Expert Systems

Artificial Intelligence 2
Lecture Notes

by Mike Uschold

Contents

Chapter 1

General Information

1.1 Introduction

These lecture notes consist primarily of photocopies of the actual lectures given in the Expert Systems module. As such, they are not proper lecture notes. Proper lecture notes for this module have been available in past years. They were written by Peter Jackson, and are still availble in the South Bridge library. They have since been expanded and reworked into a book. For this reason, they are no longer made available to students for purchase.

I begin by describing the course objectives, a brief syllabus, and a brief description of some useful references for additional reading. The slides from the lectures follow.

1.2 Course Objectives

- To understand the fundmental issues involved in the building and using of Expert Systems
- To understand the essential aspects of the following three knowledge representation formalisms: logic, production rules, structured objects; To be capable of taking small examples and casting them into any of these representations; To understand the types of inference that are associated with each, and be able to apply different control strategies to accomplish the inferencing
- To be able to demonstrate an understanding of the following four expert systems: (MYCIN, INTERNIST, CENTAUR, and MECHO) by describing the essential knowledge and control structures used; To be able to describe how a toy problem would be solved using the same approach.
- To understand the major issues involved in knowledge acquisition.
- To understand how simple explanations of reasoning can be provided by expert systsms.
- To know what expert system shells and high level programming environments are, and some of the pros and cons for using each for building expert systems.

1.3 Syllabus

Introduction: Introduction to Expert Systems: Defniition, Motivation, Their historical development in AI; Essential features and issues. Introduction to Knowledge Representation and Inference. Production rules: Definition, Recognise-Act cycle, Forward chaining, Backward chaining, simple examples

Logic: Introduction to Logic, Propositional Logic, truth tables, logical implication. Predicate Logic, Practice in encoding things in logic (brief). Using Logic: General system structure, Backward and forward chaining control strategies, Unification Automated deduction, theorem proving, sumary logic.

MYCIN: This is the classic example of a production rule-based expert system. Background of the domain; System overview; Details of the knowledge base structures; The control mechanism: backward chaining; A detailed example of how a typical consultation proceeds. MYCIN model for reasoning with uncertainty, Summary, and Evaluation of MYCIN.

Explanation: question types; illustrate HOW and WHY explanations as goal-tree search; Problems and current research topics

Knowledge Acquisition: Initial creation and later refinement; types of bugs found in rule sets.

Structured Objects: Semantic Nets and Frames, Conclusion for knowledge representation formalisms. Compare and contrast pros and cons of each.

INTERNIST: Expert system for internal medicine. It uses structured objects as a central knowledge representation formalism; Study the control strategy used for problem solving

CENTUAR: Expert system which mixes rules and structured objects. Overview of system components and interactions. Details of knowledge structures, and control mechanism.

MECHO: Expert System for solving mechanics problems. It uses logic as its primary knowledge representation formalism. Details of knowledge structures, and control mechanism.

Conclusions: Review major issues, Building your own expert system; Available tools; Expert System Shells, High Level Programming Environments, Toolkits. State of the art; Summary and Conclusions

1.4 Suggested Reading Materials

1. **Waterman; "A Guide to Expert Systems"; 1986**
   While not very technically oriented, is indeed an excellent introductory guide to expert systems. There is an overview of the field, a point by point discussion of what the process of building an expert systems entails (including a section on common problems and pitfalls). There is also an extensive bibliography of some 200 expert systems reported in the literature. The indended audience is anyone interested in getting familiar with the basics of expert systems technology with an emphasis on finding out what if anything it can do for *you*, whether you are a bank manager, software specialist, or whatever. As such, it has very much an applied flavour rather than an theoretical one.

2. **Jackson, Peter; "Introduction to Expert Systems"; 1986**
   This text is considereably more technical than the Waterman text aimed at a different audience, namely third and fourth year university students, or first year postgraduates. He begins by giving an overview of artificial inteligence and describes how expert systems grew from this parent discipline. He then goes on to describe the three primary knowledge representation formalisms which have found use in expert systems: production rules, structured objects, and predicate logic. This is augmented by discussion of control strategies which may be used for each, and a number of practical issues which arise using plenty of examples. Following this, he describes in some detail a number of expert systems which exemplify the three formalisms. There is generally a fair bit of analytical discussion comparing the pros and cons of the techniques used by each system giving the reader a fairly good grasp of many of the practical and theoretical issues involved in building expert systems. At the end of each chapter, there are several very useful (some very substantial) exerises which if faithfully done would give the reader having finished the book, a rather solid grasp of most of the important issues in building expert systems both from practical and theoretical points of view.

   Overall I reccommend the text, which began life as class notes for the Expert Systems module of the second year undergraduate course at Edinburgh University, "Artificial Intelligence 2". This book still forms the basis for the course. One major complaint is his inconsistent treatment of the assumed competence and background of the reader. Sometimes very basic issues are well described but very often, he assumes too much of the reader. He uses lots of buzz-words and phrases that readers unfamiliar with AI can't be expected to be familiar with. Also, I would be wary of his treatment of AND/OR graphs insofar as its relationship to state space search. It is rather confused and in my opinion partially wrong.

3. **Hayes-Roth, Waterman, and Lenat (editors); "Building Expert Systems"; 1983**
   This book is misleadingly titled. It is not a text as such describing how to build expert systems; rather it presents an overview of the field at the time by over forty contributing authors. Noteworthy is the fact that it contains the first attempt at classifying the sort of tasks for which expert systems may be appropriate. Also, overviews of a dozen or so of the earliest and most influential tools for building expert systems are presented. The results of an experiment which used all of these tools on a single task are presented.

4. **Buchanan and Shortliffe; "Rule-Based Expert Systems (The MYCIN Experiments of the Stanford Heuristic Programming Project)"; 1985**
   This is a large work describing in depth the MYCIN experiments at Stanford. As this work has been extremely influential on the field overall, this book is worthwhile. However, insofar as it presents the views of only one research group, is not a general text on expert systems.

5. **Weiss and Kulikowski; "A Practical Guide to Building Expert Systems"; 1984**
   This is also somewhat mistitled. It describes the many experiments performed using EXPERT, an expert system developed by the authors at Rutgers. It is more a guide to building expert systems using the EXPERT formalism. It is much less comprehensive than the "Rule-Based Expert Systems ...", but nevertheless does address most of the major issues in building expert systems.

Chapter 2

Introduction

EXPERT SYSTEMS

- WHAT ARE THEY?
- MAJOR ISSUES
- KNOWLEDGE REPRESENTATION FORMALISMS
- EXAMPLE SYSTEMS
- BUILDING YOUR OWN EXPERT SYSTEM
- WHERE ARE WE NOW AND WHERE ARE WE GOING?

MOTIVATION

Human Expertise	Computer Expertise
perishable	
hard to transfer	
hard to document	
unpredictable	
expensive	
However, current state of the art is limited:	uninspired
	non-adaptive, won't readily learn
	narrow focus
	no commonsense knowledge

WHAT IS AN EXPERT SYSTEM?

* Applied AI

* Perform tasks requiring GENUINE HUMAN EXPERTISE

  eg: - MEDICAL DIAGNOSIS
  - TROUBLESHOOT TELEPHONE NETWORKS
  - PREDICT MINERAL DEPOSITS

  KNOWLEDGE INTENSIVE TASKS

* Must explain its reasoning

HISTORICAL DEVELOPMENT

* GENERAL (WEAK) METHODS
  - PROBLEM SOLVING
  - HEURISTIC SEARCH
  - MEANS-ENDS ANALYSIS

* SPECIFIC (STRONG) METHODS
  - KNOWLEDGE INTENSIVE

  eg: Medical diagnosis
  Computer configuration

* POWER vs GENERALITY TRADEOFF

MAIN ISSUES

KNOWLEDGE REPRESENTATION

INFERENCE (uncertainty)

CONTROL (forward / backward)

KNOWLEDGE ACQUISITION

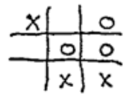
The INTERFACE
- questioning strategies
- explanation

Validation (final stages)

Borrowed from AI Intro Talk

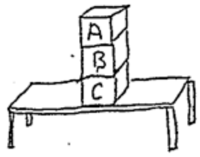
Knowledge Representation

- Stylised version of the real world
- Every 'piece' of representation must have unambiguous meaning.
- Examples:
  - Noughts and Crosses



201011022

  - Blocks World



on(a,b)
on(b,c)
ontable (c)
clear(a)
clear (table)

  - Fixing a bicycle

problem(dry-chain) :-
    pedaling (noisy),
    chain (rusty).

treatment (oil-chain) :-
    problem (dry-chain).

2EP-INF/8

KNOWLEDGE REPRESENTATION FORMALISMS

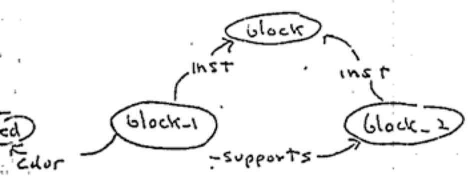
LOGIC

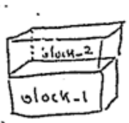
father-of (al, tom)
isa (mammal, animal)
$\forall x$ man(x) → mortal (x)

PRODUCTION RULES

IF <CONDITION> THEN <ACTION>

STRUCTURED OBJECTS





CHOOSING A REPRESENTATION LANGUAGE

* High Expressive Power

* Non Ambiguous

* Rules of Inference

Chapter 3

Production Rules

PRODUCTION RULES

Most Popular Scheme

GENERAL FORM:

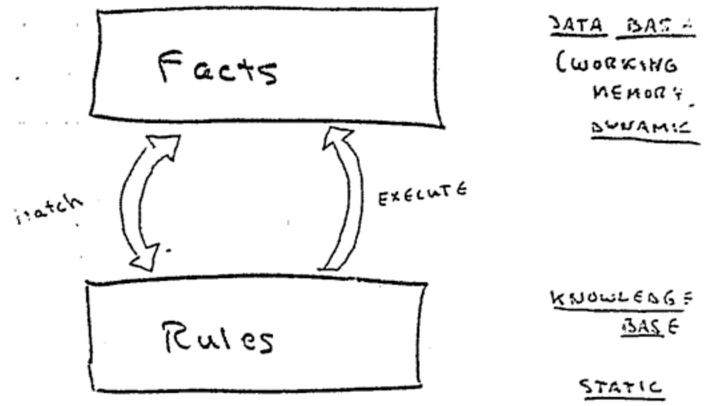
IF <CONDITION> THEN <ACTION> (CF)

where

- CONDITION: POSSIBLY COMPLEX
- ACTION: May affect real world (run a program, ask a question) / May add information to data base
- CF: Confidence Factor <optional>

EG: IF main component of a meal is fish THEN white wine should be served (.8)

USING PRODUCTION RULE



I. FIND SET OF APPLICABLE RULES

II. DECIDE WHICH ONE TO APPLY — no backtrack

III. APPLY IT — ADD / DELETE / MODIFY FACTS

GO TO I

CONTROL

forward chaining:

- Match left sides of rule w/ data base
- Apply rule
- Repeat until goal is in DB

backward chaining:

- Match goal to right hand side of rule
- Set up as subgoals the left side of the rule
- Stop when all subgoals are found in initial DB

SIMPLE EXAMPLE

R1: $m \rightarrow i$

R2: $b \rightarrow a$

R3: $i \rightarrow d$

R4: $j \rightarrow e$

R5: $k \rightarrow e$

R6: $g \,\&\, h \rightarrow c$

R7: $e \,\&\, f \rightarrow b$

R8: $c \,\&\, d \rightarrow a$

R9: $l \rightarrow i$

GOAL: Prove "a"

MAIN-COMP IS FISH [CR3]
THEN WINE IS WHITE (1.0)

IF MAIN-COMP IS UNKNOWN [QR1]
THEN ASK USER WHAT MAIN COMP IS

IF MAIN-COMP IS POULTRY [QR2]
THEN ASK USER IF MEAL HAS TURKEY IN IT

IF MAIN COMP IS POULTRY & [CR2]
NOT (MEAL HAS TURKEY)
THEN WINE IS WHITE (.9)
WINE IS RED (.3)

IF MEAL HAS TURKEY [CR5]
THEN WINE IS RED (.8)
WINE IS WHITE (.5)

SEED: MAKE MAIN-COMP UNKNOWN

QR1 IS MAIN-COMP FISH, MEAT, -- POULTRY
POULTRY

< two rules apply: CR2, QR2 must select

[CR2]: ADD TO WM WINE WHITE (.9)
WINE RED (.3)

QR2: DOES MEAL HAVE TURKEY?
YES

[CR5]: ADD TO WM WINE WHITE (.8)
WINE RED (.5)

NO MORE RULES FIRE

MAIN COMP IS POULTRY
MEAL HAS-TURKEY
WINE WHITE (.8)
WINE RED (.5)
WINE WHITE (.9)
WINE RED (.3)
} WM

SESSION II

QR1: IS MAIN-COMP FISH, MEAT or POULTRY?

POULTRY

QR2: DOES MEAL HAVE TURKEY

NO

CR2: ADD TO WM WINE WHITE (.9)
WINE RED (.3)

NO MORE RULES "FIRE"

MAIN-COMP IS POULTRY
MEAL HAS NO TURKEY
WINE WHITE (.9)
WINE RED (.3)
} WM

ES4-12

THINGS TO CONSIDER

* CONTROL STRATEGY: FC vs BC
  WHICH IS BETTER?
* CONFLICT RESOLUTION
* TYPES OF RULES
  - ASK QUESTIONS
  - ASSERT FINAL CONCLUSIONS
  - ASSERT INTERMEDIATE CONCLUSIONS
* SEED
* COMBINING CERTAINTY FACTORS
* IMPLEMENTATION DETAILS VARY

Chapter 4

Logic

AI2 - EXPERT SYST[illegible]S

LOGIC by Mike Uschold

********** DISCLAIMER **********

The intent of these notes is not to be a self contained and adequate description of logic. Such descriptions abound in textbooks. Rather, it is a summary of the lectures with commentary indicating the key issues and some explanation when necessary. I have included pointers to other sources for a more complete treatment of logic.

********** DISCLAIMER **********

We need some language for encoding our real world knowldege. What characteristics must it have?

- *High expressive power* - ie there must be a way to represent most things that you are likely to need.
- *Non Ambiguous* - No symbol or symbol structure can have more than one meaning.
- *Rules of Inference* - It must be possible to reason with the knowledge.

LOGIC is such a language.

1. What is Logic?

Logic is many things to many people Even within AI, there are a number of destinct roles that it plays. The best reference for explaining the various roles of logic in AI is an paper by bob Moore called "The Role of Logic in AI". It is highly reccommended reading. He discusses three major roles:

- As an analytical tool for representation languages
- As a knowledge representation and inference formalism
- Logic programming

We concentrate on logic as a representation language and inference system. We will largely ignore the first and third roles. Logic is a formal language for representing knowledge.

A fundamental notion in logic is that of *truth*. Statements are either true or false. Often, however, we are not concerned as much with the actual truth or falsity of an assertion or set of assertions, but rather we are concerned with how to make truth-preserving inferences. For instance, we might have a rule which says that if it is raining, then you will get wet. Here, we are not so much concerned whether it is raining or not, at a given time, but rather that if it *was*, then we can infer that you will get wet.

Logic has a rich set of such truth perserving inferences. The example above is called *modes ponens.*

Historical Overview

```
Syllogisms - Aristotle
    Pros:   Can quantify over individuals
    Cons:   Limited types of inference
            Limited Expressive Capability   (no connectives)

Propositional Logic - Boole
    Pros:   Can form complex expressions
            Many rules of inference
    Cons:   Limited expressive capability   (no quantifiers)

Predicate Logic - Frege
```



```
Pros:   High expressive capability
             * connectives
             * quantifiers
        Many rules of inference
Cons:   Can't always decide if a theorem is true
```

2. Propositional Logic

- *Definition of Proposition* - Any statement which is true or false.
- *Connectives* - These may be used to build complex formulae. They include:
  - & and
  - v or (not exclusive or)
  - ~ not
  - => implies (if ... then...)
  - <=> equivalence
- *Building complex formulae* - Any legal formula is known as a: *Well Formed Formula* (WFF). There are specific rules for building complex formulae using the connectives defined above. I summarise these here:
  1. A proposition is a formula
  2. If "p" and "q" are formulae, then the following are also formulae:
     - (~p)
     - (~q)
     - (p v q)
     - (p => q)
     - (p <=> q)
  3. Only expressions using rules 1. and 2. are formulae.

```
Eg: [p v ~(q => r)]  &  ~s        This is legal
     p => q &                     \
    [~(p => ) v r ] => s          / These are illegal
```

- *Encoding English as Propositional Logic* - Some Guidelines:
  1. Retain maximum expressive power.
  2. Look for smaller propositions within larger ones and represent these seperately.
  3. Look for keywords in English which suggest use of one of the connectives above.
     - & and
     - v or (Watch out for use of "or" in English as "exclusive or")
     - ~ Look for negatives (Doesn't, No, not)
     - => Look for: if... then...; whenever; unless
     - <=> Equivalence

```
Examples:
        Nigel is hungry                        p
        Tom is sexy                            q
        It's not warm and I'm shivering       ~r & s
        If Tom is sexy, I'll go out with him   q => t
```

```
Either Tom is not sexy, or
       I'll go out with him.            ~q v  t
```

```
NB:The last two are equivalent in meaning if you
   interpret "or" as: "one or the other or both".
   Convince yourself that this is so!
```

- *Evaluating the truth of formulae* - Using truth tables. First, we must define the meaning of each connective. Then the procedure is as follows:
  1. Pull apart the formula into its constituent parts which are joined by connectives.
  2. Evaluate each separate bit on its own. (ie, call this procedure recursively, starting at step 1 again)
  3. Treat each bit as a simple proposition and use the definition for the connective in question to get the overall truth value.

As an example, consider the WFF: [P & (P --> Q] --> Q).

Example:

P	Q	~P	P --> Q	~P v Q	[P & (P --> Q)] --> Q
t	t	f	t	t	t
t	f	f	f	f	t
f	t	t	t	t	t
f	f	t	t	t	t

- *Reasoning* - The type of inference that Logic allows is deduction. *Rules of Inference* enable one to derive new information from existing information. We refer to this new information as a *conclusion*, and refer to the old information as the *premises*. In applying logic, we do not randomly generate conclusions. Rather, one normally makes a conjecture and tries to *prove* it using the rules of inference. Once this conjecture is proven, it becomes a *theorem*, and may be added to the current set of premises. A theorem can also be used as a more new rule of inference to prove other more complex theorems. For example, recall high school geometry. You start with a set of basic axioms and gradually prove more and more theorms. The simpler theorems are used to prove more complex theorems.

A *theorem* is defined to be a formula which is always true. This is also known as a *tautology*.

Example Rules of Inference:

```
1.  & elimination    Given:  p & q
                             p
                             -----
                             Infer:  q

2.  modes ponens     Given:  p
                             p => q
                             ------
                             Infer:  q

Useful Equivalences:     a.  [~ p v q]    <=>    [p => q]
                         b.  ~[p & q]     <=>    ~p v ~q
                         c.  ~[p v q]     <=>    ~p & ~q
```

These may be used to draw inferences, or prove theorems. Convince yourself that these are true. Appeal to your intuition. If you cannot intuit well, you should attempt to prove these equivalences by using a truth table analysis.

3. Predicate Logic (also known as *First Order Logic*)

Propositional calculus is limited. You can't get "inside" a proposition and make use of the similarities of two different propositons. Eg. Consider the two propositions:
P: Socrates is a man. Q: Socrates is a philosopher.

There is no way to make use of the fact that both propositions pertain to Socrates. The language of predicate calculus provides a solution to this problem. Statements in this language are about objects or individuals and properties about them and relationships between and among them.

We would represent the above predicate P as "man(socrates)" and similarly, Q becomes "philosopher(socrates)". From these two predicates we can deduce that "Some men are philosophers". This would not have been possible in propositional logic.

At the most general level, in predicate logic, statements are about:

- objects (or individuals)
- properties of objects
- relationships between objects

More formally, Predicate calculus consists of the following:

- *Predicates*[1]: Statements which are either true or false. In this regard, they are similar to propositions in propositional logic in this respect. They may however, have one or more arguments.
- *Arguments*: These may be one of three types:
  - *Variables*: Empty slots which stand for either functions, or constants.
  - *Constants*: These are functions with no arguments.
  - *Functions*: "Return" objects related to their arguments. May have one or more arguments.
- *Connectives*: Same as for propositional logic.
- *Quantifiers*: "for all" and "there exists"
  (All X) p(X) is read: "For all X, P(X)"
  (Exists X) p(X) is read: "There exists an X such that p(X)"

Encoding English into Predicate Logic

```
Simple Examples:
All men are mortal
    (All X)  (man(X) => mortal(X))
Every child has a mother
    (All X) { child(X) => [(Exists Y) mother_of(X,Y)] }
No married person eats fish
    (All X) { [married(X) & person(X)] => ~eats(X,fish) }

Two equivalent versions conveying the same meaning:
    You can't be a husband without being married to some woman.
    (All X) ~{ husband(X) & ~(Exists Y) [married(X,Y) & woman(Y)]}
                          *or*
    Every husband is married to some woman.
```

[1] To conform with PROLOG terminology. The correct term for this is "atomic formula"

```
(All X) { husband(X) => (Exists Y) [married(X,Y) & woman(Y)]}
```

```
Things to consider:
    Choice of vocabulary depends on
        * Level of detail desired
        * Degree of flexibility required
    The predicates used will vary from domain to domain, but some
    predicates and rules keep showing up time and again, across
    many domains.   Examples of this include the equality
    predicate, and the law of transitivity.
```

Interpretation of a set of predicates and rules may vary. An example of this is the law of transitivity. It can be interpreted as many things, including descendent, taller_than, numerically less than etc. While this is the case, it is important to note that many possible interpretations are ruled out as well. For instance, if the predicate "foo" is transitive, then it MAY NOT be interpreted as "father_of", or "is_a_good_friend_of" etc. It is extremely important that some consistent interpretation of all the predicates and rules is possible, otherwise, it will be of no use to you.

Some Useful Equivalences with quantifiers:

```
~[(Exists X) p(X)]        <==>        (All X)   ~p(X)

~[(All X)    p(X)]        <==>        (Exists X) ~p(X)
```

The interpretation of these in English is fairly compelling. Let us use an example. Let the predicate "p(X)" refer to the complex predicate [person(X) & happy(X)]. To say that it is not the case that there exists a happy person (ie, there are no happy people), is really saying that all people are unhappy. Similarly if it is not the case that all people are happy, this is equivalent to saying that there is at least one unhappy person. These equivalences, and the others we saw in propositional logic can be used to show that the two logic representations used in the example above about husbands being married to women are in fact logically equivalent. I leave this as an exercise.

For a particular domain, represent *all* the relevant facts and relationships using predicate calculus notation. Consider the domain of mechanics. We will need all of the following:

- Usual axioms of arithmetic {<, >, +, -, * ...}
- Notation for units
  Eg: grams, acceleration, etc
- Object types
  Eg. instance_of(part1, pulley)
- Spatial Relationships
  Eg. contact(part2, end1)
  incline(part2, table, 30 degrees)
- Laws of Physics
  Eg. equals(Force, times(Mass, Acc))

Limitations of Predicate Logic

: The predicate logic we have been discussing is often known as first order logic. It allows you to quantify over objects or individuals. But there are still some statements which you cannot represent in first order logic. This occurs when you wish to quantify over predicates and/or functions. Examples include such statements as:

```
"Jim has some disgusting habits"
(Exists H) habit(H) & H(Jim)

"John loves everything about Mary"
```



```
(All P) personal_characteristic(P) & P(mary) --> loves(John, P)

A natural way to define of equality:
    (All X,Y)  { X=Y  <=> (All p) [p(X) <=> p(Y)]
```

Other problems include:

- Can't properly represent *possibility*. Eg. If the earth were much further from the sun, then life would
  never have evolved on it.
- Can't Properly Represent and Reason about Time and Events

Summary: Encoding into Predicate Calculus

Most difficult part is choosing vocabulary. Some important criteria for success are:

- Any fact in a domain must be representable.
- Simple facts should look simple
- Intuitively similar facts should look similar

Furthermore, it cannot be stressed too much that every symbol and every expression (ie, symbol structure) must unambiguously denote something in the real world.

4. Using Logic

Overview

- Structural Overview - What would a system which used logic look like? What are its major parts? How do they interact?
- Inference Rules - These are what we use to do reasoning. How can we derive new information from existing information in a principled way?
- Control Strategies
  - Forward Reasoning
  - Backward Reasoning
- Unification - This is a process which we use to find out which inferences we are allowed to make. Rules of inference may only be used if some way can be found to satisfy the premises. PROLOG uses unification.
- Automated Deduction - How might we automate a system so that it performs the correct inferences with minimal external guidance.
- How does Logic stack up? - We have explored one representation formalism. There are others which we will see later. What are the pros and cons.

A system for using logic will consist of a *database* which contains a set of predicate calculus assertions currently "believed", and some rules of inference. With this, we can derive new information. For example, consider the following:

Given:	Id	Justification
(All X) [male(X) => living(X)]	1.	given
male(harry_2)	2.	given

Derivation:

```
    male(harry_2) => living(harry_2)        3.      (1: universal
                                                        instantiation)

    living(harry_2)                         4.      (2 & 3: modes
                                                            ponens)
```

I have indicated the reasons for each step in the derivation on the right, the "Id" column indicates the unique identity of each "piece" of information, or assertion. An assertion may be given or derived.

SYSTEM STRUCTURE - *Idealised*

```
         -- new -->                  -- new -->
            facts                       facts
PROGRAM               INFERENCE                   DATABASE
         -- queries ->  ENGINE      <-- access --

        <-- answers --              <-- retract -
```

Inference Engine: Contains: *Rules of Inference*

- Assert Facts
- Retract Facts
- Query

There is a question which must be addresed regarding when the inferencing is to be done. We have essentially two choices which correspond to two different control strategies. These are:

- Forward Reasoning: Generate information whenever possible
  - *Fast at query time* - Since all the inferences are already performed at query time, there is a simple look up procedure.
  - *Slow at assertion time* - Which inferences to make?
    Assert: (All X) p(X)
    Infer: p(a), p(fred), p(duck), ...
  - This is a real problem! We could go on forever making inferences of this kind, most of which would be useless.
- Backward Reasoning: Generate new information when queried
  - *Slow at query time* - This is because the inferences are performed when the query is made. A query will be of the form: Is "p" true, where "p" is some arbitrarily complex predicate calculus assertion. In order for the system to answer this question it must either find the assertion in the database or prove that it "follows logically" from the existing set of assertions (ie, to deduce it).
  - *Fast at assertion time* - Since no inferences are performed until they are asked for, asserting new facts into the data base is very fast.
  - *Less wasted effort overall*

Forward Chaining

Recall the derivation we saw in the beginning of this section. It was an instance of the general form of argument which we see here.

```
Given:      (All X) [p(X) => q(X)]

                     p(a)

Show:               q(a)
```

We need to use the inference rule, "Universal Instantiation" to conclude that: *p(a) => q(a)* before we can use modes ponens to conclude *q(a)*. We have a serious problem here, namely: how can you know to derive: *p(a* => q(a)), but not: *p(tom* => q(tom)), *p(atom* => q(atom)), *p(jdhf* => q(jdhf)), ... It turns out that these quantifiers which considerably enhance the expressive power of our formalism don't come for free. It is extremely difficult to reason with them. We will now consider an alternate notation which will retain the expressiveness of the quantifiers, but which simplifies reasoning considerably.

Removing Quantifiers

Here, I describe very briefly this new notation with no quantifiers. Even though the quantifiers are not explicitly there, the expressive capability is retained. For this reason we refer to this notation as *implicit quantification.* The full blown procedure for removing quantifiers from arbitrarily complex formula is beyond the scope of these notes. It is called Skolomisation after the person who discovered it. I consider a few simple examples, leaving the general case to the interested reader to explore on their own. See Charniak and McDermott pp 344-351.

Removing Universal Quantifiers - Remove the quantifier and replace the quantified variables with a *?* preceding them. [Eg. "(All X) p(X)" becomes "p(?X)"] This denotes that the variable may match with anything, it does not matter since the relation holds for *all* X. This is exactly how PROLOG does it, except it drops the "?" and uses capitalised atoms. We will see how this works for us below.

Removing Existential Quantifiers - Remove the quantifier and replace the quantified variables with a *unique* constant. [Eg. "(Exists X) toy(X)" becomes "toy(sk_18)" The intuition here is that we only know that the relation holds for at least one constant. So, let us just pick one and give it a name. It is *absolutely crucial* that the new constant is unique. There must be no other constants in the database which match with it. If there is, the we could be in trouble because we won't have captured the same meaning as the original notation with the explicit existential quantifier. For example, suppose we have the assertion that there exists at least one happy person. According to our rule, we will give her a name, say *jill.* Suppose we had other information in the data base about *jill*, for instance, that she was a Nazi. We could then conclude that there is at least one happy Nazi (in particular, *jill.* This may, of course, be totally false! We should never have said the happy person was *jill*, because we don't know that. All we know is that there is at least one happy person, but we know *nothing* about the person. Not only isn't it *jill*, but it can't be anyone else that we know anything about. This is why the new name must be *unique.*

Examples:

```
Old:        (All X) [ cold(X)   => uncomfortable(X) ]
New:                  cold(?X)  => uncomfortable(?X)

Old:     (Exists X) happy(X) & person(X)
New:                happy(sk_18) & person(sk_18)
```

You may find the names we gave to the constants a bit strange, and indeed you may be right.

Comparison with PROLOG

PROLOG, although quite similar, is not the same as predicate logic. We developed predicate logic using propositional logic as a basis. As such, propositional logic can be thought of as a subset of predicate logic. It is less expressive. There are things that one simply cannot using propositional logic. Similarly, PROLOG is less expressive than predicate logic. In particular, it only allows a special type of clause. PROLOG, is nevertheless much closer to predicate logic than propositional logic and is thus considerably more powerful than the latter.

One of the most important differences is the inability to represent true negation. PROLOG doesn't know the difference between something being unknown and being known to be false. Take the example: Joe is Tom's brother. You might represent this in PROLOG as "brother_of(tom, joe)" If there were no such facts in the data base and no rules which allowed you to conclude that tom is joe's brother (by some combination of shared sisters, or cousins or whatever) then PROLOG would answer "no" to the query. Or, equivalently, it would answer "yes" to the query "not(brother_of(tom,joe)". It would be inaccurate for us to interpret this as definitly knowing that tom is "not" joe's brother, for the database may not contain any information about joe or tom. Similarly, a set of predicates about physics, may not have any information about philosophy in it. This way of dealing with negation is called "negation as failure". That is to say, if PROLOG fails to prove a goal, then it is taken to be false. This is popularly known as the *closed world assumption*. PROLOG's world is its data base. Another way to put it is to say if PROLOG doesn't know about it, it must not be true. Strictly speaking, then the "not" predicate doesn't necessarily mean "not" at all. Rather, it means "not provable". It is important to realise this when programming in PROLOG.

Another problem with PROLOG the inability to assert something of the form: p v q without having to explicitly assert one or the other or {p,q}.

Finally, PROLOG has no way to properly represent identity. For instance, suppose you have the following in PROLOG:
happy(tom).
happy(thomas).

You may wish at some later point to note that tom and thomas are in fact the same person. In order for this to be handled properly, a variabe (say X) should be able to unify with tom and thomas. This will never happen in raw Prolog. If you want this effect, you have to program it yourself.

4.1. SUMMARY - Using Logic

- Represent all facts & Relationships as Predicate Calculus formuale
- Attach an inference engine to generate deductions. Two possible control strategies are:
    1. Forward Chaining - Make inferences whenever possible as information is added to the database.
    2. Backward Chaining - Make inferences when you are required to answer a query.

One is only guaranteed correct deductions when premises are correct.
If the premises are in some way contradictory, then any reasoning which is carried out will be highly suspect, and therefore of no use. Taking a global view of things, what we mean by "premises" is really the whole set of predicate calculus assertions and rules. It may not in general be easy to detect contradictions, especially if the set is large. One thing which is essential, and a good way to avoiding contradictions is for there to exist some *consistent interpretation* of the set of assertions and rules. That is to say, you must be able to assign some consistent meaning to all of the predicates and together they must make sense. This, I have been stressing all along. Every

item in any representation scheme, be it logic or anything else, must unambiguously *mean* something!

EVALUATION of LOGIC

PROS:

- Very expressive formalism
- Inferences are guaranteed correct
- Simplicity of viewpoint. (Ie no worry about implementation)

CONS:

- Difficult to Encode
- Knowledge is opaque, unstructured
- Inference is Limited
    - Deduction only - Other useful types of reasoning we have seen are abduction and induction.
    - No exceptions to rules - In real life, there are always exceptions. We should like to be able to cope with this.
    - No uncertainty - Everything is either true or false. Again, this is unrealistic.

References

[Charniak & McDermott 85] Ch 1, pp 14-21
Ch 6 pp 319-353
[Bundy 84], "The Computer Modelling of Mathematical Reasoning"
Ch 1 pp 1-37
[Rich-83], "Artificial Intelligence"
Ch 5 pp 135-148

They are special constants which must have the uniqueness property described above. Since this process is called skolomisation, the constants are skolom constants (hence the names you see). We say that we have *skolomised* the formula. Let us come back to forward chaining. We noted that in order to use the modes ponens rule of inference, we had to use universal instantiation first. But this led to the problem of knowing which among the possibly infinite number of inferences one could should make. Straight forward chaining, which says make any and all inferences whenever you can is doomed to failure. With this new notation we adopt a slightly different viewpoint, which allows us to conveniently handle this problem. We have the following situation:

```
Given:         p(a)
          p(?X) => q(?X)          Alias: (All X) [ p(X)  => q(X) ]
          --------------

Derive:        q(a)
```

With this new implicit quantifier notation, we do things slightly differently. In particular, we no longer generate willy nilly instances of universally quantified expressions. For instance, in this example, we don't start generating *p(a* => q(a)), *p(tom* => q(tom)), ... Instead of making all the inferences possible in this way (and of course getting nowhere in the process), we wait until a specific instance of the predicate "p" comes along which matches with the "p(?X)". In this case, we have "p(a)", so we can safely conclude that "p(a) => q(a)" and thus conclude "q(a)". The key here is the matching process. It is required to determine when an inference will (or can be) made. It is more formally known as *unification*. A set of expressions are said to *unify* if some set of substitutions of constants for variables can be made which will make all the expressions equal. In this case, the substution { ?X=a } makes the two expressions: "p(a)" and "p(?X)" equal. The set of substitutions is called a *unifier*. In general there may be more than one unifier for a set of expressions. For example, consider the two expressions: "q(?X)" and "q(?Y)". The are infinitely many unifiers for these expressions. Two of them are { ?X=a, ?Y=a } and { ?X=tom, ?Y=tom }. In general, there will not be so many. The normal convention is to use the greek variable *theta* to represent unifiers. Here, I use the symbol; "$". We use the notation "p$" to denote the expression which results from applying the substitution "$" to the formula "p". Using this terminology, let us return to forward chaining and present the general case:

```
Given: p1
       p2 => q1
       --------------

Derive: q
```

If and only if there exists some substitution of variables "$" such that:

```
p = p1$ = p2$
q = q1$
```

p1 and p2 are said to *unify* if such a substitution can be found. This substitution ("$") is called a unifier.

Consider the following example:

```
                  _____p1_____
                 /            \
Assertion:       on(block_1, ?Y)

Rule:            on(?X,table)   ==>  above(?X,table)
                 \____________/
                       p2
```



```
$ = { ?X=block_1, ?Y=table }

p1$ = p2$ = on(block_1, table)
```

Let us now consider Backward Chaining. We noted above, that backward reasoning is a strategy which waits until a query is made, and then attempts to answer the query. A query takes the form of a predicate calculus formula which must either be found in the database or deduced from the assertions already in the database. We shall consider the same example. When doing forward chaining, we match the left hand side of the implication (the *if* part) with existing facts in the database to see if they match. If so, we deduce the *then* part and add it to the database as a new assertion. When reasoning backward, we wait for something to deduce, and only then will we attempt to make any inferences. If we want to deduce that block_1 is above the table, we try to find a rule which will allow us to conclude that. In particular, what rule has "above(block_1,table)" in its *then* part? If we can find such a rule, then we try to use it. We can only use it if its *if* part is true. So, we set up the *if* part of the rule we hope to use as a subgoal. This subgoal may be an assertion in the database. If so, great. Otherwise, we will have to apply the same procedure in attempting to prove the subgoal. We will have to look for a rule which concludes this new subgoal, in the same way we did for the original goal. Note that unification is still of paramount importance. In the example below, we have no rule which has "above(block_1,table)" in its *then* part. But, we *do* have a rule whose *then* part *unifies* with it.

Example:

```
Show goal:      above(block_1, table)

   Given:       on(?X,?Y)   ==>   above(?X,?Y)

 Subgoal:       on(block_1, table)


 Unifier:       $ = { ?X=block_1, ?Y=table }
```

I now present the general case for Backward Chaining.

```
Show goal:      q1

   Given:       p1 ==>  q2

 Subgoal:       p


Find substitution "$" such that:  q = q1$ = q2$
                                  p = p1$
```

Automated Deduction

It is now appropriate to say a few words about how we might automate the deductive process. Again, it is beyond the scope of these notes to go into any detail. The interested reader is referred to Bundy (see above) for a most thorough and well presented exposition. For our purposes, I simply outline the overall scenario. First, all quantifiers must be gotten rid of. It's just too difficult to reason with them. Secondly, we must have some formal way of doing inference. We saw many examples of inference rules. If you're trying to prove some formula, there may be many ways to go about it. You will have to choose among many options. This is seen to be a great disadvantage. It turns out that only one rule of inference is actually needed. This rule is called *resolution* and is a generalisation of forward and backward chaining. We make deductions by repeated application of this rule. With this great insight, the problem of searching for which rule of inference to apply goes away, but there are still other problems. In particular, it is in practice very difficult to find unifiers efficiently, especially for realistic sized problems. There are other serious search problems as well, which are outwith the scope of these notes, and in fact constitute a whole sub-field of research.

Notes by Graeme Ritchie. Not intended specifically for AI2. -2-
But a very good presentation, and [redacted] very appropriate.

Predicate Logic

Predicate logic is a symbolic language (a calculus) which can be used to describe and reason about items and relationships between these items. There are rules for forming symbolic expressions, and for manipulating the expressions in well-defined ways. The manipulation rules are designed so that the symbolic expressions that they create have meanings which are systematically related to the meanings of the original expressions.

(Notice that the definitions given here are slightly non-standard, since most mathematical descriptions of logic use a very small set of symbols, rules, etc., to define predicate logic. Here, we will introduce symbols and rules in a less economical way, in order to get the same effect but in a clearer, more intuitive way.)

1. The symbols

The basic vocabulary used in predicate calculus contains:

Brackets:) and (
Names: these include variable-names, constant-names, function-names, and predicate-names. Each function-name and each predicate-name has an "arity" (a positive integer indicating how many arguments it takes).
quantifiers: ∀ and ∃
Implication: =>
Negation: ¬
Boolean connectives: ∨ and ∧
Equivalence: ≡

There are syntax rules which define the valid ways of putting these together. Symbolic expressions fall into two broad categories:

(i) Terms

These consist of -

variable-names
constant-names
any function-name of arity N, combined with N terms
(e.g. f(a,b,c), or g(f(x1,x2,x3), b, h(c,d))

(ii) Formulae

The simplest form (known as an atomic formula) is a predicate-name of arity N combined with N terms, e.g..

P(a, f(a,b,x1))
Q(g(x3,x4,x5), c, h(a,b))

The other forms of formula can be built using formulae and the other symbols, as follows. Assume S and T are formulae of some kind. Then:

S => T

¬ S

S ∧ T

S ∨ T

S ≡ T

are all well-formed formulae. The connectives used to form these compound formulae are usually read as "implies", "not", "and", "or" and "equivalent to", respectively. There are other more complex forms of formula, involving quantifiers, which will be described in later sections below.

Notice that these rules simply define how the symbols may be stuck together to make formulae - they say nothing about what the formulae mean, or about how to manipulate a given set of formulae.

2. The meaning of the formulae

Predicate calculus can be used to describe any "world" which has the following form:

There is a set of objects (the "universe" or "domain").

There are a set of properties that each object may possess.

There are functions; each function will (given some objects) single out some specific object.

There are relations which can hold between objects.

Example

If we were dealing with elementary arithmetic, then we might use:

Objects: 0, 1, 2, 3,.....
Properties: Odd, Even, Prime, etc.
Functions: Addition, Subtraction, etc.
Relations : Equal, Greater-than, Less-than, etc.

Let us look now at how the expressions (terms and formulae) are associated with their meanings in terms of objects, relations, etc. This is done with "interpretation rules" (also known, in the case of formulae, as "truth conditions"). These rules specify what object in the universe each term refers to, and exactly what circumstances (in the universe) would make a formula be classed as "true". A universe like this, together with the associations between symbolic expressions and the universe, is referred to as a "interpretation" for the set of formulae involved.

LOGIC L7

It is hard to explain exactly how these rules are defined without becoming rather mathematical, but the general idea is as follows:

Each 1-place predicate-name (i.e. with arity of 1) is associated with a property.

Each predicate-name of arity N (greater than 1) is associated with a relation which has the same arity (i.e. takes the same number of arguments).

Each N-ary function-name is associated with a function taking N arguments (for any N).

Each constant-name is associated with a particular object (in the universe).

Then the truth-conditions (i.e. interpretations of formulae) are much as might be expected:

Atomic formulae

Let P be an n-place predicate, and let X1,....Xn be terms (i.e. constants, or function-argument expressions). Then

P(X1,X2,....,Xn)

is true if and only if the relation associated with P in the universe holds between the objects associated with X1,...Xn.

For more complex formulae, the truth-conditions are defined in terms of the various parts of the formula, as follows. Let S and T be any two formulae (either atomic formulae, or more complex formulae made up with connectives).

Conjunction

The formula

(S ^ T)

is true in the interpretation if and only if S is true in the interpretation and T is true in the interpretation.

Disjunction

The formula

S v T

is true in the interpretation if and only if either S is true in the interpretation or T is true in the interpretation (or both).

Negation

The formula

¬ S

is true in the interpretation if and only if S is false in the interpretation.

Implication

The connective "=>" can be confusing. One way to think of the meaning of "S => T" is to regard it as a shorthand for the formula "¬ S v T". (As noted above, we could get by with fewer symbols, and define the other symbols in terms of just (for example) the two connectives "¬ " and "^"). The formula

S => T

is true if it is never the case (within the interpretation) that S is true but T is not true. This is only an approximation to the informal, everyday meaning of "S implies T", so care is needed in using it. (Check for yourself that treating S => T as ¬ S v T will give the same meaning).

Equivalence

The formula "S = T" can be regarded as a shorthand for

(S => T) ^ (T => S)

In terms of interpretations, S = T is true if it is impossible for either S or T to be true without the other also being true (in the interpretation).

For any given set of formulae, an interpretation in which they are all "true" is called a model for those formulae.

3. Quantification

One extremely useful aspect of predicate calculus is its way of making general statements, using the quantifier symbols. The "universal quantifier", ∀ , can be used in formulae of the form

(∀ x)S(x)

where x is any variable, and "S(x)" is any formula (not necessarily atomic) which contains the variable x and in which there are no symbols (∀ x) or (∃x) already. The variable x is said to be "bound" by the quantifier, and the formula is read as "for all x, S(x)". (An unbound variable - i.e. one which does not have an associated quantifier along with it - is said to be "free").

The interpretation (truth-condition) for a universally quantified formula is that if we try replacing the original x (in S(x)) with the name of any object in the universe, the resulting version will be true. That is, "(∀x)S(x)" is a kind of general assertion that S is true for any object in the universe, with x marking the part of S that refers to the object.

The other quantifier, $\exists$, is known as the "existential quantifier", and it can also be used in formulae containing a variable which is not already bound by another quantifier. That is,

$(\exists x)S(x)$

is a well-formed formula, providing that $S(x)$ is a well-formed formula containing x and not including any symbols "$(\forall x)$" or "$(\exists x)$". This statement is read as "there exists an x such that $S(x)$", and it is again a kind of generalisation about objects in the universe. The interpretation of this formula is that some object can be found in the universe such that S is true when that object's name is inserted in place of x.

Notice that there can be well-formed formulae which have no truth-value assigned to them by an interpretation, since they do not make assertions about the universe. A formula with no free variables is called a sentence or a closed formula and is assigned a truth-value under an interpretation; a formula with free variables is called a non-sentence or an open formula, and cannot be assigned true or false.

4. Rules of inference

So far we have shown:

(a) how well-formed formulae (terms and formulae) are built using formation rules.

(b) how a well-formed term can be associated with an object in the universe

(c) how a well-formed formula (strictly, a closed formula) is given a truth-value using truth-conditions.

This does not let us do anything very exciting, except devise universes and write down formulae describing them. The importance of predicate logic comes when we add inference rules. These are rules which state how, given a set of true formulae, further true formulae can be produced, using only symbolic manipulations of the formulae (i.e. without consulting any model). This is useful, since it means that a person or machine can be given a small set of logical formulae and can derive other formulae from them, with a guarantee that the newly computed ones will be as true as the original ones. Hence a partial description of a model (using a few statements) may be filled out into a more detailed description, or "deductions" can be made which were not in the original data.

Some of the inference rules are intuitively clear. For example, one is based on the idea that if a formula is true for all objects, it is true for some particular object:

From -
$(\forall x)S(x)$
Deduce -
$S(t)$ (for any term "t").

Another useful rule captures the idea that if S implies T, and S is true, then T must be true also:

From -
S => T
S
Deduce -
T

Each individual rule may seem ridiculously simple, but they have to be kept fairly elementary if we are to be sure that applying a rule will always produce formulae that are as true as the initial ones. (If the initial formulae are not known to be true, but are merely assumptions, then the deductions are dependent on the assumptions, and may not be true if the assumed formulae are not. This can be regarded as a form of hypothetical or conditional reasoning). However, a number of inference rules, carefully applied, can produce very complex deductions. Here, in brief, are the inference rules in our sketch of logic.

1. From P => Q and P, deduce Q.

2. From $(\forall x)P(x)$, deduce $P(c)$ for any term c, (providing c does not contain any free variables which are already bound in $P(x)$).

3. From $P(c)$ (where c is any term), deduce $(\exists x)P(x)$ (providing $P(x)$ does not already contain a bound occurrence of x).

4. From $P(x)$ (where x is any free variable), deduce $(\forall x)P(x)$.

5. From P, deduce P v Q

6. From P ^ Q, deduce P.

7. From P and Q, deduce P ^ Q.

8. From P v Q and ¬ P, deduce Q.

9. From P => Q and ¬ Q, deduce ¬ P.

10. From P => Q and Q => R, deduce P => R.

11. From (P => Q) ^ (R => S), and P v R, deduce Q v S.

12. From (P => Q) ^ (R => S), and ¬ Q v ¬ S, deduce ¬ P v ¬ R.

5. Symbolic inference

Although the above Sections suggest that inference rules depend on the idea of "truth in a model", this has been a misleading simplification. The rules have been designed to preserve truth (i.e. generating formulae that are as true as those given initially), but they are used without any use of truth or interpretation. That is, the rules operate on the symbolic structure of the formulae, without using truth-conditions, so that it is possible to treat deduction as a wholly mechanical symbolic behaviour. The inferencer (person or computer) starts from some set of formulae (the "axioms") and, by applying inference rules, produces further formulae (more "theorems"). The notation

$\vdash$

is short for "this is an axiom or theorem:".

Various mathematical results can then be proved about the behaviour of logic systems, usually without any need to refer to the notion of "truth" or "interpretation" at all; the only relevant notions are the derivation of theorems from axioms using inference rules. Although the idea of "truth within a model" provides the motive for all this symbol-pushing, it can be laid aside much of the time when implementing computer systems for inference.

Example

Axioms:

$\vdash$ (∀x) ((P(x) ^ R(x,a)) => Q(x))
$\vdash$ P(b)
$\vdash$ R(b,a)

(a,b constants; x a variable)

We can deduce that

$\vdash$ Q(b)

in the following way (rule numbers are from Section 5 above):

$\vdash$ ((P(b) ^ R(b,a)) => Q(b)) (by rule 2, axiom 1)
$\vdash$ (P(b) ^ R(b,a)) (by rule 7, axioms 2 and 3)
$\vdash$ Q(b) (by rule 1, previous two theorems)

This is independent of any particular interpretation. (You may try imposing various interpretations on the predicate-names and constants, and examining the result. For example, using the set of positive integers as the model, we could have P = Prime, Q = Odd, R = Greater-than, a = 2, and b can be the name of any prime). The typical task which is considered in many systems is of the form:
Try to find a derivation (sequence of rules) which will produce theorem T starting from axioms S1, S2,...,Sn.

(WARNING: The question of whether or not a particular formula can be derived from a given set of axioms is, in general, not computable.)

LOGIC/10

7. Validity, Consistency, Unsatisfiability

A formula is said to be valid or tautologous (or a tautology) if it is true under all its interpretations. That is, if the symbolic structure of a formula is such that it could not possibly be false (for any allowed interpretation of its symbols), then it is tautologous (valid). For example,

P(a) v ¬ P(a)

A(c,d) => A(c,d)

are both valid.

Similarly, a formula is inconsistent or unsatisfiable if it is false under any possible interpretation. For example,

Q(c) ^ ¬ Q(c)

is unsatisfiable.

Notice that it may happen that a given closed formula (sentence) is neither valid (always true) nor inconsistent (always false). It may be true under some interpretations, and false under others. For example,

P(a) => Q(b)

might be true for certain meanings of P,Q,a and b, but not for others.

A formula is said to be satisfiable or consistent if there is some interpretation which makes it true (i.e. it is not unsatisfiable/inconsistent). (Remember that a formula which contains unquantified (free) variables (i.e. an open formula) is neither true nor false regardless of the interpretation).

8. Soundness and Completeness

In designing an inference system, it is essential to consider whether the proposed inference rules perform in a desirable way. The first question to consider about an inference rule is - does it generate unwarranted deductions?

An inference rule is said to be sound if, given an initial set of axioms (formulae) which are satisfiable, the deduced formula is satisfiable by any model which satisfies these axioms. That is, it does not generate formulae which might be true when the axioms are question are false.

The other obvious question about a system of inference is - will it overlook any deductions? A system of rules is said to be complete if, given a an initial set of axioms, it allows the deduction of every formula which would be true if those axioms were true.

All the inference rules given above are sound, and the system as a whole is complete; we will not attempt to prove these results here. As commented earlier, the version used here is somewhat verbose - it is possible to have a complete version of predicate logic with only one inference rule. In fact, the best known system of machine inference for predicate logic (resolution) is just such a system.

9. Logical Axioms

A logical description of some "world" or subject-matter has its information in two forms - axioms (formulae that are taken to be true) and inference rules (deductions that are sound). There is not one fixed way of arranging such a logical description - it is possible to reduce the number of inference rules by increasing the number of axioms, and vice-versa. Inference rules capture certain generalisations about what deductions can safely be made from what kinds of facts. These same generalisations can (alternatively) be represented within a logical system by inserting more axioms. The usual arrangement (not the one followed here) is to have only one or two inference rules, then have "logical axioms" which contain the extra necessary information. For example, suppose a system has the rule of "Modus Ponens" (our Rule 1 above):

```
P
P => Q
______
Q
```

Then any rule of the general form "from X deduce Y" can be replaced by a "logical axiom" of the form

X => Y

(providing both X and Y are "closed" - i.e. have no unbound variables). Consider the effect. A rule "from X deduce Y" causes "Y" to be a theorem if X is found to be a theorem. This is exactly the effect that the re-formulation will have - if X is a theorem, then this axiom "X => Y" will give the following Modus Ponens deduction:

```
X
X => Y
______
Y
```

Hence the two arrangements are equivalent.

The traditional set-up is:

(a) very few inference rules

(b) several "logical axioms" which use the "=>" connective to build-in the required inferences, as above, and which are nothing to do with the subject matter (i.e. they are purely logical generalisations).

(c) when it is required to actually describe some subject matter (i.e. apply this system to some problem) more axioms ("non-logical axioms") are added to capture the specific facts about this particular universe.

A system containing just (a) and (b) (or just (a), if using the approach employed in these notes), is called a "predicate calculus".

Notice that inference rules (and logical axioms) are really patterns which could apply to a wide range of formulae (i.e. they are "schemas" which indicate the form of a valid inference or of a logical axiom). These rules can be applied to formulae other than those called "P" or "Q".

10. Concluding Remarks

It should be emphasised that this is a very sketchy, informal, unrigorous introduction to predicate logic. It is not mathematically precise, and is intended only to convey the general ideas, so that the reader can start on a proper textbook with some understanding of the overall picture.

Chapter 5

MYCIN

OUTLINE ①

MYCIN A DETAILED EXAMPLE

- MEDICAL COMPUTING
- WHY STUDY MYCIN?
- DIAGNOSIS & TREATMENT of BLOOD INFECTIONS
- COMPONENTS OF MYCIN
  - * Consultation
    - Rule Base
    - Data Structures
    - Control Structure
  - * Explanation
  - * Rule Acquisition
- EVALUATION

WHY MYCIN? ②

- TYPICAL of Broad Class of ES
  - * diagnosis
  - * advice
- Realistic Complexity.
  - * Fulfils a need (non-toy)
  - * high performance
  - * reliability
  - * useability
- INSPIRED Other Systems
  - * new domains
  - * extra Facilities
    - explanation
    - Knowledge acquisition

MYCIN INTRO

ical Expert Systems

ADVISORY: Diagnosis

Treatment

USE COMPUTERS?

- cost
- too much information.
- geographical distribution of expertise & resources
- scarcity of physicians time

MYCIN/4

MYCIN DOMAIN:

BLOOD INFECTIONS

ANTIBIOTIC: Drug designed to kill bacteria or arrest growth

COMPLICATIONS:
- No single drug works for all bacteria
- Some are toxic

THE TASK: Recommend Therapy

1) Does patient have infection?
2) Identify Responsible Organisms
3) Find Appropriate Drugs
4) Select best drug(s)

MORE PROBLEMS

Bacteria are normal in body
Contamination may occur in lab

DIAGNOSIS:

- INITIALLY BASED ON CLINICAL CRITERIA (fever 2 pam)
- 24-48 hrs to IDENTIFY Organism
- Drug Sensitivities Vary

HELP IS NEEDED

- Misuse of Antibiotics, Cover prescribed (10-30
  - High Cost
  - Resistant Strains

INCOMPLETE Evidence UNCERTAIN ORGANISM Id
UNCERTAIN DRUG Effectiveness & Side Effects

⟹ NOT EASY !!

MYCIN

Purpose: To assist physicians who non-expert IN Antibiotics

- * DIAGNOSIS
- * TREATMENT

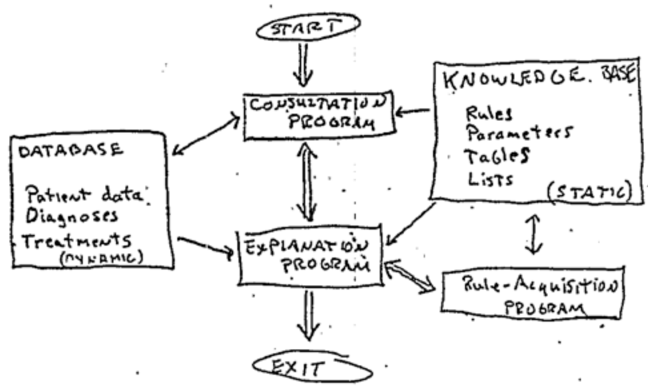
of BLOOD INFECTIONS

HISTORY: 1972, PHD THESIS 1976
SHORTLIFFE, VAN MELLE

MAIN COMPONENTS:

- * Consultation
- * Explanation
- * Acquisition

MYCIN OVERVIEW



⟹ Flow of Control
⟶ Flow of Information

Typical Consultations:

System Collects Information on Patient
Clinical Condition; Symptoms
Medical history; Lab Findings

Begin with basic questions.
Attempt diagnosis
Infer additional information or request it.
System Suggests list of Therapies

A SAMPLE SESSION

1) Patient's name: (first-last)
** FRED BRAUN
2) Sex:
** M
3) Age:
** 55

(Questions 1-3 request background patient data.)

4) Are there any cultures for Fred Braun which may be related to the present illness, and from which organisms have been grown successfully in the microbiology laboratory?
** Y

--------CULTURE-1--------
5) From what site was the specimen for CULTURE-1 taken?
** BLOD
=BLOOD

(System corrects typing error.)

6) Please give the date and time when CULTURE-1 was obtained. (mo/da/yr time)
** JUN 20, 1977
The first organism isolated from the blood culture of 20-JUN-77 (CULTURE-1) will be referred to as:

--------ORGANISM-1--------
7) Enter the laboratory-reported identity of ORGANISM-1:
**UNKNOWN

(Typically identity is not yet known.)

8) The stain (gram or Ziehl-Neelsen acid-fast) of ORGANISM-1:
** NEG

(Preliminary lab results give some clues.)

9) Is ORGANISM-1 rod or coccus (etc.):
** ROD
10) What is the form of the individual organisms (e.g. lancet-shaped for cocci, fusiform for rods, etc.)?
** FUSIFORM

(... more questions follow in order to gather sufficient information to infer the identity and significance of the infecting organisms ...)

23) What is the suspected portal of entry of the gramneg rod from the blood culture of 20-JUN-77 (ORGANISM-1) into this sterile site?
** GI (6)

(User indicates the uncertainty of his response.)

24) Is Fred Braun a burn patient?
** N
25) Is Fred Braun a compromised host (e.g. alcoholic, sickle-cell-disease, immunosuppressed)?
** N
26) Is Fred Braun febrile?
** Y

(... after about 50-60 questions, MYCIN prints the diagnostic hypotheses on which therapy will be based ...)

My therapy recommendations will be designed to treat for organisms that are either very likely or, although less likely, would have a significant effect on therapy selection if they were present. It is important to cover for the following probable infection(s) and associated organism(s):

INFECTION-1 Is BACTEREMIA

<ITEM-1> E.COLI [ORGANISM-1]

MYCIN RULES

VGLISH:

ΞF ORGANISM HAS STAIN: GRAMNEG
& ORGANISM MORPHOLOGY IS: ROD
& ORGANISM AEROBICITY IS: AEROBIC

HEN THERE IS STRONGLY SUGGESTIVE EVIDENCE (.8) THAT THE CLASS OF THE ORGANISM IS: FOO

SP:

F ($AND (SAME ?CNTXT STAIN GRAMNEG
(SAME ?CNTXT MORPH ROD)
(SAME ?CNTXT AIR AEROBIC
)

EN (CONCLUDE CNTXT CLASS FOO TALLY .

RULES: General Format

$<rule>$:= $<premise><action>$

$<premise>$:= ($AND $<cond><cond>\ldots$)

$<cond>$:= ($<fn><object><att><value>$) | ($OR <COND> ... <Cond>)

$<action>$:= $<conclusion>$ | $<actionfunction>$

function: truth valued predicate with CF
e.g. SAME, KNOWN, DEFINITE

object: Domain entities; ("*context*")
e.g. PERSON, DRUG, ORGANISM

attribute: property of an object

e.g.
- PERSON: {name, age, sex}
- ORGANISM: {stain, class, form}
- DRUG: {name, dose}

value: of the attrubute
e.g. sex=male
dose=3mg

ction function: instruction to be carried out
eg. compile list of therapies

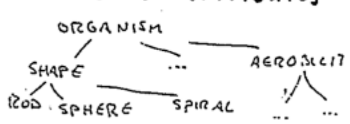
5

MYCIN/12

DATA STRUCTURES

ATIC: Definitional Knowledge

- lists (of possible organisms)
- tables (of parameter values)
- trees (classifying clinical parameters i.e.: objects' attributes
- rules



MUCH MEDICAL KNOWLEDGE IS NOT IN RULE FORM!

PARAMETERS

PARAMETERS are assigned PROPERTIES
- guide application of rules
- monitor user interaction
  YES-NO (fever) / SINGLE-VAL (name) / MULTIPLE-VAL (infection)

* RULE GUIDANCE (Efficiency)

LOOKAHEAD: Rules containing parameter in PREMIS
UPDATED-BY: Rules containing parameter in CONCLUSIC

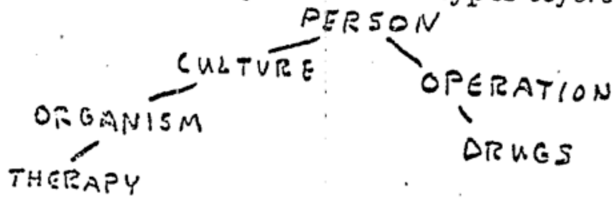
* USER INTERACTION

LABDATA: clinical test result (certain info)
EXPECT: Possible Values (male / Female)
PROMPT: Text seen by user
TRANS: Translate into English

DATA STRUCTURES: Dynamic

PATIENT CONTEXT TREE: A record of information about various entities in domain gained during a session.

- Static heirarchy of context types *before* session



- During session, contexts are instantiated as necessary.
  - PERSON: *exactly* one (the root node)
  - CULTURE: *at least* one
  - OPERATION: *optional*

PATIENT DATA BASE: Symptoms, Lab results, Conclusions, etc.

CONTROL

SUMMARY:

* REPRESENTATION
  RULES + FACTS & TABLES
* INFERENCE
  RULES FIRING
* CONTROL
  BACK CHAINING ?
  FORWARD CHAINING ?
  BOTH ?

MYCIN CONTROL:

MOSTLY BACK CHAINING

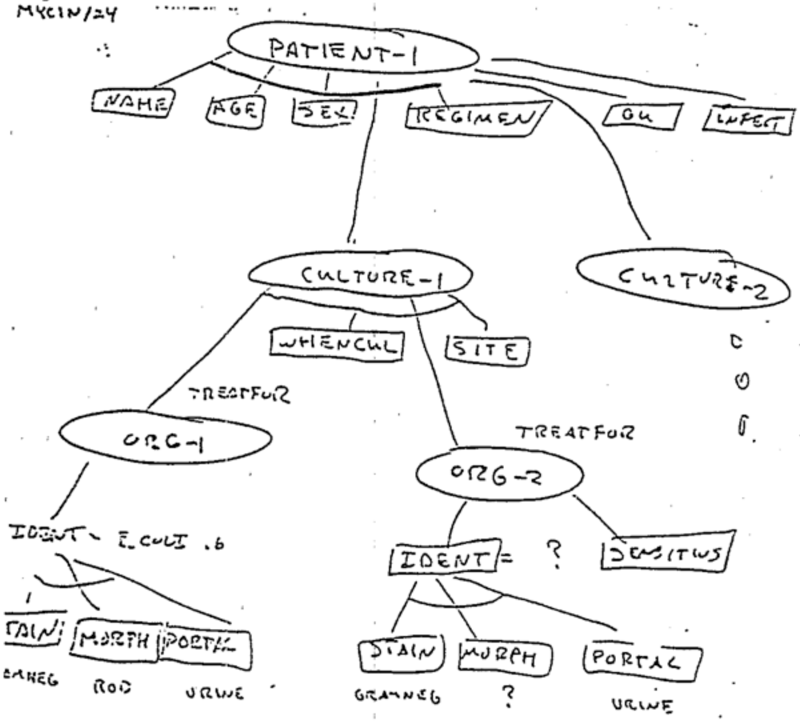
TOP LEVEL GOAL:

PRESCRIBE AN OPTIMAL THERAPY

FIND RULES WHICH HAVE THE (ACTION) ON THEIR RHS

...

MYCIN/24



AND-OR TREE

AND: Rule Premises
Main Props

OR: Rule Premises
Other Props

(build this dynamically on board)

GETTING STARTED

INSTANTIATE PERSON CONTEXT

- ASSIGN UNIQUE NAME (PAT-1)
- ADD TO CONTEXT TREE (ROOT NODE)
- "TRACE" MAIN PROPERTIES
  - NAME — LABDATA
  - AGE — LABDATA
  - SEX — LABDATA

  (ask right away)
  - THERAPY — infer

REE! (PAT-1)

FIND RULE WHICH HAS THERAPY ON RHS

GETTING STARTED

RULE Ø92 "GOAL" Rule

IF THERE IS AN ORGANISM REQUIRING THERAPY <TREATFOR>

THEN COMPILE A LIST OF POSSIBLE THERAPIES & PRESCRIBE THE OPTIMAL THERAPY

THIS RULE DRIVES ENTIRE CONSULTATION!

NEXT: FIND RULES WHICH HAVE <TREATFOR> IN THEIR RHS

USE "UPDATED-BY" PROPERTY OF <TREATFOR>

— BACK CHAINING —

9

N/17

EXAMPLE

RULE Ø9Ø ORG-RULE

IF IDENTITY OF AN ORGANISM IS KNOWN & THERE IS A SIGNIFICANT DISEASE ASSOCIATED WITH THIS ORGANISM

THEN IT IS DEFINITE (1.0) THAT THERE IS AN ORGANISM WHICH REQUIRES THERAPY <TREATFOR>

PROBLEM: WE ARE CURRENTLY IN "PERSON" CONTEXT. THUS ORG-RULES DO NOT APPLY!

CIN/18

EXAMPLE

PROCEDURE:

- LOOK IN C-TREE FOR SON OF TYPE "ORG" — FAIL
- TRY TO CREATE "ORG" CONTEXT
  - * TEST: IS "ORG" a SON of "PERSON" — NO
  - * "ORG" IS a SON of "CULTURE"
  - * TEST: IS "CULTURE" SON of "PERSON" — YES
  - * INSTANTIATE "CULTURE" CONTEXT
    - Select Prompt
      - Are there any ?X (MAYBE NONE)
      - Are there any more ?X (VARIABLE)
      - Give me a ?X (AT LEAST ONE)
    - Name CULTURE-1
    - Add to C-TREE
    - TRACE MAIN PROPERTIES

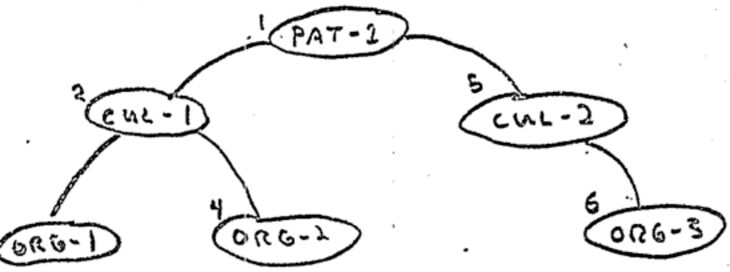
1N/14

EXAMPLE cont'd

* NOW INSTANTIATE "ORG" CONTEXT

* ASK FOR ANY MORE ORGANISMS
  - YES: ORGANISM-2, ORGANISM-3...
  - NO: MARK QUESTION UNASKABLE

* ASK FOR ANY MORE CULTURES
  - YES: CULTURE-2 ...
  - NO: MARK QUESTION UNASKABLE



DEPTH FIRST

"TRACING" PARAMETERS

AS EACH CONTEXT IS CREATED, MUST TRACE IT'S MAIN PROPS

CULTURE: SITE, WHENCUL — LABDATA ⇒ ASK RIGHT AWAY

ORGANISM: IDENT, STAIN, MORPH, ⋮

TRACING IDENT for ORGANISM-1

Set "isbeing traced" Flag

Fetch "prompt": Ask physician. <UNKNOWN>

Check "traced" flag <OFF>

Retrieve Relevant Rules "updated-by" (back-chain)

1N/21

TRACING PARAMETERS

RULE Ø156 ORG RULE

IF	SITE = BLOOD	<CUL>
	& GRAMSTAIN = NEG	<ORG>
	& MORPH = ROD	<ORG>
	& INFECT = CYSTITIS	<PERSON>
THEN	IDENT = E.COLI TALLY (.6)	

+ OTHER RULES
⋮

5) IS RULE APPLICABLE? YES
   CURRENT CONTEXT = ORG-1
   RULE TYPE = ORG

6) APPLY RULE - Evaluate each premise
   TRACE IF NECESSARY
   3 POSSIBILITIES:
   * istraced: (ask anyway if multiple valued)
   * ASK
   * infer: (more BC)

SITE: Culture PROPERTY ⇒ Move to CONTEXT: CULTURE-1 [CULTURE MAINPROP]
"istraced" flag set
TEST SITE = BLOOD |CF| ≥ .2
  YES ⇒ CONTINUE
  NO ⇒ ABANDON RULE

GRAMSTAIN ... MORPH

/22
TRACING PARAMETERS

EVALUATE ACTION: Compute CF For
E COLI
UPDATE DATA BASE: IDENT= E.COLI (CF)

DONE w/ RULE Ø156 !

INVOKE NEXT RULE IN UPDATED-BY PROPERTY OF IDENT

E.COLI again ⇒ update CF in DB
ELSE add new fact to DB

KEEP INVOKING Rules (UNLESS CF=1.0) FOR IDENT IN SAME MANNER UNTIL NO MORE

* set "isbeing traced" for ORG-1 (IDENT) OFF
* set "is traced" ON

TRACE REMAINING MAIN PROPS For ORG-1 [STAIN, MORPH ...]

ARE THERE MORE ORGANISMS FOR CULTURE-1 yes ⇒ instantiate ORG-2 ...

13

/23
SEARCH

CONSULTATION IS A SEARCH THROUGH AN IMPLICIT GOAL TREE DEFINED BY THE RULES

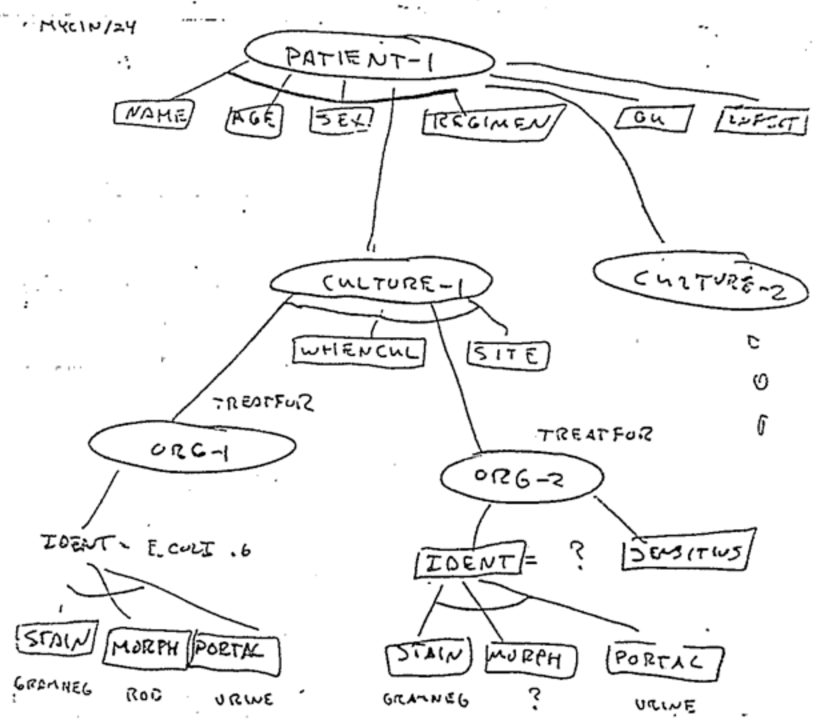
TOP GOAL: FIND THERAPY

SUB GOALS: DETERMINE ORGANISM
GET CULTURES
:

CONTEXT TREE: A "SOLUTION"

ALL LEAF NODES ARE INITIAL FACTS OR LAB DATA

MYCIN/24



AND-OR TREE

AND: Rule Premises
Main Props

OR: Rule Premises
Other Props

(build this dynamically on board)

CONTROL SUMMARY

* TOP LEVEL GOAL RULE
  - IF THERE IS ORGANISM REQUIRING THERAPY <TREATFOR>
  - THEN COMPILE LIST OF THERAPIES & PRESCRIBE OPTIMAL THERAPY
* SEED GOAL
  - PRESCRIBE OPTIMAL THERAPY
* CONSULTATION
  - PROCEEDS BY INVOKING RULES WHICH CONCLUDE ABOUT THE PARAMETER: TREATFOR
* BACK CHAINING
  - w/ MODIFICATIONS

Departures from Back Chaining

MAINPROPS: traced immediatly when context INSTANTIATED

ANTECEDENT RULES: if $CF(parm) = 1$ forward chain on all rules with parm on L.H.S.

SELF-REFERENCING RULES:

META-RULES: find definite conc first.

PREVIEW: Stop if part of premise known false

⇒ more efficient
avoid redundant questioning
easier to modify/update rule base

COMBINING CERTAINTY FACTORS

AND: $CF(C1 \& C2) = \min\{CF(C1), CF(C2)\}$

OR: $CF(C1 \text{ or } C2) = \max\{CF(C1), CF(C2)\}$

COMBINING EXISTING CONCLUSION C1

WITH NEW CONCLUSION C2

$$\text{new } CF = C1 + C2(1 - C1) \qquad C1, C2 > 0$$

$$= \frac{C1 + C2}{1 - \min\{|C1|, |C2|\}} \qquad C1 \cdot C2 < 0$$

REASONING with UNCERTAINTY

INSTANTIATED RULE:

IF SAME ORG-1 GRAM-STAIN (1.0)
& SAME ORG-1 MORPH (.8)
& SAME ORG-1 AIR (.6)

THEN CONCLUDE ORG-1 CLASS = FOO TALLY CF (.8)

CONCLUSION CONFIDENCE = Tally × RULE-CF

$Tally = \min(1, .8, .6) = .6$

$Concl\text{-}CF = .6 \times .8 = .48$

$Combination\ CF = .48 + .1(1 - .48) = .57$

[existing fact: class(ORG-1, FOO, .1)]

ACTION: ADD FACT: CLASS(ORG-1, FOO, .57)

EXPLANATION

[B]ased on 3 capabilities

- Display current rule
- Record rule invocations
- Search Knowledge Base

[W]HY: was a particular question asked <look up>

[H]ow: was a conclusion reached <look down>

RULE ACQUISITION

How to build and later, modify or update Knowledge base

Simple Code generation from English-like rule specifications. KEY WORD Detection

UPDATE PROPERTY LISTS & HOOKS
UPDATED-BY or LOOK-AHEAD

DETECT INCONSISTENCIES / REDUNCIES

ADD NEW RULE ⇒ contradiction
make another rule redundant
is already subsumed by existing rule ⇒ redundant

DIFFICULT PROBLEM

MORE NEXT TIME

EVALUATION

Compares favorably w/ Top Experts in Field

[B]UT KNOWLEDGE BASE INCOMPLETE
COMPUTING INTENSIVE

[W]EAKNESSES!

Model for INEXACT Reasoning Flawed theoretically, May not work for other applications

PROD RULES are Rigid
Not always easy to map knowledge into this form

BACK CHAINING: UNNATURAL. ORDERING RULES THINKING BACKWARD TAKES SOME GETTING USED TO

MYCIN: Conclusion

Production Rules

Reasoning with UNCERTAINTY

BACKWARD CHAINING

KNOWLEDGE ACQUISITION

EXPLANATION

Chapter 6

Explanation

EXPLANATION

- **What is it?**
  - *The ability of a program to explain it's actions.*

- **Why is it a good thing?**
  - To enhance user understanding of
    * the static Knowledge Base
    * the reasoning employed during a consultation
  - To facilitate debugging
  - For education purposes
    * Filling in gaps
    * Tutorial systems
  - To ensure user acceptance.

Questions Asked by Users

- **General Questions**
  - What rules consider symptom Z?
  - What organisms are found in the throat?
  - What dosage of drug Y is usually prescribed
  - What do you prescribe for disease X?
  - etc.

- **Questions about the Consultation**
  - **WHY** was a particular question asked?
  - **HOW** was a particular conclusion reached?
  - **Others:**
    * How was a certain piece of information used
    * What is the current status of parameter X

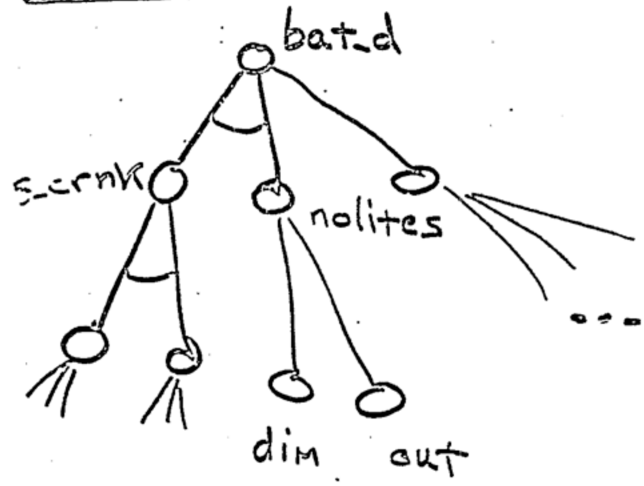
Implementation Steps

1. Identifying question type (*natural language input*)

2. Determining the relevant pieces of knowledge

3. Generating the response
   - General questions:
     - Data base lookup
     - Rule indexing (*updated-by*; *look-ahead*)
   - Questions about the consultation:
     - Examine patient data
       * Data base lookup
     - Justify reasoning
       * Program trace of back-chaining of rules.
       * Essentially: *searching the goal tree*

EXAMPLE



s_crnk & nolites ⇒ bat_d

dim or out ⇒ nolites

HOW

It is concluded that
battery is dead

ES> HOW

Used rule 1

If starter cranks slowly
and lights are not working

Then battery is dead

ES> How lights not working?

used rule 4

IF headlights are dim or out

Then lights are not working

WHY

Check headlights

1) bright

2) dim

3) out

ES> WHY

Working on rule 4

If headlights are dim or o

Then lights are not working

ES> WHY

working on rule 1

If starter cranks slowly
and lights are not
working

Problems

imple rule trace inadequate

- Explanations awkward and verbose
- Some of the rules were control rules
- System has no knowledge about it's rules
(User assumed to know)
- Can't explain why its rules work

ange of questions insufficient

- Is X a good remedy?
(*Judgement*)
- Which is the best remedy: X, Y, or Z?
(*Comparison*)
- How does remedy X work?
(*Procedure*)
- Why doesn't remedy X work?
(*Expectation*)
- What happens if remedy X is used?
(*Causality*)

evel of detail same for all users

Chapter 7

Knowledge Acquisition

KNOWLEDGE ACQUISITION

Definition: The transfer of human expertise to a computer

A difficult problem!

Knowledge: key to any intelligent system

- Basic concepts and relationships
- Reasoning strategies

Major Phases:

1. Initial Creation:
   - ad hoc techniques
   - little automation
2. Debug, Refine, Maintain:
   - various tools exist

Why is it difficult?

- Human knowledge is:
  - complex
  - messy
  - ill-formulated
- Difficult for humans to identify
  - *what* knowledge they possess
  - *how* they apply it
- The *more* expert one becomes, the *less* conscious one is of their knowledge
- The few techniques developed are
  - poorly understood
  - not robust
  - limited in applicability

Fringe Benefits

Sharpen's an expert's thinking

Can reveal gaps in an expert's knowledge
e.g. DENDRAL

Permanent record of knowledge for future generations

Initial Creation: Outline

- Identify the role of the system
  - Nature of the task
  - Degree of interaction
- Characterise the domain
  - Concepts and Structure
  - Strategies used by the expert
- Select representation formalism
- Elicit actual knowledge
  - Interact with expert
  - Encode

The Role of the System

The nature of the task:

- Interpretation (e.g. diagnosis)
- Design & Planning
- Monitoring

⋮

Degree of interaction

- interactive (e.g. advisory system)
- autonomous (e.g. monitoring system)

The Domain

- Concepts and Structure
  - What are significant concepts?
  - How are they related?
- Inference
  - How is new information derived?
- Strategies used by Expert
  - Are tasks reduced to subtasks?
  - If so, is order important?
  - Does optimal order vary from case to case?
  - If so, what are the criteria?
    - * Probability of a fault
      High ⇒ do first
    - * Cost of a test
      Low ⇒ do first
- Choose sub-domain

Representation and Elicitation

Select representation structures

- based on problem characteristics.

Elicit actual knowledge

- Interact with expert.
- Encode

Types of Knowledge:

- Facts
- Rules
- Procedural
- Heuristic
- Causal

Additional Comments

- Measure of Success
  - Easy to encode ⇒
    - * problem well analysed
    - * structures well chosen
  - Hard to encode ⇒
    - * Try again.
- This is an iterative process!

Elicitation Techniques

- Informal Interviews
- Verbal Protocols:
  (think aloud problem solving)
- Observational Studies
  (watch expert in 'natural' setting)
- Automated Techniques
  - Conceptualisation aids
  - Generate rules from examples

Informal Interviews

- Most widely used technique
- Method of recording:
  - Detailed notes
    * Distracting
    * Wastes expert's time
    * Can miss important information
      · Can't write fast enough
      · Importance not recognised at the time
  - Tapes: a better idea
- PROS
  - Quickly get at basic problem structure
  - Requires little of expert's time
- CONS
  - Details are difficult to tease out
    * Hard to ask the right questions
    * Easier with prototype to criticise

Verbal Protocols

- Think-aloud problem solving
- Sessions are recorded and transcribed for later analysis
- PROS:
  - Natural task situation
  - Requires little of expert's time
  - Provides much information about *how* knowledge is used.
- CONS
  - Provides insufficient information about *what* the knowledge is and how it is structured.
  - Interferes with problem solving
    * May affect competence.
    * May try to be more structured than usual
- May effectively be combined with *interviewing*.

Observational Studies

- Passively record actual consultations
- Transcribe and analyse tapes
- PROS
  - Gives insights into what experts *actually* do as opposed to what they *thinks* they do.
  - Good at providing the following information:
    * The role of expert and client
    * The order in which things are done
    * How quickly is problem solved?
    * Is speed important?
    * The nature of the dialogue
    * The total range of knowledge used by the expert.
      · Causal models for explanations
      · Knowledge about the cleint (user)
- CONS
  - Uses much of expert's time and resources

Automated Techniques

- **Conceptualisation aids**
  - A computer system carries on a dialogue with a human expert
    - What are the goals of the system?
    - What are objects and relationships?
    - What are the reasoning processes?
  - Contain knowledge about how to *build* knowledge bases.
  - Helps codify a knowledge engineer's knowledge
  - Few tools exist
- **Machine Induction**
  - Automatically generates a set of rules from a data base of cases
  - Various commercial tools exist

Machine Induction

- Attempting to make general statements about a class of objects (*rules*) based upon particular information about these objects (*cases*).
- A *case* consists of
  - A set of parameters with values
  - A decision category (e.g. diagnosis)
- A *rule* consists of
  - IF condition
  - THEN decision category
- PROS
  - Will account for *all* examples.
  - Easier for experts to cite examples than rules
  - Less need for expert's time if cases already exist.
  - Can be very quick

Machine Induction: CONS

- Still required to manually specify the major concepts
- Not suitable for all domains
  - No random or stochastic processes
  - Substantial data base of cases not always available
- Must carefully choose cases
  - Ensure adequate coverage
  - Sensitive to small changes
- The induced rules
  - may not make '*sense*'
  - may not be what the expert uses
  - may be complex and difficult to understand

Conclusions for Induction

- NOT a major solution to the problem of knowledge acquisition.
- Adequate for some domains
- Not suitable for large systems (yet)
- May be effectively combined with other methods

Knowledge Base Refinement

The second major phase of knowledge acquisition

Goal: To achieve expert performance level

Types of Bugs

- Rules apply in wrong circumstances
- Overlapping rules leading to
  - Redundant conclusions
  - Inconsistent conclusions
- Gaps in rule set
- Rules interact unexpected ways (Must consider control structure)
- Rules become outdated with new discoveries (maintenence)

Various tools available

SUMMARY: Knowledge Acquisition

- The crucial aspect of building intelligent systems
- Two main phases:
  - Initial Creation
    - Identify task
    - Characterise domain concepts
    - Formalise
  - Refining and Extending
- A difficult and poorly understood area.
- Best to use a variety of techniques
- Various tools have been developed
  - many research systems
  - some commercial systems

Chapter 8

Structured Objects

STRUCTURED OBJECTS

KNOWLEDGE REPRESENTATION FORMALISMS

LOGIC: Collection of assertions & Rules of Inference
THEOREM PROVER

PRODUCTION RULES: Collection of Rules: IF <Cond> THEN <ACT>
RULE INTERPRETER

NO EXPLICIT STRUCTURE

⇒ NEED EFFICIENT INDEXING

EXAMPLES

LOGIC

(1) contains(uk, scotland)
(2) lives_in (tom, scotland)
(3) brother-of (nigel, tom)
(4) climate-of (scotland, wet)
⋮

scotland [1-2, 2-
tom: [2-1, 3-
⋮

for **UNIFICATI**

RULES

(1) c & a ⇒ b
(2) d ⇒ h
(3) a ⇒ k
(4) b ⇒ h
⋮

updated-by (b, [1])
updated-by (h, [2, 4])
⋮
look-ahead (a, [1, 3])
⋮

for **MATCHING**

STRUCTURED OBJECTS: an ALTERNATIVE

EXPLICITLY GROUP RELATED ASSERTIONS INTO LARGER STRUCTURES

TWO FLAVORS

Graph Structures
* nodes
* arcs

} SEMANTIC NETS

Record Structures
* slots
* fillers

} FRAMES

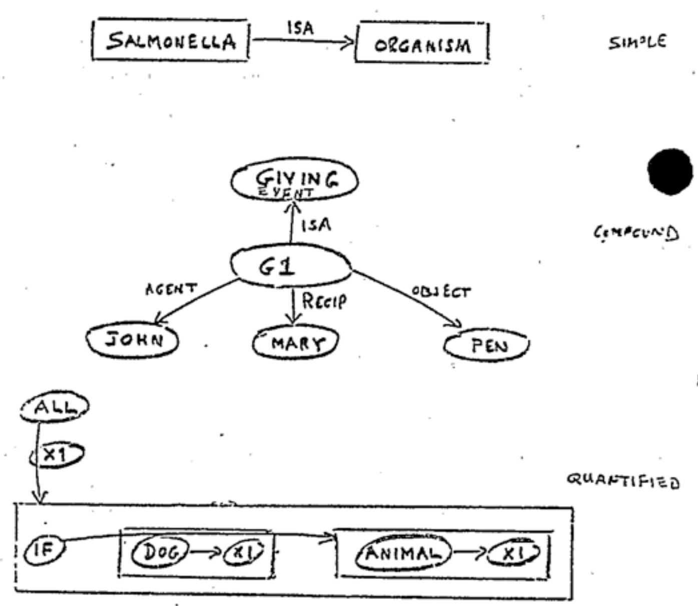
SEMANTIC NETS

ODES: Objects John; pencil; gum
Concepts OWNERSHIP
Situations Selling, Giving ...

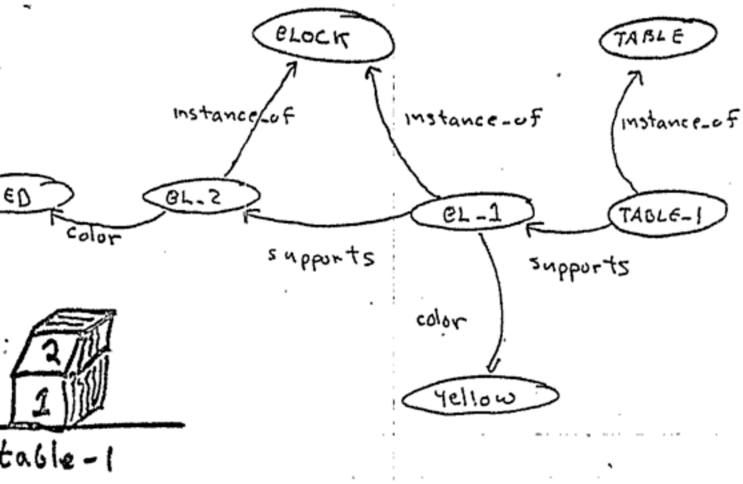
RCS: Relations between nodes
- IS_A
- agent
- likes

ROBLEM SOLVING:
graph search
net operations
use of proximity

Examples of SN structures



EXAMPLE



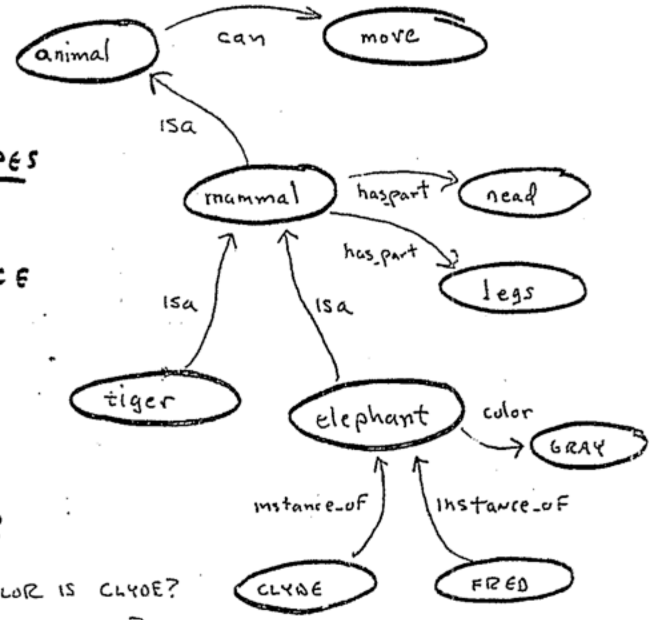
instance-of (table-1 , table)
instance-of (bl-1 , block)
instance-of (bl-2 , block)
color (bl-1 , yellow)
color (bl-2 , red)
supports (table-1 , bl-1)
supports (bl-1 , bl-2)

"ISA" HEIRARCHY

INFERENCE MECHANISMS
(chasing Pointers)

NODE TYPES

ACTION
CLASS
INSTANCE
VALUE
PART



QUERY:

WHAT COLOR IS CLYDE?
CAN FRED MOVE?

4

WIDELY USED

* psychological memory models
* meaning in natural language
* causality

NOT RIGIDLY DEFINED!

FRAMES

ANALOGOUS TO RECORD STRUCTURES
- SLOT-FILLER NOTATION

COMPLEX NODES IN SEMANTIC NETS
- SLOTS:
  - ATTRIBUTE-VALUE PAIRS
  - DEFAULT VALUES
  - MAY BE PROCEDURES

USEFUL FOR MODELLING
- EXPECTATIONS & ASSUMPTIONS
  - eg. disease models
  - visual scenes

FRAMES (cont'd)

* STEREOTYPICAL SITUATIONS / OBJECTS
* SCENARIO
* CONSISTS of Various info
  - procedures:
    - how to use frame
    - future expectations
    - how to recover from failed expectations
    - how to fill slots
  - pointers to parent & sub-frames
  - basic data: name, size, ⋮

6

EXAMPLE

'DOG' FRAME

NAME: ______

OWNER: ______

ADDRESS: ______

LICENSE NO ______

IF (DOG APPEARS LOST
THEN CONTACT OWNER)
[INVOKE "CONTACT PERSON" FRAME]

EXAMPLE

SIMPLE FRAMES EASILY REPRESENTED IN LOGIC AS WELL.

catch-object (jack-2, ball-1)

catch-object (id (catch-22)
catcher(jack-2)
caught (ball-1))

PREDICATE : FRAME TYPE

ARGUMENTS : SLOTS

EXAMPLE

JANE WAS INVITED TO JACK'S BIRTHDAY PARTY

SHE WONDERED IF HE WOULD LIKE A KITE

Friend: He already has a kite.
He will make you take it back

SHE WENT TO HER ROOM AND SHOOK HER PIGGY BANK

IT MADE NO SOUND

Possible FRAMES USED:

Birthday Party Frame
- DRESS =
- PRESENT
  - WHAT WILL HE LIKE
  - BUYING SUBFRAME
    - OBTAIN MONEY PIGGY BANK SUBFRAME
    - SELECT STORE
    - :
- FOOD ICE CREAM, CAKE
- DECOR BALLOONS, etc.

CONCLUSION

FRAME: Set of questions about a hypothetical situation.

Conceptually: what are relevant issues?
WHAT are methods for dealing w these issues?

Implementation View:
- Generalised Record Structure
- Slots to Fill
- Rules for Filling slots
- Default Assumptions

FRAME SYSTEMS:
CREATE, MODIFY, DESTROY frames

REPRESENTS AND USES KNOWLEDGE IN FAIRLY NATURAL

— PSYCHOLOGICAL VALIDITY —

8

ISSUES for SELECTING a FORMALISM

* "NATURALNESS"
  How easy to cast a problem into a framework?

* EXPRESSIVENESS
  Can you represent all that you need to?

* INFERENCING (REASONING)
  - Types: Deduction/Abduction/Induction, Uncertainty
  - Mechanisms: Implementation, Efficiency / Indexing

* CONTROL FLEXIBILITY / FC / BC

SUMMARY

RULES
- 'NATURAL' (sometimes)
- ? MODULAR
- EXPLANATION "TREE"
- ~ INDEP OF CONTROL
- EFFICIENT

- * NO STRUCTURE
- * EFFECT OF NEW RULES UNPREDICTABLE
- * EXPLANATION CLUMSY (rule trace)
- * NO BACKTRACK

LOGIC
- INFERENCE GUARANTEED CORRECT
- PATTERN MATCHING
- GOOD ANALYTICAL TOOL
- expressive

- * CLUMSY TO ENCODE
- * NO REASONING W/ UNCERTAINTY
- * NO EXCEPTIONS/D
- * INEFFICIENT INFERENCE
- * NO STRUCTURE

STRUCT OBJECTS
- EFFICIENT INFERENCE
- 'NATURAL'
- EXCEPTIONS & DEFAULTS
- Flexible CONTROL

- * NO BACKTRACK
- * LESS EXPRESS

TRADEOFFS

EXPRESSIVE POWER

VS

EFFICIENCY

eg: SEMANTIC NETS VS LOGIC

ELEGANCE & UNIFORMITY

VS

EFFICIENCY

eg: LOGIC FOR PLANNING

YOU MUST ASSESS YOUR OWN NEEDS !

SUGGESTED READING

Charniak & McDermot
INTRO TO AI.

Jackson
INTRO TO ES

SEMANTIC NETS
FRAMES

COMPARISON w/
LOGIC & F

Chapter 9

INTERNIST

INTERNIST

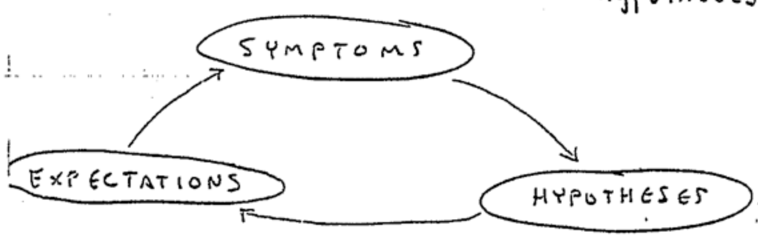
CH 8 Intro ES Jackson

* Real world application
* Model human reasoning
* BIG
  - 600 diseases
  - 4000 symptoms
  - 10,000 links
* 10 YRS DEVELOPMENT

  GOAL: Clinical Use!

MODEL HUMAN DIAGNOSTIC LOGIC

- patterns of symptoms EVOKE hypotheses
- hypotheses give rise to EXPECTATIONS about the presence or absence of other symptoms
- further observations REFINE these hypotheses



DATA-DRIVEN: Symptoms → Hypos ~ FC?

MODEL-DRIVEN: Hypos ← Other Symptoms ~ BC?

DIAGNOSTIC LOGIC

PROBLEMS w/ MYCIN REASONING

* MAY NOT BE ABLE TO REASON DIRECTLY TO GOAL Because:
  - NOT ENOUGH INFORMATION TO INFER DISEASE AREA
  - NO STRUCTURE IN RULE SET: SYMPTOMS SCATTERED THROUGHOUT
* SEARCH SPACE TOO LARGE for EXHAUSTIVE DFS
  - 2,000 – 10,000 DIAGNOSTIC CATEGORIES
  - patients can suffer ten or more diseases concurrently

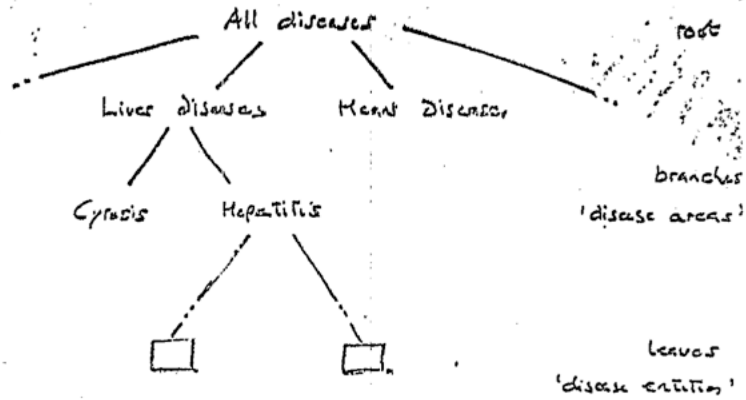
⇒ NEED MORE EFFICIENT APPROACH

DIAGNOSTIC LOGIC

* MUST EXPLOIT ANY INHERENT STRUCTURE
* APPROXIMATE REASONING MUST BE MORE SOPHISTICATED THAN PROPAGATION OF MYCIN-LIKE UNCERTAINTY FACTORS
  eg: COST, IMPORTANCE, RISK ...

3-STEP JUDGEMENT PROCESS

- SYMPTOMS SUGGEST DISEASES
- DISEASES SUGGEST CO-OCCURING SYMPTOMS
- DIFFERENTIATE COMPETING HYPOTHESES
  - CONSIDER AVAILABLE EVIDENCE
  - GROUP INTO MUTUALLY EXCLUSIVE SUBSETS TO ENABLE RULING OUT OPTIONS



DIAGNOSIS : A set of leaf nodes

The Disease tree is a static data structure, however, it plays an active role in directing reasoning.

There are no rules, only 'models'.

5

KNOWLEDGE BASE

* Diseases - Tree heirarchy

\+ Manifestations - History, symptoms, physical signs, Lab findings etc

\+ Relations -

EVOKE - manifestation suggests a disease [0-5] ≈ P[D/M]

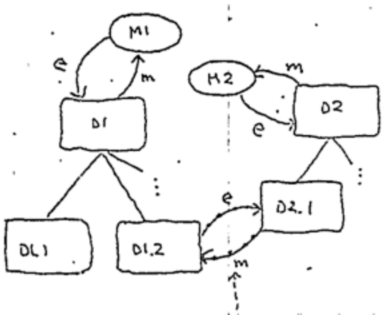
MANIFEST - diseases manifest symptoms (Frequency) [1-5] ≈ P[M|D]

TYPE - cost of tests, risk to patient

IMPORT - importance of manifestation ie can it be ignored? [1-5] Is it CRUCIAL?

MISCELLANEOUS - CAUSAL, TEMPORAL, other associations

STATIC STRUCTURES



Diseases may be caused by other diseases and thus be a manifestation of another disease

DISEASE "FRAME"

ID: D1.2
FORM-OF: D1
MANIFESTATIONS: [M3 (evoke, Frequency)
M4 (---
⋮]
PRE-DISPOSES TO: D2.1
CAUSES: ?
COINCIDENT-WITH ?
PRECEDES: ?

CONTROL: Overview

* ENTER PATIENT DATA
- positive
- negative

* HYPOTHESISE DISEASES WHICH EXPLAIN the DATA {abduction / data-driven

* SELECT MOST 'PROMISING' HYPOTHESES TO EXPLORE {heuristics

* ELICIT MORE DATA TO CONFIRM/DENY COMPETING HYPOTHESES {deduction / hypo-driven

FOCUSSING SEARCH

* CREATE MODEL FOR EACH 'active' DISEASE AREA
  categorise relevant findings
* SCORE & PARTITION MODELS
  measure explanatory power

DYNAMICALLY SELECT STRATEGY FOR CONTINUING DIAGNOSIS MOST EFFECTIVELY
heuristic guidance

9

[IN]TERNIST CONTROL

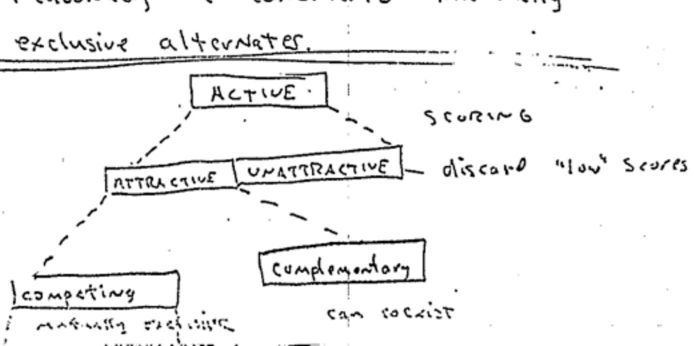
Scoring: Measure explanatory power of all active disease hypotheses

Points Awarded for each manifestation explained by it. More important ⇒ more pts

Points Removed for each manifestation expected but not found. More points, the stronger the disease suggests the finding [MANIFEST relation]

Bonus Points for causally related diseases already confirmed.

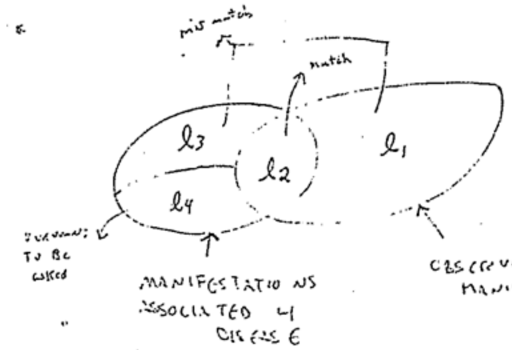
[Pa]rtitioning: Groups the top ranked model with those diagnoses that may reasonably be considered mutually exclusive alternates.



DYNAMIC DISEASE MODELS

4 LISTS

1) SYMPTOMS OBSERVED NOT ASSOC w/ D
   NOT EXPLAINED BY D
2) SYMPTOMS OBSERVED ASSOCIATED WITH D
   MIGHT BE EXPLAINED BY D
3) SYMPTOMS NOT OBSERVED, ASSOCIATED w/ D
   EVIDENCE AGAINST D
4) SYMPTOMS NOT MENTIONED, ASSOCIATED w/ D
   THINGS TO ASK ABOUT TO HELP CONFIRM OR DENY D



DIAGNOSTIC STRATEGIES

Strategy for deciding between competing hypotheses depends on how many there are.

* N ≥ 4 RULEOUT
  Ask questions which are almost certain to occur in competing diseases.
  Hope for NO ⇒ Can Rule that one out!
* N = 2, 3, 4. DISCRIMINATE
  Ask questions which support one model at the expense of another.
* N = 1 VERIFY
  Ask questions which strongly support leading contender. [EVOKE property: h...

META-LEVEL Reasoning.

THE LIFE OF A Node in the Disease Tree

All nodes initially inactive

Node (hypothesis, disease area) activated [EVOKED] when ≥ 1 unexplained Finding which is indicitive of the disease is present

Remain active until
- can discriminate children

or)
- questioning exhausted
- concluded

Semi-Active if parent disease node is active. Waiting for evidence to back it and kick parent out. ⇒ Progress toward solution state

Concluded iFF has no competitors
or way ahead of next most likely.

Conclusion ⇒ all observed manifestations explained by it are removed from future consideration. Marked "explained"

⇒ Activate semi-active "children"

13

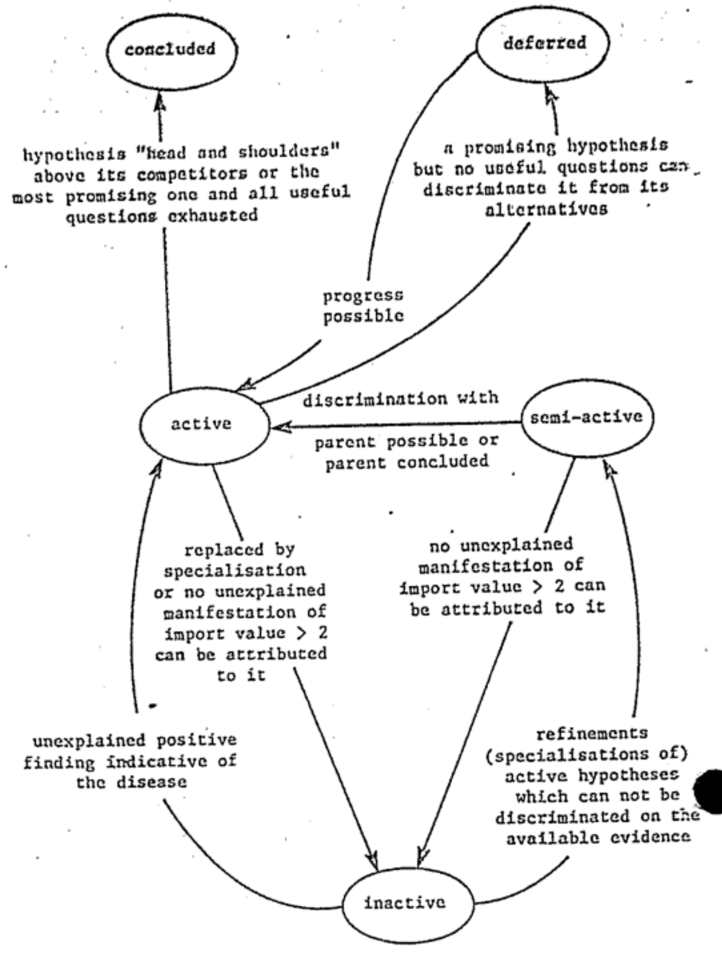


Fig. 5-3: Hypothesis status transition diagram.

From "EXPERT SYSTEMS TECHNOLOGY A Guide" Johnson & Keravnou

SHORTCOMINGS

* Cannot consider more than one disease at a time
* Assumes in any patient, a manifestation has only one cause
  (since it's removed when a disease is confirmed)
* Cannot undo previous conclusion
* No intermediate states of diseases
* Poor explanation Facility
* Initial focussing when multiple diseases present was poor ... Sequential instead of parallel

CADUCEUS: Trying to solve these problems ONGOING!

SUMMARY: INTERNIST

* MODELLED HUMAN DIAGNOSTIC REASONING
* REALISTIC SIZE & COMPLEXITY
* SEARCH REDUCTION
  - HEIRARCHY OF DISEASE "FRAMES"
  - HEURISTICS For Choosing which hypothesis to pursue
  - Mix Forward & Backward reasoning
* NO RULES
* REASONING with UNCERTAINTY – SCORING FUNCTION –

Chapter 10

CENTAUR

CENTAUR: Rules *and* Frames

- An experiment in representation and control
- Motivation
- Knowledge Base
- Control Structure
- Comparisons with INTERNIST
- Summary

Historical Background

- The Domain: Pulmonary (lung) Disease
- The Problem: Interpret test measurements
  - Identify any likely diseases
  - Gauge the severity

MYCIN ⇒ EMYCIN ⇒ PUFF ⇒ CENTAUR

Motivation

- **Various types of knowledge:**
  - Structural (e.g. taxonomic)
  - Heuristic (rules of thumb)
  - Causal (first principles)
  - Control (problem solving strategies)

  ⋮

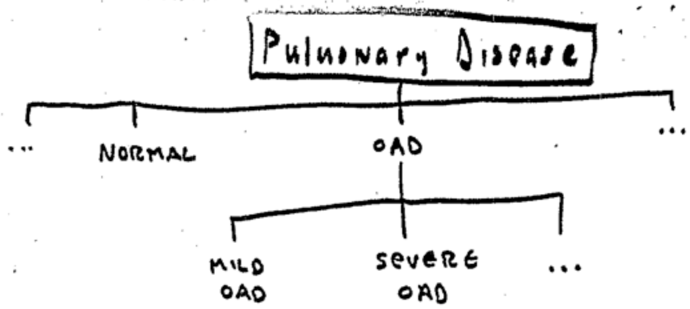
- **Too much to ask of a single formalism:**
  - Gives rise to unprincipled kludges (e.g. to achieve question ordering)
  - Functional distinctions blurred
  - Important information hidden away

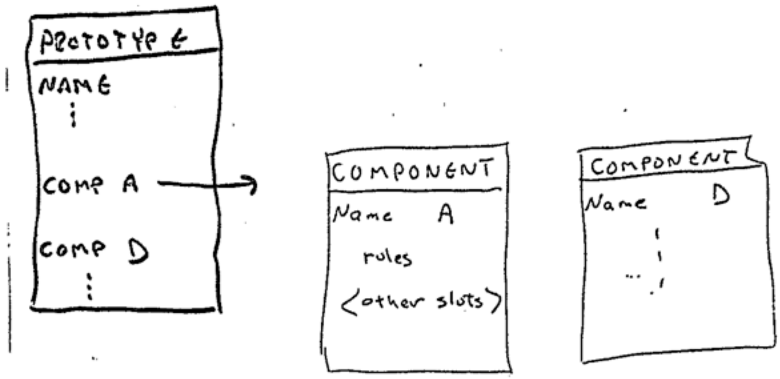
Problems with Rules

- Cannot represent prototypical situations
  - Cannot reason about *expectations*
    - What objects to find?
    - What events should occur?
    - Default values
  - Cannot reason about *exceptions*
  - Cannot generate high level hypotheses with incomplete information
- Difficult to exploit structural regularities
- Inflexible control mechanism
- Explanation awkward, especially with control rules.
- Difficult to update/maintain knowledge base
- **FRAMES to the rescue!**

The CENTAUR Solution

- Mix Rules *and* Frames
- Knowledge Base consists of three frame-like objects, and rules.
  - *Prototypes*:
    - characterise typical features of each disease
    - linked in a hierarchy
  - *Components*
    - A feature of the prototype (a subframe)
    - Each component has *rules* (in slots) which are used to deduce a value.
      (A 'natural' way to classify rules)
  - *Rules*
    - Classified according to function
  - *Facts*
    - Specific data for a case





HOW IT WORKS:

Match test results and patient data with one of the stored prototypes.

Test "current" prototype by exploring it's components.

Inference rules used to infer component values

Prototypes guide invocation of prod rules. Focusses search for new info on "most likely prototype" assessment

SEPERATES OUT:

CONTROL KNOWLEDGE

SITUATIONAL KNOWLEDGE

KNOWLEDGE BASE

PROTOTYPES

Slot Name	Type of Info	
Author, Date, Source	BookKeeping Info	GENERAL SLOTS
More General, More Specific, Alternate	Pointers to other Prototypes Heirarchy	GENERAL SLOTS
Explanation, Hypothesis	English Phrases	GENERAL SLOTS
To-Fill-In, If-Confirmed, If-Disconfirmed, Action	How to proceed in Future	CONTROL SLOTS
Fact Residual Rules, Refinement Rules, Summary Rules		RULE SLOTS
Components		

KNOWL BASE Ctd

COMPONENTS (of prototypes)

- Component Name eg: TOTAL LUNG CAPACITY
- Default Value - If none given or deducible
- Plausible Values -
- Possible Error Values
- Importance Measure
- Inference Rules - To deduce a value
- Actual Value

A PROTOTYPE WITH ALL COMPONENT VALUES SET TO DEFAULT VALUE CORRESPONDS TO A "TYPICAL PATIENT"

Facts

- Name
- Value
- CF

(Like MYCIN)

- Origin (*user*, *rules*, *default*)
- Classification with respect to prototypes (*plausible*, *error*, *surprise*)
- Justification
  - Which confirmed prototypes account for it?
  - Used to determine which facts are unexplained.

Control Structure

A simple interpreter: 'Hypothesise and Match'

- Simple task agenda
- Considers exactly one prototype at any one time.

Control rules embedded in Prototypes

- tell interpreter what to do at each of four stages in the consultation:
  (i.e. add tasks to agenda)
  * How to instantiate
    (what data to collect)
  * When confirmed
    (what prototype to explore next)
  * When disproved
    (what prototype to explore next)
  * When consultation ends
    (Print summary)
- Each rule can be thought of as the consequent of a rule whose antecedent matches the situation described by the prototpye

Rules

A Functional Classification:

Inference rules: For determining parameter values for components

Triggering rules: Suggest hypotheses for consideration given initial data (like INTERNIST *evokes* relation)

Refinement Rules: Guide further information gathering to confirm/deny current hypotheses.

Fact residual rules: Account for unexplained data *after* a hypothesis has been confirmed.

- May find co-occurring disease

Summary rules: Provide for output of results in English.

...g a consultation a prototype is in 1 of 3 states:

- inactive – not considered as an hypothesis
- relevant – suggested by data values
- active – placed on the <u>hypothesis list</u>

...are also lists for confirmed & disconfirmed hypotheses

...are 3 different strategies for <u>selecting</u> a prototype

- confirmation – pursue best match
- elimination – pursue worst match
- fixed-order – prespecified

STATICALLY DETERMINED BEFORE CONSULTATION

Less flexible than INTERNIST in this Regard

STAGES IN CONSULTATION

1) Enter Initial Data
2) Trigger Prototypes – TRIGGERING RULES, Ind Certainty measure of suggested Prototypes
3) Scoring Prototype
4) Select best Prototype
5) Fill in "Current" Prototype: Infer new Facts. Return to 2 if new PT's suggested
6) Test Match Does it "Fit" well enough?
   - IF-CONFIRMED: tasks } Suggest further PT's
   - IF-DISCONFIRMED: tasks }

   Back to step <u>3</u>
7) Account for Data Fact Residual Rules
8) Refine Diagnosis Refinement Rules
9) Summarising Results Summary Rules
10) PRINT Results Action Slot of Confirmed PT's guide printing <added to agenda>

FINAL INTERPRETATION

LIST of confirmed prototypes

- trigger values: findings suggestive of PT
- plausible : findings consistent with PT
- error/surprise: findings inconsistent with PT
- residual : findings not accounted for by <u>any</u> PT

LIST of <u>dis</u> confirmed prototypes

SUMMARY

<u>REPRESENTATION</u>

* CONTROL KNOWLEDGE EXPLICIT (NOT HIDDEN IN RULES)
* RELATED RULES ARE STRUCTURALLY LINKED (INTERACTIONS NOT HIDDEN)
* ALL THE KNOWLEDGE ABOUT A DISEASE IS LUMPED TOGETHER
* EASY TO ADD KNOWLEDGE

<u>REASONING & CONTROL</u>

* CLOSER TO PHYSICIAN'S REASONING (≈ INTERNIST)
* EASY TO CONTROL QUESTIONING
* ONLY ASKS RELEVANT QUESTIONS
* INCONSISTENT INFORMATION IS INDICATED

CENTAUR ...

INTERNIST vs CENTAUR

Facts: ≈ Manifestations

[C]ENTAUR only: Classify Facts — Plausible, ERROR, surprise

⇒ can deal w/ erroneous and inconsistent values

INTERNIST would chase down more and more unlikely HYPO'S

[I]NTERNIST: NO Rules!
No Explanation!

[B]OTH: Heirarchy of Disease

SIMILAR TO HUMAN REASONING

[CO]NTROL: Both use PROTOTYPES to guide control.

CENTAUR has explicit control info in its PT's ⇒

- more flexible
- Allows explanation
- Knowl acquisition

Chapter 11

MECHO

MECHO

Ref: A. Bundy, The Computer Modelling of Mathematical Reasoning, pp 191 – 205

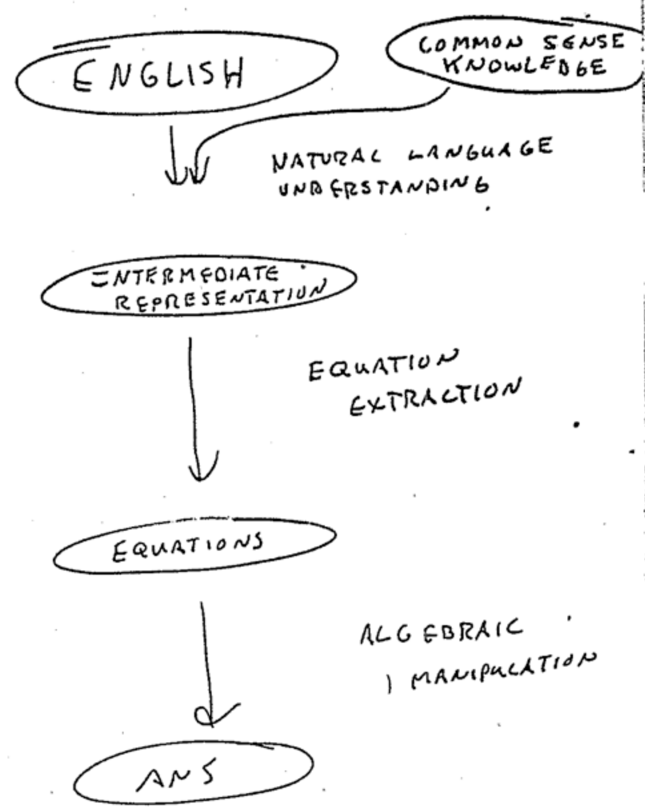
JACKSON, INTRO TO ES CHAPTER 11

Found in SB or EHL LIBRARIES

- LOGIC-BASED KNOWLEDGE REPRESENTATION

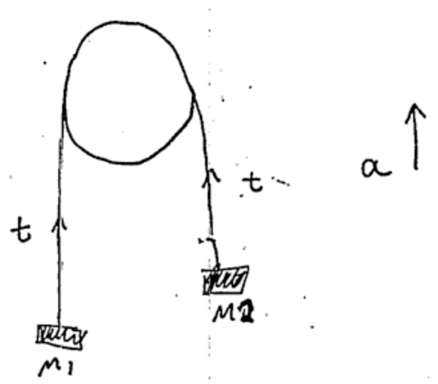
SOLVES PHYSICS PROBLEMS STATED IN ENGLISH

OVERVIEW



PROBLEM:

- "Two particles of mass m1 and m2 lbs are connected by a light inextensible string passing OVER A SMOOTH PULLEY. Find the acceleration of the particles and the tension in the string"



A PULLEY SYSTEM

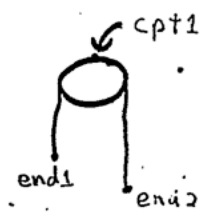
KNOWLEDGE REPRESENTATION in LOGIC

WHAT SORT OF OBJECTS?

isa(part1, particle)
isa(part2, particle)
isa(str, string)
isa(pull, pulley)

PARTS OF OBJECTS:

end(str, end1, left)
end(str, end2, right)
contact-pt(str, cpt1)



KNOWL REPN CONT'D

RELATIONSHIPS Between OBJECTS

Low Level!

contact (end1, part1)
contact (end2, part2)
contact (cpt1, pull)

partition (Str, < bit1, bit2>)

incline (bit1, end1, 90)
incline (bit2, end2, 270)

concavity (bit1, stline)
concavity (bit2, stline)

KNOWLEDGE REPN CONT'

ASSOCIATING PHYSICAL QUANTITIES to OBJECTS

mass (part1, mass1)
measure (mass1, lbs, m1)

accel (part2, acc, 90)
measure (acc, ft/sec², a)

friction (pull, ø)
extensibility (str, ø) } no dimension

THE PROBLEM:

Sought (acc) given (mass1)
Sought (tsn) given (mass2)

ECHO/7

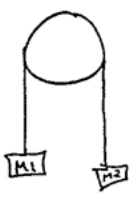
SCHEMATA

STANDARD CONFIGURATIONS

eg: pulley system

isa (part1, particle) &
isa (part2, particle) &
isa (Pull, pulley) &
isa (Str, string) &
end (Str, End1, left) & end (Str, End2, right)
& contact-pt (Str, Cpt) &
contact (End1, Part1) & contact (End2, Part2) &
contact (Cpt, Pull)

→ pulley-sys (Pull, Str, Part1, Part2)



* Bring in unstated assumptions (Defaults)

eg: pulley-sys → Frictionless
pulley-sys → ∃ partition <Bit1, Bit2>

INFERENCE

* Data base lookup

* Prolog depth first, left to right

* Creative Call:

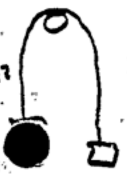
Generate placeholders and continue computation

MECHO explicitly controls the amount of work done by varying depth of call.

INFERENCE

tension(Bit2, Tsn) &
partition(Str, <Bit1, Bit2>) &
extensibility(Str, ∅), &
pulley-sys(Pull, Str, Part1, Part2) &
friction(Pull, ∅)

⟹ tension(Bit1, Tsn)



In a pulley system with inextensible string and frictionless pulleys the tension in both bits of string is the same   ALSO for Acceleration

MECHO16 **FORMULATING INTERMEDIAT**

I ENGLISH STATEMENT ⟹ LOW LEVEL INFO

II SUGGESTS CERTAIN SCHEMA (Scenarios, Prototypical Situation)

III INFER MISSING INFORMATION NEEDED IN SCHEMA

eg: Tension is same on both string.

OR Fill in w/ Defaults

COMPARE w/ INTERNIST & CENTAUR

MECHO/9

EQUATION EXTRACTION

- WHICH EQUATIONS TO FORM?
  MARPLES ALGORITHM
- INFERENCE – Bridge gap between info given and that needed
- SCHEMATA – Supply background knowledge, defaults etc

PHYSICAL FORMULAE

is_formulae(resolve-forces, <name>
situation(Object, Dir),
<formula> F = M·A,
<context> {mass(Object, M) &
accel-component(Object, A, Dir)
sum-forces(Object, A, Dir) F}

→ relates situation to formula

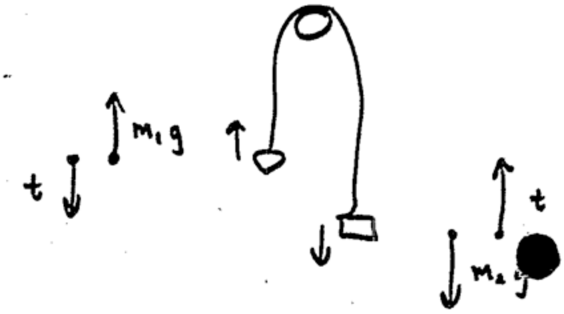
CHOOSING EQUATIONS <MARPLE'S Alg>

1) Analyse sought quantity (acceleration)
   what sort of quantity?
   WHAT situation?
   <eg ACC of P2 in dir 90° >
2) Select Formula + Situation
   resolve relates accel, situation(P2, 90)
3) Fill in blanks (inference)
   F, M, A
4) Does new unknown need adding?
   yes ⇒ pause, try other option
   no ⇒ ok
5) If All options require new UNKNOWNS
   go back to pause, invent one. Mark as sought (Var)
6) Check independence
7) Repeat For all sought quantities
8) Resolve units

MATHEMATICAL MODEL

$m_1 \cdot g - t = m_1 \cdot a$

$t - m_2 \cdot g = m_2 \cdot a$



NEXT STEP: Pass to Equation sol

ECHO/13

SEARCH STRATEGY

* BACKWARD CHAINING?

* MEANS ENDS ANALYSIS?

* META-LEVEL INFERENCE

  EXPLICIT RULES FOR DECIDING WHAT TYPES OF INFERENCES TO MAKE.

  - DON'T CREATE AN UNKNOWN VARIABLE UNLESS YOU HAVE TO
  - Rules only apply in appropriate situations
  - Explicit "relate" ing of quantities to ease the search for possible formulae
  - Types of arguments reduce

GENERALITY

MBASE: Logic-based representatio & inference formalism
   special purpose
   domain independent

   Marples Algorithm

Predicate Library: Domain dependent kn
   pulleys, levers, roller coasters, moments of inertia

   Thermodynamics
   Ecology

MGCHOK:

SUMMARY

* SOLVES PHYSICS PROBLEMS STATED IN ENGLISH
* USES A LOGIC-BASED KNOWLEDGE REPRESENTATION FORMALISM
* "PURE" PROLOG TOO WEAK, BUT PROLOG CAN EASILY BE USED TO INTERPRET EXTENSIONS OF PROLOG (as in KRF)
* INTERMEDIATE REPRESENTATION
* INFERENCE TO BRIDGE GAPS
* SCHEMATA TO PROVIDE IMPLICIT ASSUMPTIONS (ie defaults)
* Meta-level Inference to control search

Chapter 12

Conclusions

REVIEW

REPRESENTATION & INFERENCE

LOGIC
- MECHO

PRODUCTION RULES
- MYCIN

STRUCTURED OBJECTS
- INTERNIST
- CENTAUR

KNOWLEDGE ACQUISITION

EXPLANATION

BUILDING YOUR OWN EXPERT SYSTEMS

CONCLUSION

OUTLINE

* Fundamental Issues
* Available Tools
* CHOOSING THE RIGHT TOOL
  - Representation
  - Control
* STAGES of Development
* SUCCESS STORIES
* STATE of THE ART
* LOOKING TO THE FUTURE

BUILDING YOUR OWN EXPERT SYSTEM

FUNDAMENTAL ISSUES

* KNOWLEDGE REPRESENTATION
* INFERENCE (uncertainty?)
* CONTROL (FC? BC? M.+?)

OTHER ISSUES

* KNOWLEDGE ACQUISITION
* DIALOGUE - QUESTIONING STRATEGY, EXPLANATION
* VALIDATION

CONCL/3

AVAILABLE TOOLS

GENERAL PURPOSE:

* PROGRAMMING LANGUAGES
* HIGH LEVEL PROGRAMMING ENVIRONMENTS
* SHELLS

(general ↑ specific; much effort ↓ little effort)

OTHER AIDS:

* KNOWLEDGE ACQUISITION
* DIALOGUE TOOLS (natural language interface?)

PROGRAMMING LANGUAGES

CONVENTIONAL: FORTRAN, PASCAL, ...
Procedural, Numeric

SYMBOLIC:
- **PROLOG:** Logic-based
- **LISP:** Procedural

MAXIMUM FLEXIBILITY & POWER
TREMENDOUS SET UP COST

EXPERT SYSTEM SHELLS

SPECIAL PURPOSE EXPERT SYSTEM — eg MYCIN

⇓

STRIP OFF DOMAIN DEPENDENT CODE

⇓

EXPERT SYSTEM SHELL — eg EMYCIN

INTACT:
- KNOWLEDGE REPRESENTATION
- INFERENCE MECHANISMS
- CONTROL STRATEGY

SHELLS

ADD KNOWLEDGE FROM NEW DOMAIN — eg PUFF, SACON

* PRESTO * ANOTHER EXPERT SYSTEM.

PROS	CONS
VERY FAST	Rigid: KRF, Control
QUICK PROTOTYPING	Special Purpose (Diagnosis)
WORK FOR MANY DOMAINS	Inappropriate
MANY FEATURES: explanation, acquisition aids, ...	

SEVERELY LIMITED

EXAMPLE

EMYCIN:

* Depth-1st Backtracking unsuitable for long chains of reasoning (too much search)
* Reasoning w/ uncertainty techniques rendered useless for long inference chains
* Need to "fudge" rules to achieve special effects (eg control)
* Dialogue Control limited

LESSON: MUST CAREFULLY SELECT DOMAIN

ROGRAMMING ENVIRONMENTS

NTERMEDIATE SOLUTIONS

PROVIDE USER WITH EASY ACCESS TO A VARIETY OF SPECIAL PURPOSE FEATURES

KNOWLEDGE REPRESENTATION & INFERENCE
- RULES
- FRAMES
- NETWORKS
- LOGIC

CONTROL
- FORWARD CHAINING
- BACKWARD CHAINING
- MIX

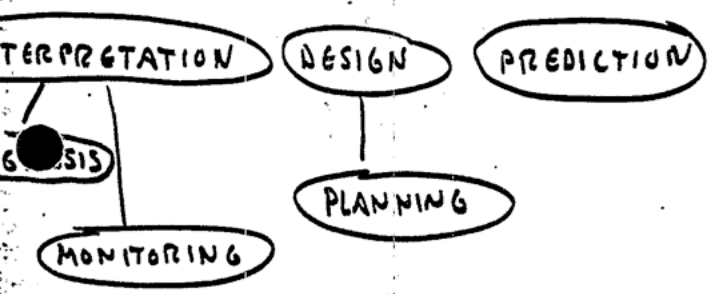
PROGRAMMING ENVIRONMENTS

PROS	CONS
FLEXIBLE	NO GUIDANCE
MANY OPTIONS	

WE STILL KNOW VERY LITTLE ABOUT SELECTING APPROPRIATE FORMALISMS & CONTROL STRATEGIES FOR A PARTICULAR PROBLEM

HOW TO INTEGRATE MIXED FORMALISMS & STRATEGIES?

TASK TYPES



EPAIR: (debugging) — INTERPRETATION & DESIGN

STRUCTION: — INTERPRETATION & DESIGN

NTROL: — INTERPRETATION, DESIGN, MONITORING

CONCL/70

CHOOSING A REPRESENTATION

QUESTIONS TO ASK:

- Does domain have inherent structure?
  eg taxonomy
  Yes ⇒ exploit it. Structured Objects
- DYNAMIC vs STATIC Data
  STATIC ⇒ less flexible representation scheme will suffice
- How large is body of data? Rule Set?
  Large ⇒ must use indexing
  exploit inherent structure (if any)
  - partition rule set by task
  - levels of abstraction (INTERNIST)

CONCL

CHOOSING A CONTROL STRUCTURE

Exhaustive Search infeasible
⇒ o may not find sol
o Sol may be sub-optimal

QUESTIONS TO ASK:

- EXPERT reasoning model-driven or data-driven?
  <back chain> <for chain>

- WILL ONE GLOBAL CONTROL SCHEME WORK?
  Yes ⇒ simpler

- Do you need optimal sol?
  No ⇒ great!

  yes ⇒ may need scoring function & traversal algorithms.
  May not exist!

- META-LEVEL CONTROL

  Program Reasons about control @ run time

- WHAT IS THE NATURE OF THE SEARCH SPACE?

OTHER TOOLS

* KNOWLEDGE ACQUISITION

  LEARNING from EXAMPLES

  These tend to be complete systems which take the rules automatically generated and integrate within existing framework
  ⇒ LIMITED TO SMALL CLASS OF PROBLEMS

* KNOWLEDGE BASE REFINEMENT AID
  USUALLY AN "ADD ON" TO EXISTING RIGID FRAMEWORK

* DIALOGUE:
  NATURAL LANGUAGE INTERFACES
  LARGELY UNDERDEVELOPED AREA OF RESEARCH

WHAT WE REALLY NEED...

FLEXIBLE TOOLKITS

WITH GUIDANCE!!

ANALYSE DOMAIN & TASK

?⇒ Representation formalism(s)

Control structure(s)

VERY early days: ACTIVE RESEARCH AREA

CONCL/4

ES's: STAGES OF DEVELOPMENT

+ SYSTEM DESIGN
* SYSTEM DEVELOPMENT
* FORMAL EVALUATION of Performance
* Formal evaluation of Acceptance
* EXTENDED use w PROTOTYPE environment
* Development of maintainence plans
* System Release

VERY FEW GET BEYOND THIRD STAGE

TRULY SUCCESSFUL SYSTEMS

CON: Computer Configuration

UFF: Interprets Medical Instrument Data

CE: Troubleshoots telephone networks

ACSYMA: Algebraic expression manipulation

+

Many other "almost made it"

PROSPECTOR

STATE of the ART

* Narrow domain of expertise
* Fragile behavior @ boundaries
* Limited Knowledge Representation Language
* Limited I/O
* Limited Explanation
* One EXPERT only
* Knowledge is empiracal in nature

CHALLANGES

How to represent Structure & Function? special purpose languages?

[redacted] to integrate multiple specialised representations?

WHEN is WHICH Representation appropriate? BOTH ACROSS PROBLEMS & WITHIN SINGLE PROBLEM

COMPLEXITY vs TRANSPARENCY & FLEXIBILITY

efficiency tradeoff

CONCLUSION

EXPERT SYSTEMS:

* Practical Application of AI techniques
* Perform @ level of human expert
* Few real success stories
* Generating Tremendous Economic Interest
* Long way to go ...