Introduction to Artificial Intelligence

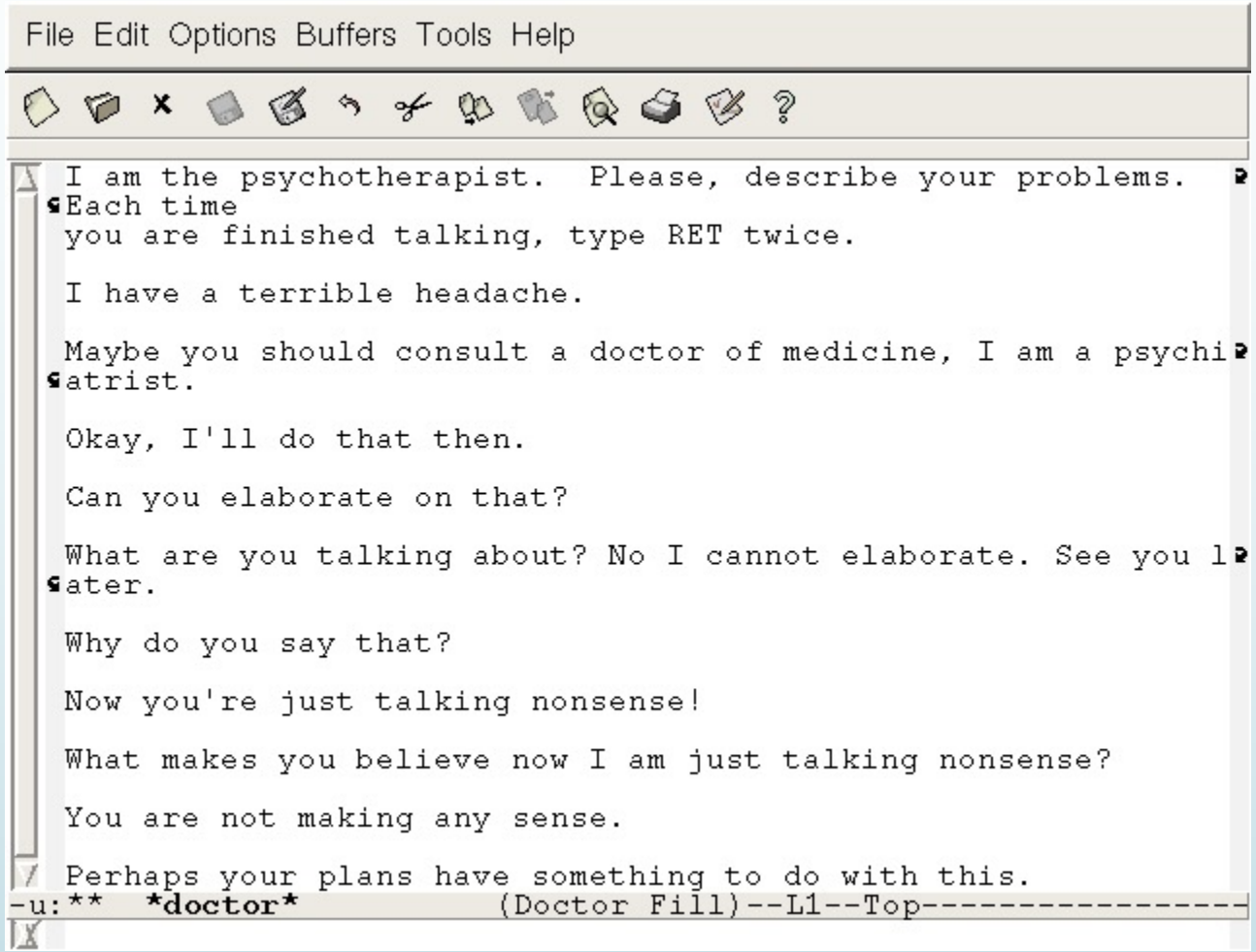
Week 8

Expert Systems

ELIZA

- Developed at MIT (1964-66) by Joseph Weizenbaum
- Idea was for it to act as a parity showing the Chinese Room
- Lead to a number of text adventure computer games
  - GO LEFT
    - "you go left and see a gnoll with an axe, what do you do?"
- Limited success at the Turing test

ELIZA in action on DOCTOR Script



How does it work?

- “I feel like this is an English sentence”
  - First the string is disassembled based on some rules finding keywords
- <I> <feel> <this is an English sentence>
  - Next it is matched to a list of constructions
- <I> <feel> <this is an English sentence> = <I> <FEEL> <X>
  - This produced a construction rule
- <I> <FEEL> <X> ?:- <Why do you feel><X><?>
  - This rule is applied and response is made
- Why do you feel like this is an English sentence?
  - If no rule is found – delay
- Go on….
  Tell me more about <X>

Logic Programming

- Roots back to 1930s and Alonzo Church with Lambda Calculus
- Developed in 1950/60s
- Is a fundamentally differing paradigm than imperative languages (e.g. C, C++, C#, Java, Python, etc.)
  - Still Turing Complete
  - Examples include LISP and PROLOG

Statements - Lambda Calculus

Syntax	Name	Description
a	Variable	A character or string representing a parameter or mathematical/logical value
(λx.M)	Abstraction	Function definition (M is a lambda term). The variable x becomes bound in the expression
(M N)	Application	Applying a function to an argument. M and N are lambda terms

Operations - Lambda Calculus

Operation	Name	Description
$((\lambda x.M)\ E) \rightarrow (M[x:=E])$	β-reduction	Substituting the bound variable by the argument expression in the body of the abstraction
$(\lambda x.M[x]) \rightarrow (\lambda y.M[y])$	a-conversion	Renaming the bound (formal) variables in the expression. Used to avoid name collisions

Imperative v. Logic Programming

Imperative

- Statements
- Code and Data
- Code by programmer
- Data as input from programmer/user
- Actions from User (sometimes)
- Program build from libraries but many of the units are special purpose

Logic

- Rules
- Inference Eng./Knowledge Base
- Inference Engine built in
- Knowledge base by programmer/user
- Queries from User
- Rapid deployment and prototyping as the language has the inference engine by default

AI Winter

- Again these systems while being very popular in the 1980s suffered under the AI winter
- The idea of the these systems was that giving a computer all of the knowledge will make it “smart”
- Legal issues
  - What if a doctor makes a misdiagnosis

Areas of Application for Expert Systems (Hayes-Roth 1983)

Category	Problem addressed	Examples
Interpretation	Inferring situation descriptions from sensor data	Hearsay (speech recognition), PROSPECTOR
Prediction	Inferring likely consequences of given situations	Preterm Birth Risk Assessment
Diagnosis	Inferring system malfunctions from observables	CADUCEUS, MYCIN, PUFF, Mistral,[Eydenet, Kaleidos
Design	Configuring objects under constraints	Dendral, Mortgage Loan Advisor, R1 (DEC VAX Configuration), SID (DEC VAX 9000 CPU)
Planning	Designing actions	Mission Planning for Autonomous Underwater Vehicle
Monitoring	Comparing observations to plan vulnerabilities	REACTOR[41]
Debugging	Providing incremental solutions for complex problems	SAINT, MATHLAB, MACSYMA
Repair	Executing a plan to administer a prescribed remedy	Toxic Spill Crisis Management
Instruction	Diagnosing, assessing, and repairing student behavior	SMH.PAL,[Intelligent Clinical Training, STEAMER
Control	Interpreting, predicting, repairing, and monitoring system behaviors	Real Time Process Control, Space Shuttle Mission Control

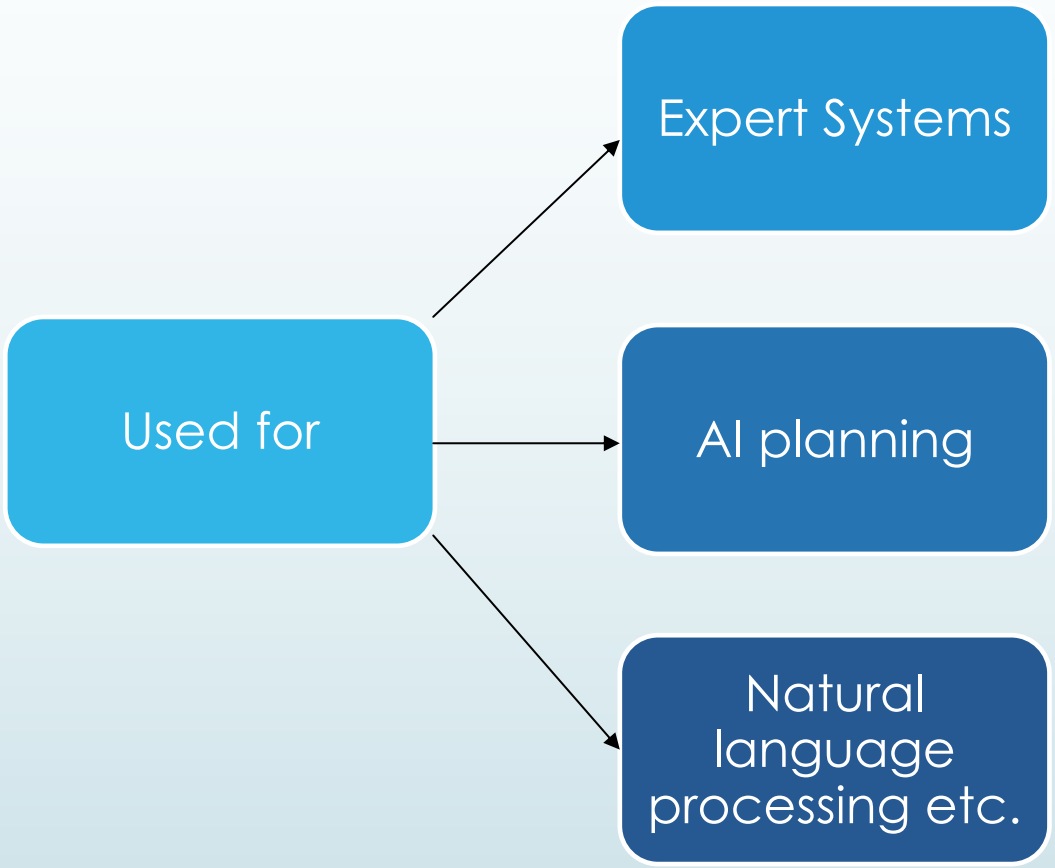
Training an Expert System

- Interviews with experts
  - Experts classify knowledge cases
  - Provide a series of rules which they use to make the classifications
  - Very costly - knowledge acquisition problem
    - Experts by definition are rare
      - Meaning expensive
      - Why would they want to train the competition?
- Machine Learning Techniques – especially Decision trees are used to define a pathway to a solution
- Front end development and user understanding

Prolog

- **Prolog** is a general-purpose logic programming language associated with artificial intelligence and computational linguistics
- Prolog stands for **Pro**gramming in **Log**ic
- Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is declarative: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a *query* over these relations
- The language was first conceived by a group around Alain Colmerauer in Marseille, France, in the early 1970s and the first Prolog system was developed in 1972 by Colmerauer with Philippe Roussel

Prolog Applications



Why to Study?

- Prolog was one of the first logic programming languages and remains the most popular such language today, with several free and commercial implementations available
- Currently, not widespread, and not the fastest in terms of execution
- Very easy to understand and apply
- Someday its concept can be useful for programming like it happened to functional programming languages

Prolog Semantics

- Head :- Body.
- Clauses with empty bodies are called facts

cat(tom).

- Clauses with bodies are called rules. Rules describe new rules based on facts and/or other rules

animal(X) :- cat(X).

- Queries are needed to show outputs. They are also called goals

?- cat(X).

Answer: X = tom

Rule Example

- Facts and Rules together comprises a knowledge base of the program. Rules consists of facts with added conditions
- Example
  - father(Y,Z) :- man(Y), son(Z,Y).
    - If the body of the rule is true then the head is true
    - ":-" is read as "if"
    - "," is logical AND
    - ";" is logical OR
    - The statement is read as: Y is a father of Z if Y is a man and Z is a son of Y.

Atoms and Variables

- **Atoms** – upper case letters, lower case letters, digits (0, 9), characters (+-*/<>=:~)
- **Variables** – consists of letters, digit and underscore. Variables start with upper-case letter or underscore. They can be free and bound. Free is the one, which was not assigned the value, bound is opposite
  - ?- 5 = 4 // false
  - ?- X = 5, X = 4 // false
  - ?- X = 5 + 4 // it is not a mathematical expression!
  - ?- X is 5 + 4 // X = 9
  - ?- X = Y, X = 5, Z is Y + 4 // X = 5, Y = 5, Z = 9

Example: Family Tree (1/5)

- Let's consider family tree. First step is to build family tree including sisters, brothers, mother, father, aunts, uncles, grandfathers and grandmothers. It should be possible to check all these abovementioned relations
- Create **facts** with

male(nick).

- Create **rules** for these people like

mother(X,Y) :- parent(X,Y), female(X).

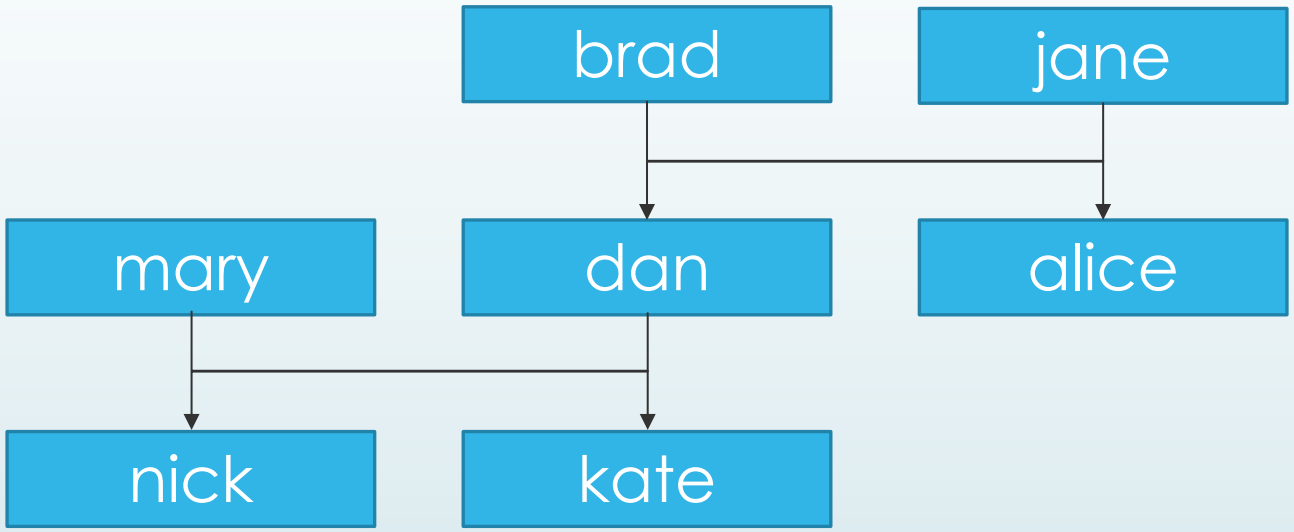
where *X* is mother, *Y* is a child.

- Check **queries** with

?- mother(jane,nick).

- If it is correct, then the output should be ***true.*** Otherwise, ***false***

Example: Family Tree (2/5)



Example: Family Tree (3/5)

Facts:

male(nick).
male(dan).
male(brad).
female(jane).
female(alice).
female(kate).
female(mary).

parent(dan,nick).
parent(dan,kate).
parent(jane,dan).
parent(brad,dan).
parent(jane,alice).
parent(brad,alice).
parent(mary,nick).
parent(mary,kate).

Example: Family Tree (4/5)

Rules:

father(X,Y) :- parent(X,Y), male(X).

mother(X,Y) :- parent(X,Y), female(X).

grandfather(X,Y) :- father(X,Z), parent(Z,Y).

grandmother(X,Y) :- mother(X,Z), parent(Z,Y).

brother(X,Y) :- parent(Z,X), parent(Z,Y), male(X), X\=Y.

sister(X,Y) :- parent(Z,X), parent(Z,Y), female(X), X\=Y.

aunt(X,Y) :- parent(Z,Y), sister(X,Z).

uncle(X,Y) :- parent(Z,Y), brother(X,Z).

Example: Family Tree (5/5)

- **Goal:** *?- mother(jane,nick).*
  Answer: false
- **Goal:** *?- father(dan,nick).*
  Answer: true
- **Goal:** *?- parent(jane,X).*
  Answer: X = dan ;
  X = alice
- **Goal:** *?- parent(jane,X), !.*
  Answer: X = dan
- **Goal:** *?- aunt(X,Y).*
  Answer: X = alice, Y = nick ;
  X = alice, Y = kate
- **Goal:** *?- aunt(X,_).*
  Answer: X = alice