Introduction to Prolog and logic programming

to read:

Chapter 10, Russel & Norvig

What will be discussed

- Inference machines
  - *Logic programming* (Prolog) and theorem provers
- Next
  - Production systems
  - Semantic networks and Frames

Fundamental Considerations

Necessary to implement efficiently

- Complex inferences
- STORE and FETCH

Store

- KB = some form of knowledge base in which intermediate and permanent knowledge is stored
- TELL
  - TELL(KB, A ∧ ¬ B), TELL(KB, ¬C ∧ D)
  - [A, ¬ B, ¬C, D]
- Array of list of conjuncts
  - Costs: check: O(1)
    - check: O(n)

Data-structures

- List of array is inefficient O(n).
- Hash table (O(1)) : P of $\neg$P
  - but not $P \wedge Q \Rightarrow R$
  - Brother(John, x)

Key	
P	F
Q	T
R	F

- cannot solve: Brother(John, Richard) for query Exist x Brother(John, x).

Table indexing

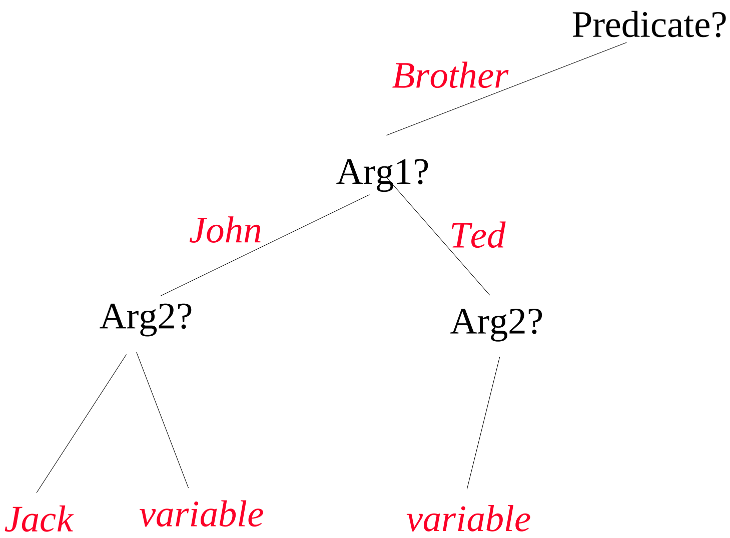
- implicative normal form.
- predicate symbol.
- Other:
  - positive literals
  - negative literals

ASK(KB, Brother(Jack, Ted))

```
Brother(Richard,John)
Brother(Ted,Jack)∧Brother(Jack,Bobbie)
¬Brother(Ann,Sam)
Brother(x,y)⇒Male(x)
Brother(x,y) ∧ Male(y)⇒Brother(y,x)
Male(Jack)∧Male(Ted) ∧ … ∧ ¬Male(Ann) ∧ …
```

Key	Positive	Negative	Conclusion	Premisse
Brother	Brother(Richard, John) Brother(Ted,Jack Brother(Jack, Bobbie)	¬Brother(Ann,Sam)	Brother(x,y) ∧ Male(y)⇒ Brother(y,x)	Brother(x,y) ∧ Male(y) ⇒ Brother(y,x) Brother(x,y)⇒ Male(x)
Male	Male(Jack) Male(Ted)	¬Male(Ann)	Brother(x,y) ⇒ Male(x)	Brother(x,y) ∧ Male(y) ⇒ Brother(y,x)

Tree-based indexing



Brother(John,Jack)
Brother(John, x)
Brother(Ted, x)

Program length

Language	Length in pages
Fortran	36
Cobol	25
Ada	24
PL/I	22
C	22
Pascal	20
Basic	19
MProlog	9

program

Programming paradigms

1. Imperative Programming

It is described, **how** the problem should be solved

- Assembler
- ADA
- BASIC
- C / C++
- COBOL
- FORTRAN
- Java
- Modula
- PASCAL
- Perl
- PL/1
- Simula
- Smalltalk
- und viele mehr...

2. Functional Programming

Program is a set of Functions.

- Lisp
- Logo
- Haskell
- ML
- Hope
- Scheme
- Concurrent Clean
- Erlang
- NESL
- Sisal
- Miranda

Example of Programs in functional Languages

```lisp
(print "Hello World")
```

```lisp
(let ( (a 0) )
  (while (< a 20)
    (princ a) (princ " ")
    (setq a (+ a 1))
  )
)
```

Lisp

```
fun iter (a,b) =
    if a <= b then (print(Int.toString(a*a)^" "); iter(a+1,b) )
    else print "\n";

iter(1,10);
```

```
fun iter 0 = ""
|   iter n = (iter(n-1); print(Int.toString(n*n)^" "); "\n");

print(iter 10);
```

ML

Object-oriented Programming Languages

- ***Object-oriented programming*** languages provide an alternative approach
- The problem is broken into modules called ***objects***
  - consist of both the data and the instructions which can be performed on that data

Object-oriented Programming Languages

- **Smalltalk** was the earliest such language
  - designed from start to be OOP based
- More recent OOP languages include
  - **C++** - the C language with OOP extensions
  - **Java** - designed for use with the Internet
    - Java is likely to become increasingly common
      - as you can use it to build application programs into your web pages

Object-oriented principle…..

It is being developed within all three paradigms below:

Imperative:

- Smalltalk
- Java
- Eiffel
- C++
- Object Pascal

Functional:

- XLISP
- Haskell
- others...

Declarative:

- various versions of Prolog
- others...

All new programming languages are object - oriented

4th Generation Languages

- **4GLs** intended as design tools for particular *application domains*
  - meant to be used by non-programmers
  - FIND ALL RECORDS WITH NAME "SMITH"
- One class of 4GL is ***program generators***
  - generate a program based on specification of problem to be solved

4th Generation Languages

- 4GLs often found as part of **database** packages
  - Oracle, Sybase, etc.
  - Use language called SQL
- Also called ***nonprocedural*** languages
  - problem is defined in terms of desired results rather than program procedures

Declarative Programming Languages

- Some languages take still other approaches to programming
  - Mainly used in special purpose areas
  - Artificial Intelligence, Expert Systems
- Often called ***declarative*** or ***rule-based*** languages

Declarative Programming Languages

- Examples include
  - Lisp (List Processing)
  - Snobol (String Processing)
  - Prolog (Programming in Logic)
- **Prolog** deals with ***objects*** and ***relationships***
  - declares **facts**
  - defines **rules**
  - asks **questions**

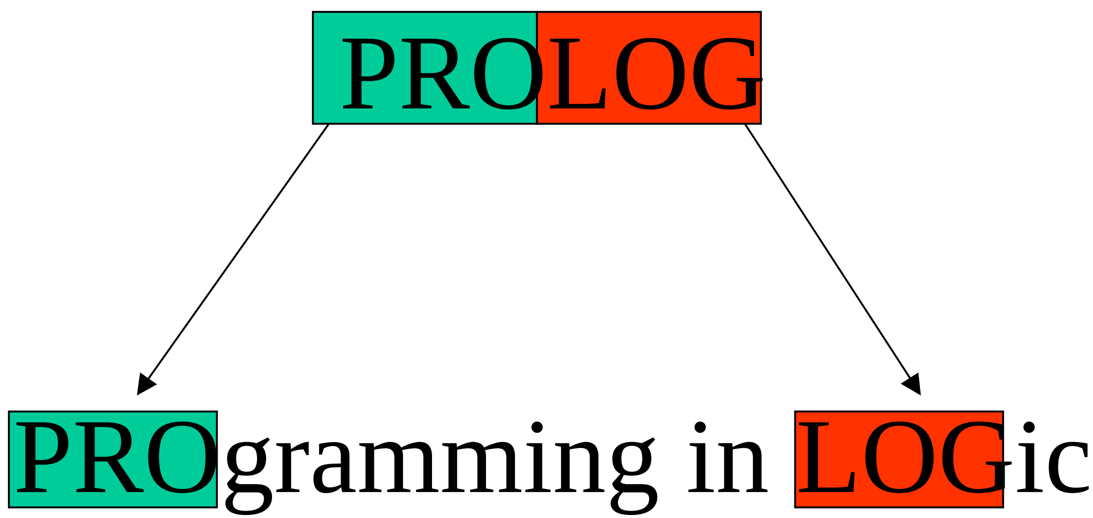
Declarative Programming Languages

It is described, **WHAT** the problem is, but not how to solve it.

Solution is found by the computer

- Prolog
- Goedel
- Escher
- Elf
- Mercury

What means PROLOG?



Programming in Prolog

- Developed in the early 70's
- It is the most popular logic programming language (in Europe, was even more popular than Lisp)
- It is an interpreted language

But recently compilers are developed

- Prolog programs are:
  - sequences of logical sentences
  - only Horn clauses are allowed:

    $Member(x, l) \Rightarrow Member(x, [y|l])$
  - terms can be constant symbols, variables, functional terms
  - syntactically distinct terms assumed to be distinct objects (e.g., A cannot be unified with F(x))
  - Uses "negation-as-failure" operator:

    not P is considered true if language fails to prove P

- Horn form
- Negation by failure: `not(p)` means `p` cannot be proven
- (*Closed World Assumption*)

History of Logic Programming

- History is short
- Theoretical foundations in 1970s
- Kowalski
- First Prolog Interpreter - 1972
  - by Alain Colmerauer in Marseilles
- In 1980 first commercial Prolog Interpreter

- **Algorithm** = **Logics + Control**
- Most known logic programming language = Prolog
  - $B \wedge C \Rightarrow A$
  - $C \wedge B \Rightarrow A$

Predicate Logic in programming

- Programming by description
  - describe the problem's facts
  - built-in *inference engine* combines and uses facts and rules to make inferences
- These are true for any logic programming approach, not only Prolog

Prolog Programming

- Declaring facts about objects and their relationships --> **likes (john,mary)**
- Defining **rules** about objects and relationships
- Asking **Questions** about objects

```
sister-of(X,Y) :- female(X),
                  parents(X,M,F),
                  parent(Y,M,F)
```

Applications of Prolog

- Expert systems (Diagnosis systems)
- Relational Data **Bases**
- mathematical Logic
- abstract Problem solving
- Simulation of human speech and communication
- formal verification of software and hardware
- robot planning and problem-solving
- automatic production and fabrication
- solving symbolic equations
- analysis of biochemical structures
- various areas of knowledge engineering

Prolog Language Example

More family examples…..

```
male(jack).
male(jim).
female(jill). female(mary). female(anne).
father(jack,jim).
father(jack,mary). father(jack,anne).
mother(jill,jim). mother(jill,mary). mother(jill,anne).
brother(X,Y) :- father(Z,Y), father(Z,X), mother(W,Y), mother(W,X),
    male(X).
sister(X,Y) :- father(Z,Y), father(Z,X), mother(W,Y), mother(W,X),
    female(X).

?- brother(X, mary).
```

Another Prolog Example

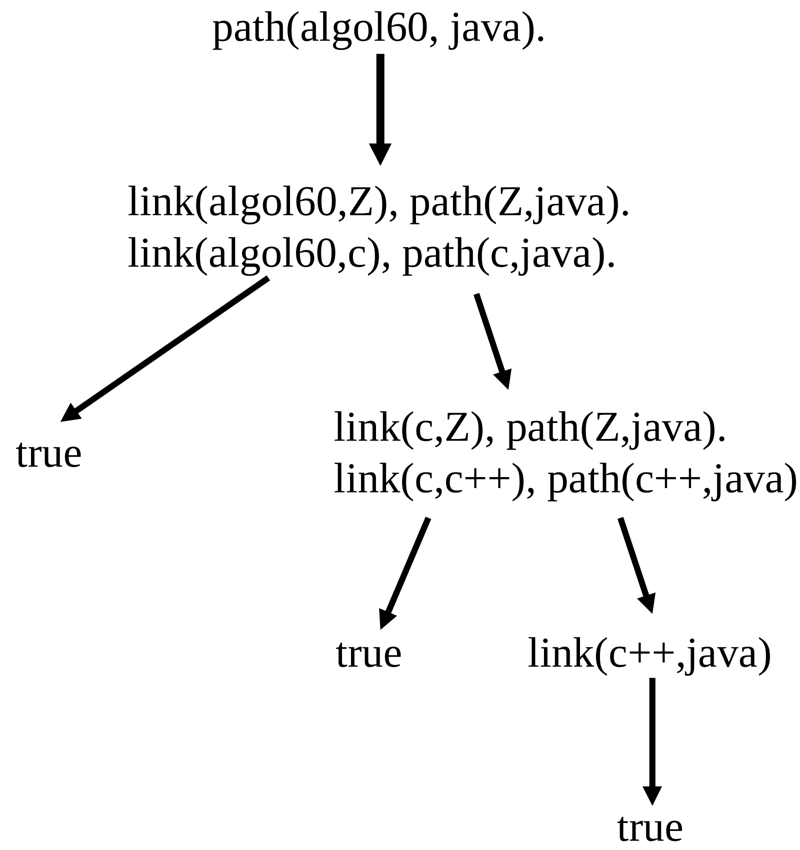
- predicate calculus is good to describe attributed relational worlds such as hierarchical graphs
- Facts, rules, queries

link(algol60, c)
link(algol60, simula)
link(c, c++)
link(simula, smalltalk)
link(c++, java)
path(X,Y) :- link(X,Y).
path(X,Y) :- link(X,Z), path(Z,Y).

?- *path(algol60, java).*

?- path(java, smalltalk).

?- path(c, X).



Hierarchies and graphs are also good for Prolog

Properties of Prolog

- **Negation as failure:** If I can't prove it, it must be false.

```
Not(p) :- p, !, fail.
Not(p) :- true.
```

- **Unification:** Matching in two directions

```
?- f(X,b)=f(a,Y).
X=a
Y=b
```

Prolog's machine

- Backward Chaining
  - to prove `<the head>`, prove `<the body>`
- cut!
- Logic variables
- check in unification algorithm
  - **`x=Pred(x)`** would lead to
    `Pred(Pred(Pred(Pred`

Complete Prolog examples

Example 1

member

:- means <==

- Syntax:

  $\forall x \forall l. Member(x, [x|l])$

  written as

  ```
  > member(X, [X | L])
  ```

  $\forall x \forall y \forall l. Member(x, l) \Rightarrow Member(x, [y|l])$

  written as

  ```
  > member(X, [Y | L]):-
    member(X, L)
  ```

- To check whether 1 is a member of list [1,2,3] we simply type

  ```
  > member(1,[1,2,3])
  ```

- We can enumerate all members of [1,2,3] by typing

  ```
  > member(X,[1,2,3])
  X=1 (press Enter)
  X=2 etc
  ```

Member continued

$\forall x, l\ Member(x, [x|l])$

$\forall x, y, l\ Member(x, l) \Rightarrow Member(x, [y|l])$

Capital letters
Small letters

↓

$Member(x, [x|l])$

$Member(x, l) \Rightarrow Member(x, [y|l])$

Transformation sequence

↓

$Member(x, [x|l])$
$Member(x, [y|l]) \Leftarrow Member(x, l)$

→

```
member(X,[X|L]).

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

Programming in Prolog

Prolog structures:

- clause
- depth first
- ∧ = ,
- **P ∨ Q = P ; Q**

```
a:-
b, c.

a:-
e; f.

f.
d.
b.
```

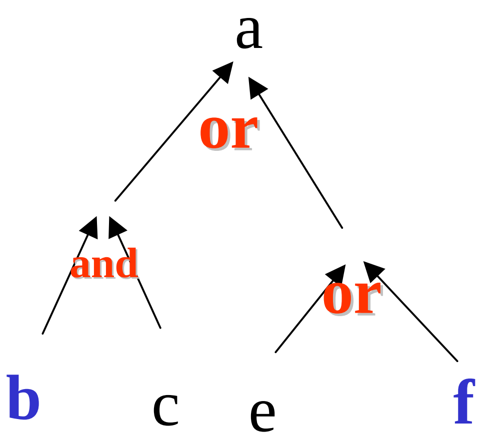
OR

Prolog trace

7 `a:-`
`b, c.`

`a:-`
`e; f.`

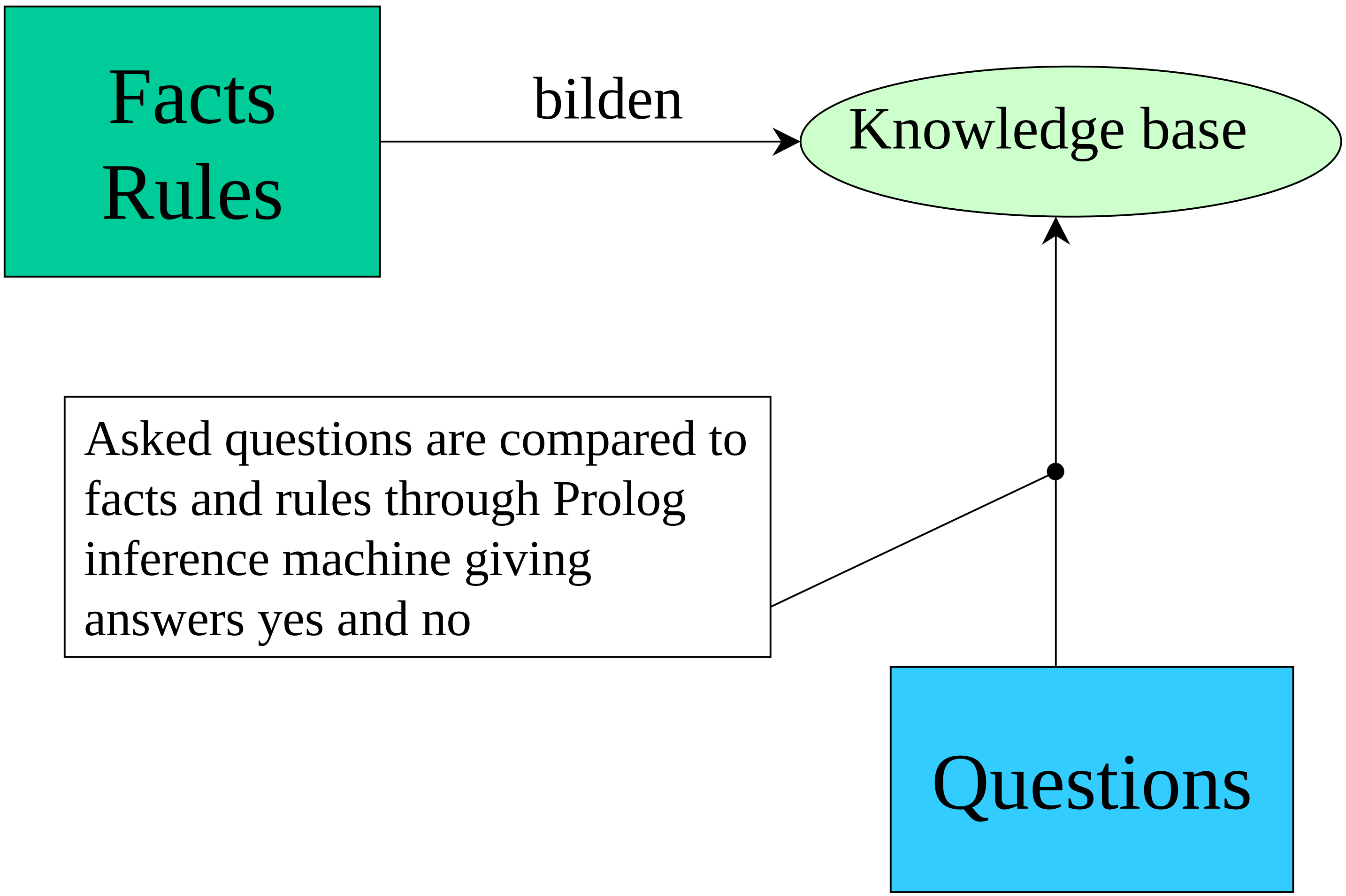
8 **f.**
d.
b.



d

trace, a.
Call: (7) a ?
Call: (8) b ?
Exit: (8) b
Call: (8) c ?
Fail: (8) c ?
Redo: (7) a ?
Call: (8) e ?
Fail: (8) e ?
Call: (8) f ?
Exit: (8) f
Exit: (7) a

Fundamental structure of PROLOG

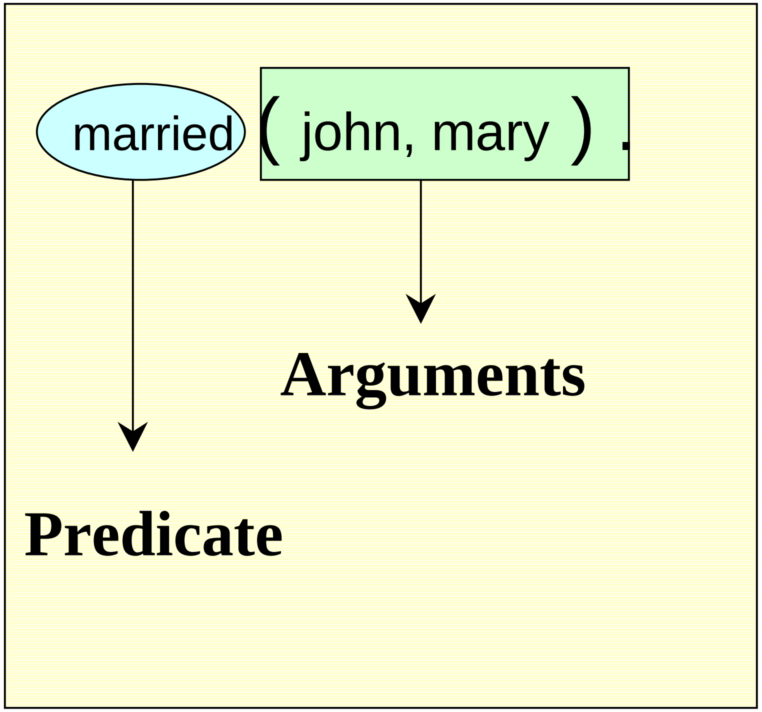


Facts 1

Example:

Prolog description:	Natural meaning:
• ‘man eats‘.	• Man eats.
• it_snows.	• It snows.
• costly(gold).	• Gold is costly
• man(daniel).	• Daniel is a man.
• married(john,mary).	• John is married to Mary.
• has(john,gold).	• John has Gold.
• father(hans, gabriel).	• Hans ist a father of Gabriel.

Facts 2



married(john,mary) is not the same as married(mary,john) .

Prolog does not know what the fact means.

You have to provide this knowledge

Like transitivity or commutativity of relations.

Facts 3

Example:

Given facts:

```
has(john, gold).

has(john, book).

married(john, mary).

maried(joe, barbara).
```

Questions to Prolog Interpreter:

```
?- has(john,gold).
yes

?- married(joe, fish).
no
```

Variables

Knowledge base:

has(john, gold).

has(john, book).

married(john, mary).

maried(joe, barbara).

has(john,X)

X is a Variable.

Prolog answers:

X = gold

X = book

_ is an anonymous variable.

Has john something?

Variables start from capital or underline.

Arithmetics 1

Example 1: factorial

```
fact(0,1).

fact(N,X) :-        N > 0,   M is N - 1,   fact(M,Y),   X is N * Y.
```

```
FUNCTION fact(N : Integer) : Integer;
BEGIN
        IF N = 0 THEN fact := 1
        ELSE IF N > 0 THEN fact := N * fact(N-1);
END;
```

Pascal

Calling:

?- fact(0,N).

N = 1

?- fact(6,N).

N = 720

Arithmetics 2

Example l: Fibonacci Sequence

```
fib(0,1).

fib(1,1).

fib(N,X) :- N1 is N - 1, N2 is N - 2, fib(N1,X1), fib(N2,X2), X is X1 + X2.
```

Calling:

```
?- fib(11,N).

N = 144
```

```
?- fib(4,N).

N = 5
```

Arithmetics 3

+ - * /	Addition, Subtraction, Multiplication, Division
mod	Modulo
// ^	Integer-Division, Power
( )	Priority
is	result of arithmetic calculation
> <	larger, smaller
=> =<	larger or equal, smaller or equal
=:=	equal arithmetic
=\=	non-equal arithmetic

Infix: **(4 + 1)*4** Postfix: ***(4,+(4,1))**

Lists 1

[1,2,3,4,5]

[1 | [2 | [3 | [4]]]]

[father, ‘Hello‘, 1, 3, maried(john,mary)]

[1, [2,3,4], 5]   nested list

[]   empty list

[Head | Tail] = [1,2,3,4]

Head = 1

Tail = [2,3,4]

Unification
(Pattern-Matching)

[X, Y, Z] = [1, 2, 3]

X = 1

Y = 2

Z = 3

Lists 2

Calculate the length of a list:

```
list_length( [], 0 ).
list_length( [H | T], N ) :- list_length(T,M), N is M + 1.
```

Member predicate in a list:

```
member(E, [E|T]).
member(E, [H|T) :- member(E,T).
```

Calculate number of elements in a list:

```
count_element( [], Element, 0).
count_element( [H|T], Element, Sum) :-
        H = Element, count_element(T, Element, X), Sum  is X + 1.
count_element( [H|T], Element, Sum) :-
        count_element(T, Element, Sum).
```

Lists 3

Concatenation:

```
concat([], L, L).
concat([H|T], L, [H|NeueListe] ) :- concat(T,L,NeueListe).
```

Structures

time(datum(16,5,1999),hour(18,30)).

~~time(16,5,1999,18,30).~~

book(author(clocksin,mellish), title(‘Programming in PROLOG‘)).

~~book(clocksin, mellsih, ‘Programming in PROLOG‘).~~

```
cd( band(metallica), title(‘Load‘), length(67) ).
cd( band(metallica), title(‘Reload‘), length(72) ).

?- cd( band(metallica), title(X), _ ).
```

Gives all titles of Metallica.

Example of Backtracking

```
man( john ).
man( george ).

is_father_of( george, mary ).
is_father_of ( george, john ).
is_father_of ( harry, sue ).
is_father_of ( edward, george ).

is_son_of(X,Y) :- is_father_of(Y,X), man(X).
```

```
?- is_son_of(X,Y).

X = john
Y = george

X = george
Y = edward
```

read(X).	Reads from keyboard in X
write(X).	writes X .
get(Ascii).	Reads in Ascii.
put(Ascii).	Writes in ASCII.
nl.	New line.
tab(N).	Writes N spaces.
tell(DATEI).	opens Datei.
told(DATEI).	closes Datei.
append(DATEI).	Appends to Datei.
true	always true
fail	not true

Theorem Provers

- Not only Horn Clauses (complete FOL)
- Other search techniques (e.g. iterative deepening).
- Other treatment of NOT
- New logic programming languages

Additional reading:

- Read section 10.3 :
  - Compilation of Logic Programs
  - Other Logic Programming Languages
  - Advanced control facilities

Summary

Resolution

Resolution proofs

Answering questions

Resolution strategies

Extensions

Programming paradigms

Declarative Languages

Prolog

Reading for the next time

Logic Programming Systems (chapter 10)

"Knowledge Engineering" Chapter 8 in R&N

Sources

1. Richard Benjamins
2. Luger and Stubblefield book has many excellent Prolog examples in later chapters.
2. **Michael Neumann**

eMail
neumann@s-direktnet.de
Homepage
http://www.s-direktnet.de/homepages/neumann/index.htm