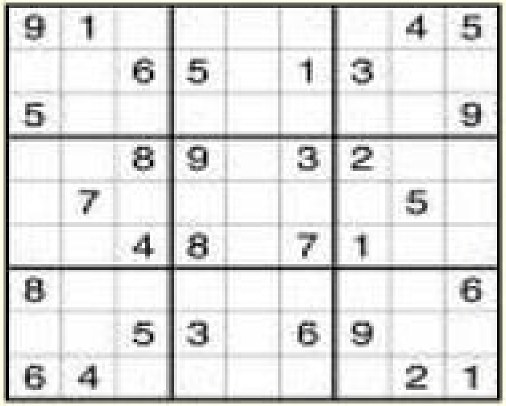
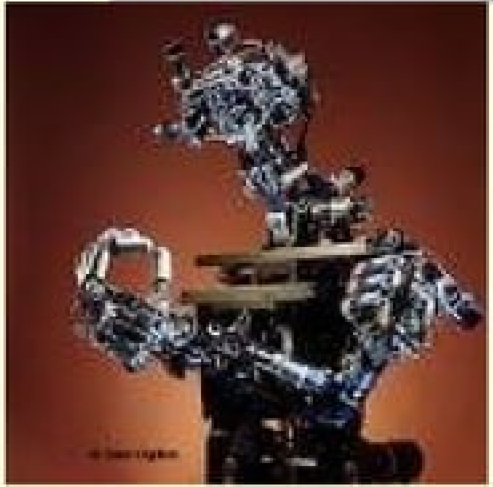


9	1						4	5
		6	5		1	3		
5								9
		8	9		3	2		
	7						5	
		4	8		7	1		
8								6
		5	3		6	9		
6	4						2	1



Artificial Intelligence (Part 3) AI PROGRAMMING LANGUAGE: PROLOG

Course Contents

Again..Selected topics for our course. Covering all of AI is impossible!

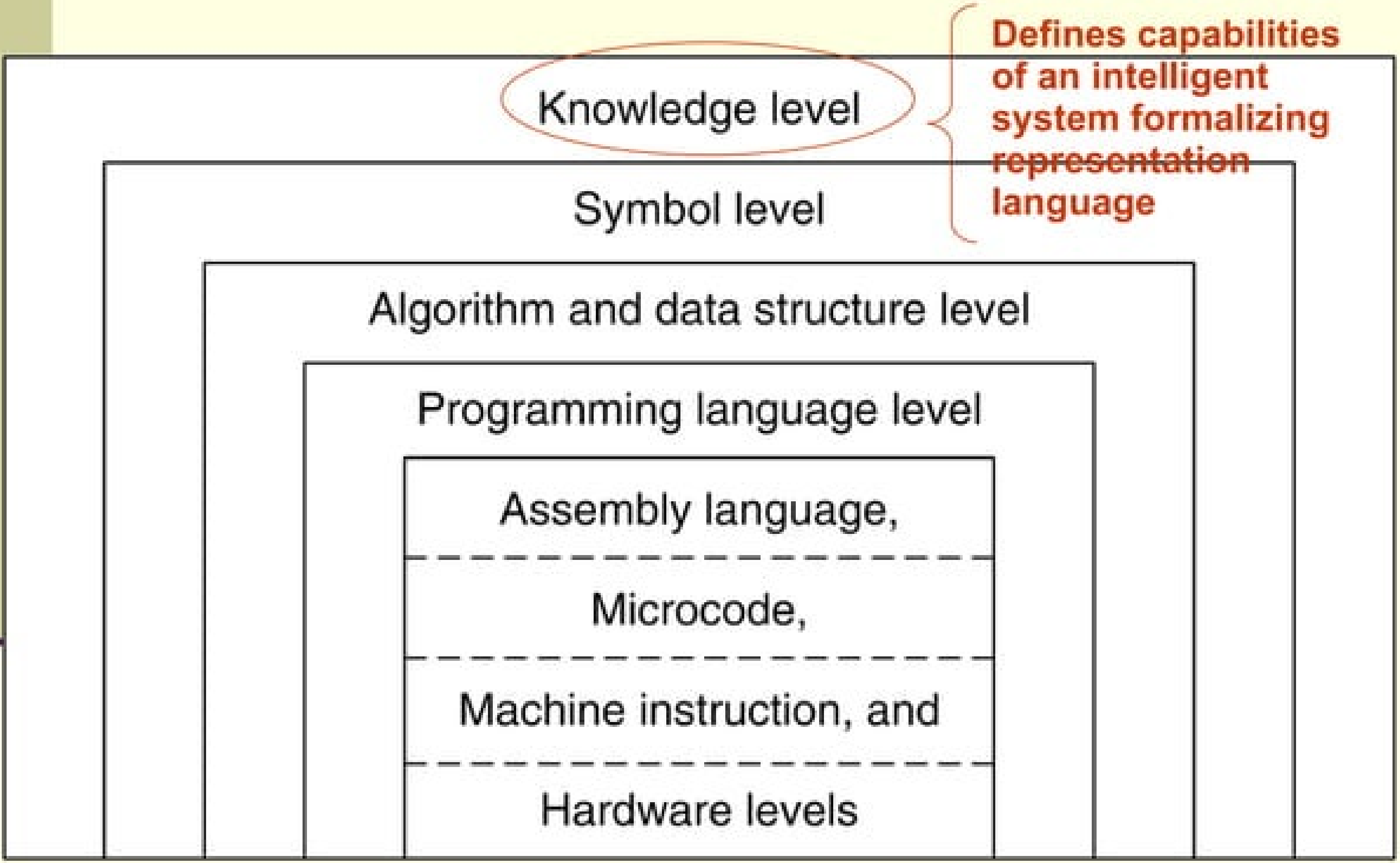
Key topics include:

- Introduction to Artificial Intelligence (AI)
- Knowledge Representation and Search
- Introduction to AI Programming
- Problem Solving Using Search
- Exhaustive Search Algorithm
- Heuristic Search
- Techniques and Mechanisms of Search Algorithm
- Knowledge Representation Issues and Concepts
- Strong Method Problem Solving
- Reasoning in Uncertain Situations
- Soft Computing and Machine Learning

PROLOG

- LISP and PROLOG are most frequently used languages in AI
- Syntax and semantic features encourage powerful way of thinking about problems and solutions
- Tools for thinking

LEVELS OF KNOWLEDGE-BASED SYSTEM



Defines capabilities of an intelligent system formalizing representation language

Intro to PROLOG

- Best-known example for LOGic PROgramming Language
- Uses first-order predicate calculus to express specification
- Elegant syntax and well-defined semantics
- Based on theorem proving by J.A.Robinson 1965. He designed proof procedure called resolution

Syntax for predicate calculus programming

- To represent facts and rules

English	Pred calculus	Prolog
and	∧	,
Or	∨	;
Only if	←	:-
not	¬	not

Facts, Rules, and Queries

- **A knowledge base of facts** -are terms which are followed by a full stop.
  - parent(ayah,saya). %ayah is my parent
  - parent(mak,saya).
  - female(mak). %mak is a female
  - male(ayah).

- **Rules -create new knowledge**
  - mother(X,Y) :-
    parent(X,Y) , female(X). %X is mother of Y if X is
    %parent of Y and X is female

- **Queries-** are also complex terms which are followed by a full stop.
  - ?- parent(X,saya). %who is my parent

Prolog command..facts

- Open Swi-Prolog window
- File-New-Type the facts and rules with full stops at the end
- Add facts to database

```
 parent(ayah,saya).
  parent(mak,saya).
  female(mak).
  male(ayah).
```

- Save- close file -backtoSwiprolog-File-Consult-choose file-open <enter>

Prolog command..rule

- **To add more facts and rules, eg.**
  - **File- Edit –Choose File – type new rule to indicate relation mother**

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

- **Save-Close file**
- **Consult, then write a Query to:**
  - **List who is my mother**
  - **See if ayah is my mother?**
  - **List how many mothers in the database?**

Prolog command..queries..

- Type at the command line for query. Use symbol ; to list next parent

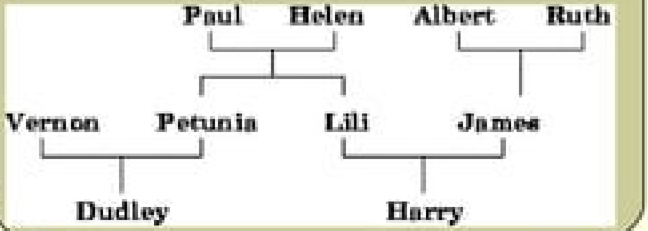
```
?- parent(X,saya).

?- female(ayah).

?- male(X).
```

Exercise (**Family relationships)**

1) Use the predicates male/1, female/1, and parent_of/2 to represent your family tree as a Prolog knowledge base



2) Now, formulate rules to capture the following relationships:

```
father(Father,Child)
mother(Mother,Child)
grandparent(Grandparent,Child)
sister(Sister,Person)
grandchild(Grandchild,Child)
```

Ex:

```
grandparent(X, Z)      :-  parent(X, Y),  parent(Y, Z).
father_of(X, Y)        :-  male(X),       parent(X, Y).
```

Exercise (**Family relationships)**

3) Test your knowledge base with these queries:

Do you have an aunt?
Who are your grandparents?
Who are the grandchildren of your grandparents?
Do you have a sister?

Recursion in Prolog

- Recursion is the primary control mechanism for prolog programming
- In Prolog, a list is either an empty list or a term connected by '.' to another list
- Someone's ancestor can be one of their parents or an ancestor of one of their parents
- Find an ancestor

```
ancestor( Old, Young ) :- parent( Old, Young ).
ancestor( Old, Young ) :- parent( Old, Middle )
, ancestor( Middle, Young ).
```

Recursion in Prolog

- When we want to write recursive programs, we need to think about two things:
  - 1. How will the program terminate?
  - 2. How will the program break up the data it works on?

- Recursion is an example of a *divide-and-conquer* Strategy
- Note that we normally put the base case first, so that Prolog tests it first!
- To ensure that a program terminates, we must have at least one *base case* – a non-recursive clause
- We must also ensure that something gets (in some sense) "reduced" each time a recursive step happens, so that we can say when we have got to the end

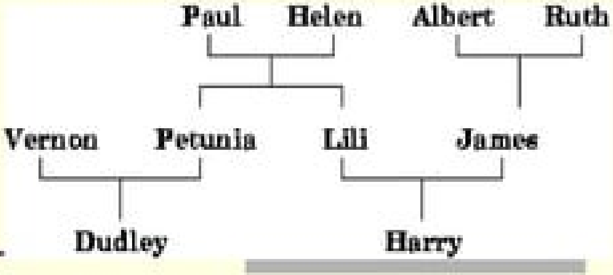
- Example – testing if a term is a list:
- – The base case is when we have an empty list –the smallest list possible
- – The recursive case breaks down a non-empty list into a head and a tail and then tests the tail, so the thing being tested gets smaller each time.

Base case

```
ancestor( Old, Young ) :- parent( Old, Young ).
ancestor( Old, Young ) :- parent( Old, Middle ),
ancestor( Middle, Young ).
```

Recursive-clause

Recursion in Prolog



- Example run:

```
?- ancestor( paul, harry ).
Call: ancestor(paul, harry ).
Call: parent(paul, harry ).
Fail.
Retry: ancestor(paul, harry ).
Call: parent(paul, Middle ).
Unify: Middle = lili.
Succeed: parent(paul, lili ).
Call: ancestor( lili, harry ).
Call: parent( lili, harry).
Succeed: parent( lili, harry ).
Succeed: ancestor( lili, harry ).
Succeed: ancestor(paul, harry)
```

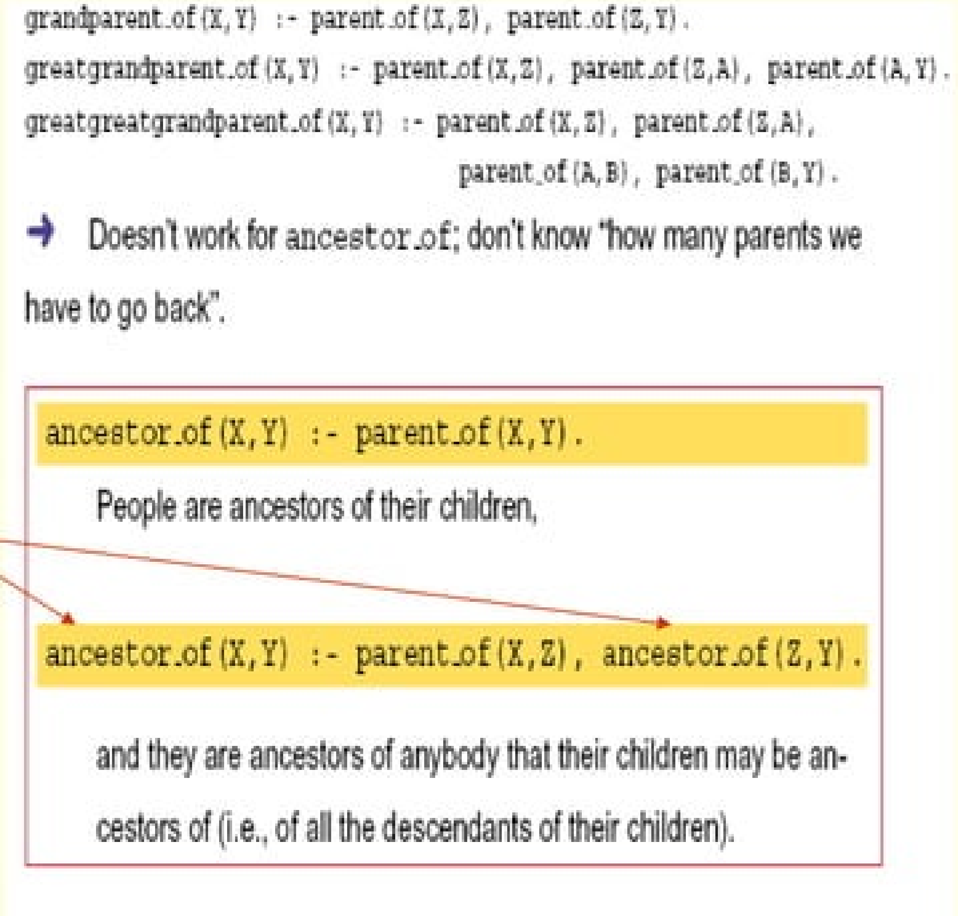
```
ancestor( Old, Young ):- parent( Old,Young ).
ancestor( Old, Young ):- parent( Old,Middle ),
ancestor( Middle, Young ).
```

recursive predicate definitions

http://www.coli.uni-saarland.de/~kris/esslli04prolog/slides/0.day2.pdf

Task: Define a predicate ancestor of (X,Y) which is true if X is an ancestor of Y.

RECURSIVE



```
grandparent_of(X,Y)  :- parent_of(X,Z), parent_of(Z,Y).
greatgrandparent_of(X,Y)  :- parent_of(X,Z), parent_of(Z,A), parent_of(A,Y).
greatgreatgrandparent_of(X,Y)  :- parent_of(X,Z), parent_of(Z,A),
                                  parent_of(A,B), parent_of(B,Y).
```

➜ Doesn't work for `ancestor_of`; don't know "how many parents we have to go back".

```
ancestor_of(X,Y)  :- parent_of(X,Y).
```

People are ancestors of their children,

```
ancestor_of(X,Y)  :- parent_of(X,Z), ancestor_of(Z,Y).
```

and they are ancestors of anybody that their children may be ancestors of (i.e., of all the descendants of their children).

Exercise in Prolog

```
ancestor( X, Y):- parent( X,Y ).
ancestor( X, Z ):- parent( X,Z ), ancestor( Z,Y ).
```

- Exercise the recursive predicate ancestor using your family tree, add the rule above, then make queries:

```
?- ancestor (saya,X).
?- ancestor (X,saya).
?- ancestor (X, mak).
```

- USE TRACE FACILITY TO DISPLAY RECURSIVE

EXERCISE on RECURSION- KNIGHT'S LEGAL MOVE

- move(1,8).
- move(2,7).
- move(2,9).
- move(3,8).
- move(3,4).
- move(4,3).
- move(4,9).
- move(7,6).

move(7,2).
move(6,7).
move(6,1).
move(1,6).
move(8,3).
move(8,1).
move(9,4).
move(9,2).

-TERMINATE RECURSIVE IF X IS IN X POSITION

-AVOID DUPLICATE STATES

```
member(X,[X|T]).
member(X,[Y|T]) :- member(X,T).

path(Z,Z,L).
path(X,Y,L) :- move(X,Z),not(member(Z,L)),path(Z,Y,[Z|L]).
```