Intelligent Systems: Reasoning and Recognition

James L. Crowley

ENSIMAG 2 and MoSIG M1 Winter Semester 2010

Lecture 2 5 February 2010

Outline:

Introduction to Expert Systems

Application Domains

Expert systems are a class of software that is useful for domains that are

Subjective,
Poorly formalized, and
require manipulating large numbers of poorly related facts.

Examples include diagnosis, counseling, debugging, game playing, design in complex spaces and problem solving.

Expert system provides an alternative to algorithmic programming.

Expert Systems are typically constructed by hand coding symbolic expressions of the expertise provided by a "domain expert".

Such systems are typically constructed by iterative refinement through the collaboration of a programmer and a domain expert. The programmer "imitates" the system behavior, evoking corrections and advice by the expert. The programmer then encodes these as fact, rules and data structures.

The result is a system that proposes solutions to problems using superficial reasoning.

Such systems may appear to understand but are in fact superficial. They tend to reason without regard to meaning. The system can make non-sense statements if applied outside their domain.

The classic formula for an expert system is :

Program = symbolic knowledge + inference.

The symbolic knowledge encodes poorly structured facts, connected by symbolic (syntactic) reasoning. The classic formula proposed by E. Feigenbaum, is

Expertise = A lot of (symbolic) knowledge + a little bit of (syntactic) reasoning

An expert system combines a domain independent "inference engine" with a database of domain knowledge. The domain knowledge is encoded symbolically as rules and facts. The inference engine provides a form of reasoning.

This view decomposes intelligence into two problems:

1) The design of processes for domain independent reasoning, and
2) The symbolic representation of knowledge.

Programming Techniques for Expert Systems.

We can identify three categories of expert system technologies:
1) Rule based systems
2) Schema based systems
3) Logic programming.

Most expert systems rely on combinations of two of these or all three.

Rules:

Rules have the form:

IF condition THEN action
or IF condition THEN conclusion DO action

Forward chaining rules systems reason by matching conditions to facts to determine actions. Such systems view rules as conditioned reflexes.

Backward chaining rule system reason by matching conclusions to search for conditions that "explain". In such systems, rules are used as representation of causal relations.

In some systems rules can be used for both forward and backward chaining.
In either case, the core problem is "matching" available facts to either the "condition" part of the rule (forward chaining) or the conclusion part (backward chaining).

Examples that we will see:

Forward Chaining :
OPS-5 -> ART -> CLIPS (C-Language Integrated Production Systems).

Backward Chaining:
MYCIN -> E-MYCIN

Schema Systems:

Schema systems provide a form of structured symbolic knowledge. They are often used to "declarative" knowledge. Examples include "Frames", "Scripts" and "Semantic Nets".

Schema systems encode facts as "objects" and are naturally encoded with object-oriented programming. Schema systems enable expression of symbolic hierarchies and allow reasoning by inheritance and message passing.

Examples :

KRL -> KEE -> Nexpert
-> Knowledge Craft (Integrates OPS-5, Prolog and KRL).

Logic Programming:

Symbolic logic is the mathematics for expert systems.

Expressing ideas and natural language with symbolic logic is a powerful tool for analysis and communications.

Logic expressions are often used for communication of declarative knowledge, and to express superficial reasoning.

Logical programming can be implemented with a universal inference rule named Resolution using Horn Clauses.

At one point, logic programming was proposed as a universal tool for informatics. However, logic programming suffers from a problem of algorithmic complexity.

Example : Prolog-1 => Prolog-2 => Prolog-3

The MYCIN Expert System

In the 1970's, at Stanford University, Edward Feigenbaum directed a research group named the "Heuristic Programming Project". Their central thesis was

> Intelligence = Large quantity of domain knowledge and a little bit of reasoning.

This led to an investigation into Knowledge Representation Techniques.

From 1970 to 1973 the sought to build a system that could interpret data from Mass Spectrograms.

A Mass Spectrograph is a device that uses electrical or magnetic fields to determine the masses of atoms or molecules in a sample. A beam of ions is passed through the electrical or magnetic field. The field deflects the ions at different angles depending on their masses, thereby breaking the beam into separate, identifiable bands.

The result of their project was a system named DENDRAL.
DENDRAL was an un-maintainable "hack". However, by 1973 the group had learned to express declarative knowledge as "rules". It was decided to start over, building a "rule based" system for "anti-biotic Therapy".

Penicillin was discovered in 1929 and came into widespread use as an antibiotic in the 1940's. During the 1950's and 1960's a variety of new antibiotics were discovered. Each had unique properties and uses. By the 1970s, most medical doctors required consultation with a specialized expert to prescribe antibiotics.

The Stanford University Medical School was a world famous center for research in antibiotics. The Medical School asked the Computer Science School for help. Feigenbaum proposed to construct an "Artificial Expert" antibiotic therapy advisor.

The system developed from 1973 to 1978. It evolved into the first true "Expert System". As such it became the model for a new class of systems.

MYCIN: An Antibiotics Therapy Advisor.

By 1975, a large variety of Antibiotics were available. Each antibiotic was effective against a specific set of microbes, and triggered a specific set of side effects.

Patients were often allergic to certain families of antibiotics.
MYCIN was designed to be used by ordinary doctors who lacked the specialized training required to develop anti-biotic therapies.

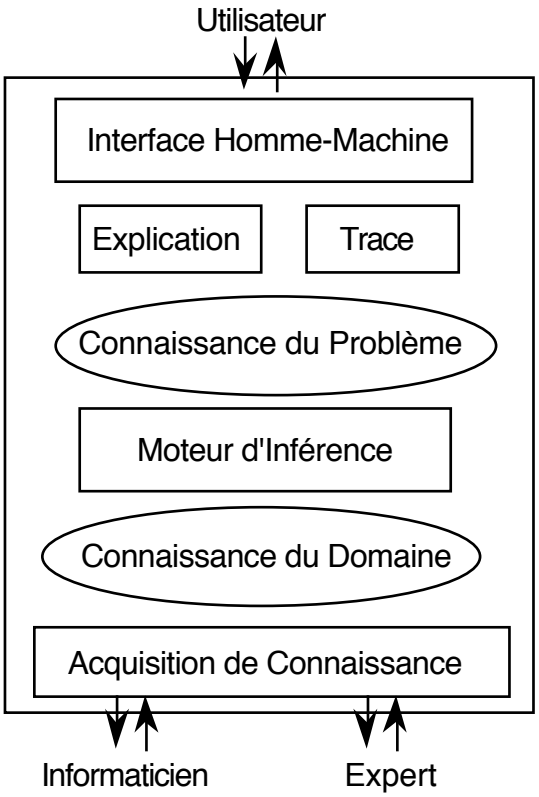
Specifications: The system was required to be:

- Easy to use
- Reliable
- Able to manipulate large numbers of unrelated facts.
- Able to use inexact and incomplete facts

Able to explain its advice.

MYCIN was composed of approximately 500 rules, manipulating a large base of structured facts. The rules provided procedural knowledge to

1) Request or infer the required information
2) Apply specialized knowledge to determine a therapy
3) Provide advice to doctors
4) Respond to questions about its reasoning.


Utilisateur
Interface Homme-Machine
Explication
Trace
Connaissance du Problème
Moteur d'Inférence
Connaissance du Domaine
Acquisition de Connaissance
Informaticien
Expert

Facts

All facts in MYCIN are represented by a "quadruple":

(CONTEXT, PARAMETER, VALUE, CF)

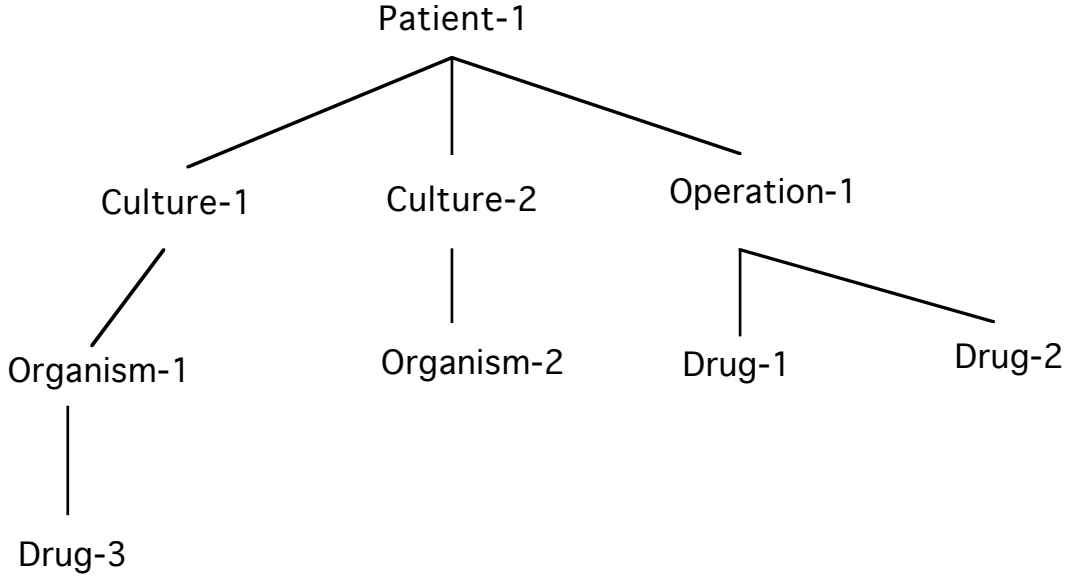
CONTEXTs structure reasoning and thus provide control.
For the MYCIN antibiotic therapy advisor, 10 contexts were required.

PERSON: Data about the patient
OPERS: Past Medical procedures
CURCULS: Medical cultures taken from the patient
CURDRUGS: Current drug therapies for the patient
CURORGS: Known microbial infections in the patient
OPDRUGS: Drugs used during recent medical procedures
PRIORCULS: Past medical cultures
PRIORDRUGS: Past medical therapies
PRIORORGS: Past infections.

Contexts are organized in a tree.

MYCIN Context Tree was structured to respond to 4 questions.
1) What symptoms does the patient show
2) What microbes infect the patient
3) What antibiotics are effective against the microbes
4) What is the most appropriate antibiotic.

The context tree was called the "Dynamic Tree", and was composed of instances of each context.



The context tree served to focus reasoning.

PARAMETERS

Each context was composed of a number of parameters. Each parameter was described by a data structure.

Attributes of Parameters included:

Expect : {Y/N, NUMB, ONE_OF, ANY_OF}
PROMPT : A sentence to ask for the value of the parameter.
LABDATA (Y/N) : Whether the parameter should be requested from the doctor or inferred automatically by the system.
(LABDATA was later renamed "ASK-FIRST")
LOOKAHEAD: A list of rules for inferring the value of a Parameter.
TRANS: An English language explanation of the parameter and the meaning of its values.

Extensive pre-coding of English sentences allowed to system to appear capable of intelligent dialog. The MYCIN system could almost pass the Turing Test!

PARAMETER Categories:
SingleValued: Parameter could take a single value. If multiple values are provided the most likely must be determined.
MultiValued: The parameter could have multiple values. A fact was created for each value.
Binary: A Boolean value; A single value that can be Yes or No

Reasoning between alternative single valued parameters required some for me evidential reasoning. For this, the MYCIN team invented a "Confidence Factor".

The MYCIN Confidence Factor

ALL facts in MYCIN are labeled with a confidence Factor CF $\in$ [-1, 1]
ALL rules in MYCIN are labeled with a FORCE: CF $\in$ [-1, 1].

RULES

MYCIN reasons with goal directed backward chaining rules.

Rules in MYCIN implement a form of "ABDUCTION".

$$A \wedge B \xrightarrow{CF_R} C$$

At any instant, MYCIN is trying to PROVE a GOAL (C).
The goal is the conclusion part of a rule. For this MYCIN attempts to prove the condition part of the rule, A ∧ B

$A \wedge B \xrightarrow{CF_R} C$ = To prove C, try to prove A and B.

Allowed Rule Structures:

A ∧ B ∧ C -> D
A ∧ (B ∨ C) -> D
(A ∨ B ∨C) ∧ (D ∨E) -> F

Disjunctions of conjunctions are not formally required. However, doctors to explain reasoning often use them.

Not allowed
A ∨ B ∨ C -> D
A ∧ (B ∨(C ∧ D)) -> E

MYCIN contains templates that allow it to interpret the chain of reasoning as English language sentences

Evidential Reasoning and combining Hypotheses :

Given two hypotheses H_1 and H_2

$H_1 \wedge H_2 \rightarrow H_3$

Conjunction: The confidence for $H_1 \wedge H_2$ is :

$CF(H_1 \wedge H_2) = \min\{ CF(H_1), CF(H_2) \}$

Disjunction : Confidence for H_1 or H_2

$CF(H_1 \vee H_2) = \max \{ CF(H_1), CF(H_2) \}$

Rules apply a force "CF_R"

$A \wedge B \xrightarrow{CF_R} C \quad CF_C = CF_R * \min \{ CF_A , CF_B \}$

$A \wedge (B \vee C) \xrightarrow{CF_R} D \quad CF_D = CF_R * \min \{ CF_A , \max\{CF_B, CF_C\}\}$

An important case is when two independent paths reach the same hypothesis:

The MYCIN team invented a rule called COMBINE

Combine(CF_1,CF_2) =

$CF_1 + CF_2 (1 - CF_1)$ Si CF1 ≥ 0 et Si CF2 ≥ 0

$\dfrac{CF_1 + CF_2}{1 - \min\{ |CF_1| , |CF_2| \}}$ Si $CF_1 \bullet CF_2 < 0$

$-Combine(-CF_1, -CF_2)$ Si $CF_1 \leq 0$ et Si $CF_2 \leq 0$

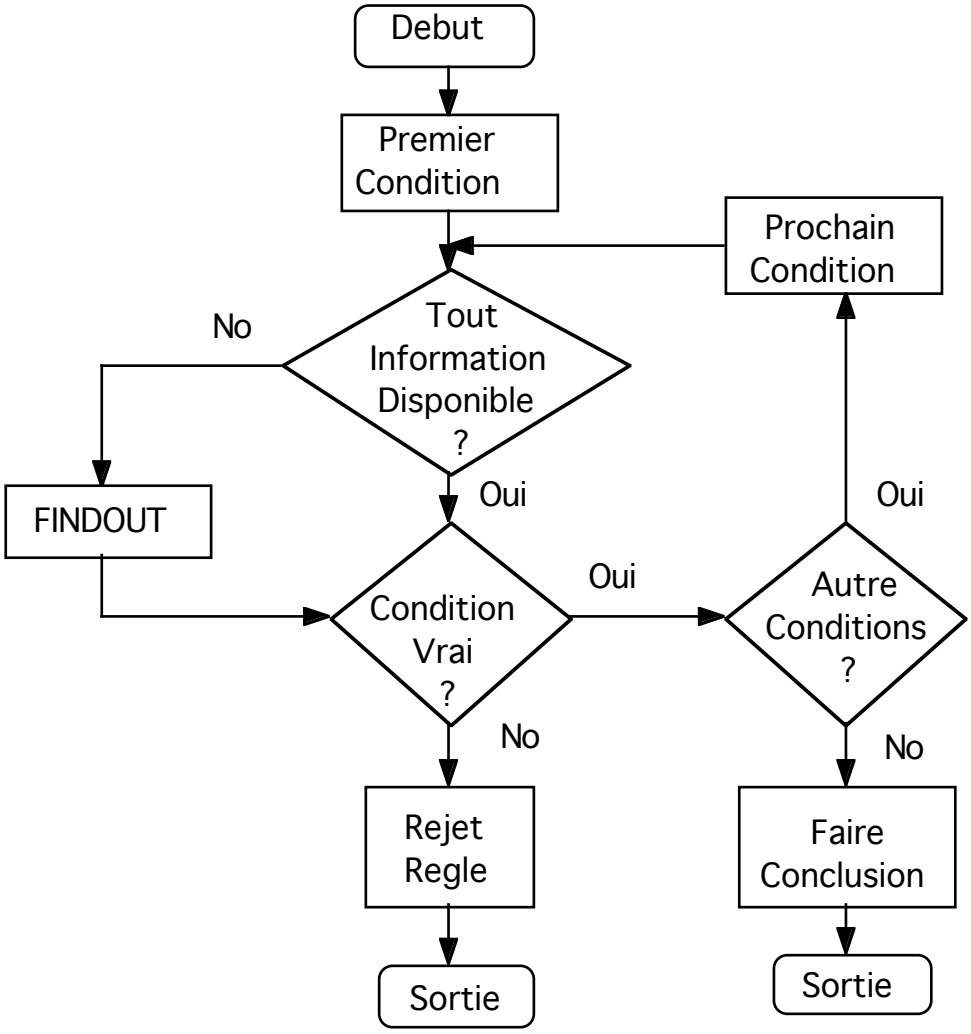
Alors :
(H_1, CF_1) et (H_2 , CF_2) et $(H_1 = H_2)$ -> $(H_1, CF_1 = Combine (CF_1, CF_2))$

En MYCIN $CF(\neg H) = - CF(H)$

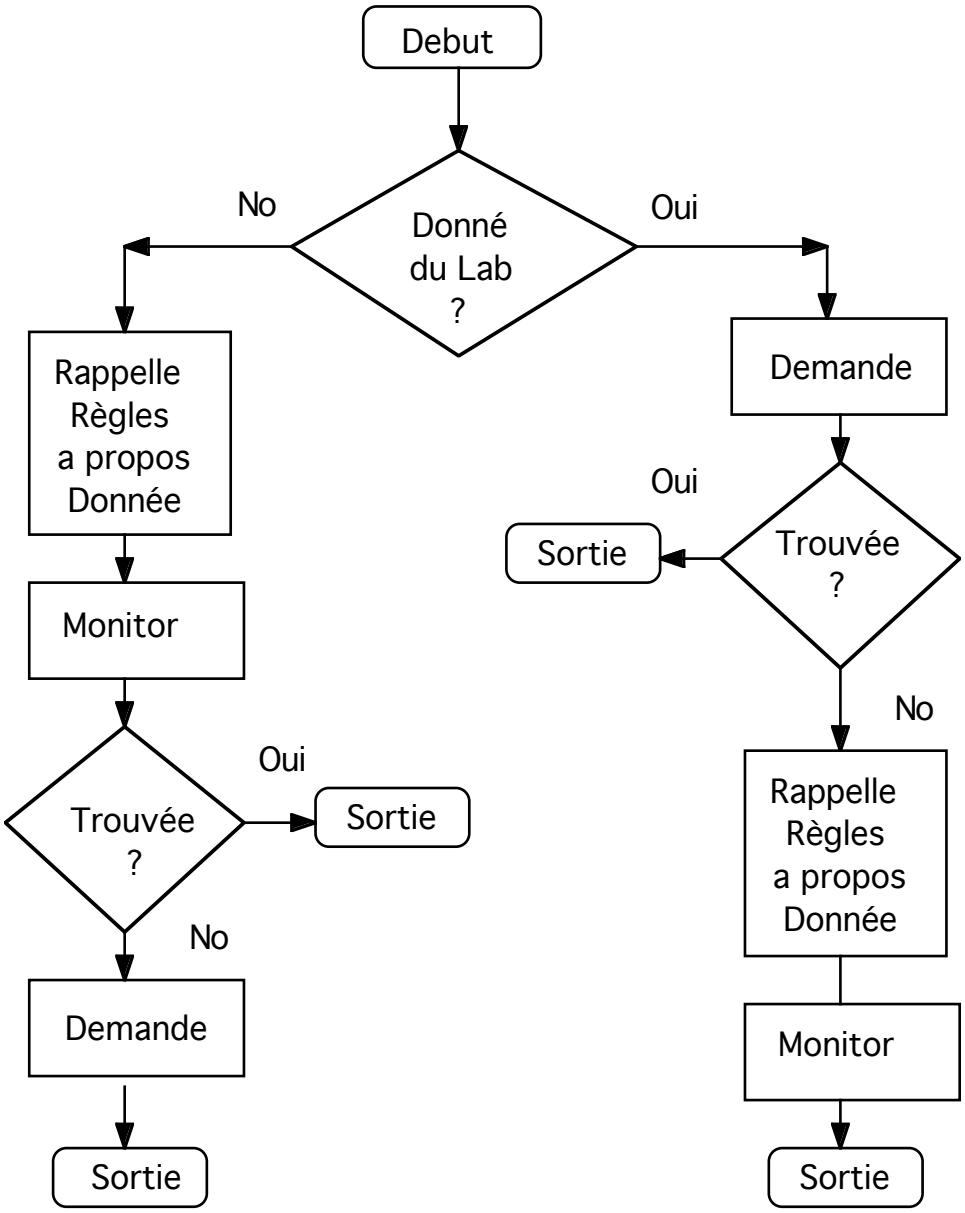
Control

Two interleaved procedures are used:

MONITOR :



FINDOUT:


Debut
Donné du Lab ?
No
Oui
Rappelle Règles a propos Donnée
Monitor
Trouvée ?
Oui
Sortie
No
Demande
Sortie
Demande
Oui
Trouvée ?
Sortie
No
Rappelle Règles a propos Donnée
Monitor
Sortie

Interactions :

At any instant the user may ask

WHY? The system provide an interpretation of the trace of reasoning
HOW: The system provides the source for a fact.

Coupled with the extensive use of preprogrammed sentences, this made the system appear to be intelligent. However the knowledge was shallow.