988.

Eggen-90

Eshelman-87

Expert-89

Feltovich-84

Fink-87

Forbus-86

Genesereth-87

Gentner-83

Ginsberg-85

Greiner-80
Gruber-88

Guha-90

Hall-89

Hammond-86a

Hammond-86b

Hammond-87

Hammond-88

Harmon-88

Harmon-89

Hayes-85

Hempel-65

Hinrichs-88

Hinrichs-89

Hobbs-85

Holland-86
Holte-88

Jansson-86

Johnson-88

Johnson-Laird-83

Josephson-87

Kass-88

Kedar-Cabelli-86

Keravnou-86

Klinker-87

Kodratoff-87a

Kodratoff-87b

Kodratoff-88

Kodratoff-89
Kolodner-83a  

Kolodner-83b  

Kolodner-86  

Kolodner-87  

Kolodner-88  

Koton-88  

Koton-89  

Koton-89b  

Koulitchev-90  

Kuczora-89  

Kuipers-87  

Laird-86  

Larssen-90  
REFERENCES

Leake-88

Lenat-86
Doug Lenat, M. Prakash, Mary Shepherd: CYC - using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. AI Magazine, Winter 1986. pp. 65-85

Lenat-87

Lenat-89
Doug Lenat, R. Guha: Building Large Knowledge-Based Systems; Representation and Inference in the CYC Project. Addison-Wesley, 1989.

Levesque-85

Littman-89

Long-86

López-90

Marcus-87

Mark-89

Marty-89

Mayer-83

McCarthy-69

McDermott-86
REFERENCES

McDermott-87

McDermott-88

Michalski-83

Miller-56

Minsky-75

Minsky-88

Minton-89

Mitchell-82

Mitchell-85

Mitchell-86

Morik-89

Murray-88a

Murray-88b
REFERENCES

Murray-89

Newell-72

Newell-82

Nordbø-89

Nordbø-89b

Nordbø-90

Nutter-87

Offutt-88

Ogden-23

Plaza-90

Politakis-85

Pople-82

Porter-86

Porter-89
Porter-90

Quillian-68

Quinlan-86

Reddy-88

Rissland-87

Rissland-89

Rothman-88

Rumelhart-87

Schank-82

Schank-86a

Schank-86b

Schank-86c

Schank-89

Shastri-89
Lokendra Shastri: Default reasoning in semantic networks; A formalization of recognition and inheritance.
Sembugamoorthy-86

Sharma-88

Shepherd-88

Shortliffe-84

Simpson-85

Skalak-89

Smith-81

Smolensky-87

Sowa-84

Stanfill-88

Steels-85

Steels-86

Steels-87

Steels-89
REFERENCES


Sticklen-87

Swartout-85

Sølvberg-88

Sølvberg-88b

Sølvberg-91

Tecuci-88

Thagard-88

Thagard-89

Touretsky-87

Van de Velde-88a

Van de Velde-88b

Van de Velde-89
Van Marcke-88

Van Melle-80

Vestli-89

Waltz-75

Weiss-78

Wielinga-86

Wilensky-86

Winograd-75

Wittgestein-53

Zadeh-65