Prolog

... an introduction
Introducing Prolog

• Language
  • Declarative defines relationships
    facts + rules + queries a database!
    based on logic (predicate calculus)
  • Procedural C & Lisp

• History
  • Developed as part of the 5th generation (language) project
  • 1972 Marseille / Edinburgh

• Application areas
  • Artificial intelligence, theorem proving, expert systems
  • Database (DB) system
  • Language (Natural and computer) processing
Prolog session & program file

?- man(socrates).
?- mortal(X) :- man(X).
?- mortal(socrates).
  true.
?- 

man(socrates). % socrates is a man fact
mortal(X) :- man(X). % all men are mortal rule

?- mortal(socrates). % is socrates mortal? Query
Using Prolog from Linux

>swipl % start system
>prolog % start system
>swipl –s abc.pl % load and go
>prolog –s abc.pl % load and go

Prolog session:-

?- consult(['abc.pl']). % load program abc.pl
?- ['abc.pl']. % or the quick version!
?- listing. % list the content of abc.pl
?- ... % play with Prolog
?- halt. % exit the system
What can possibly go wrong? 😊

• **Answer** – quite a lot!

• **Common errors**
  - Forgetting the ‘.’ (full stop) after each query or definition
  - Missing ‘,’ comma or ‘;’ semicolon in a query or definition
  - Unbalanced parentheses ( ... ) and [ ... ]
  - Singleton variables – a variable is defined but not used
  - ... and quite possibly a lot more!

• **Solution(s)**
  - Develop programs stepwise (read examples/ use the web)
  - Test all constructions separately

• **Working method**
  - Switch between an editor and the Prolog session – change & test
  - The up arrow in the Prolog session is very useful
How to learn – by example!

- Read example programs (course website / online info).
- Study the example programs for this course.
- Learn TO READ Prolog

**Essential ideas**
- Prolog works by defining goals which in turn are composed of sub-goals – you need to think in terms of goals!
- Prolog solutions “prove” a sequence of sub-goals
- E.g. \(\text{task}(A,D) : - \text{solveA}(A,B), \text{solveB}(B,C), \text{solveC}(C,D).\)
  - A is input; B & C are outputs then inputs; D is the output
  - i.e. D holds the solution
  - the commas are logical conjunctions i.e. A and B and C
  - \text{task} is true if \text{solveA} and \text{solveB} and \text{solveC} are true

**NB:** Name instantiations apply ONLY within the RULE
What **YOU** need to do!

- **READ the following**
  - “Using Prolog” / “Att använda Prolog” on the lab spec webpage
    - [http://www.cs.kau.se/cs/education/courses/dvgc01/PROLOGINFO/Prolog_en.htm](http://www.cs.kau.se/cs/education/courses/dvgc01/PROLOGINFO/Prolog_en.htm)
    - [http://www.cs.kau.se/cs/education/courses/dvgc01/PROLOGINFO/Prolog_se.htm](http://www.cs.kau.se/cs/education/courses/dvgc01/PROLOGINFO/Prolog_se.htm)
  - Go to the Prolog info web page
    - [http://www.cs.kau.se/cs/education/courses/dvgc01/PROLOGINFO/](http://www.cs.kau.se/cs/education/courses/dvgc01/PROLOGINFO/)
  - Read the documentation there
  - I will go through the programs in class with a commentary
  - **Study (and run!)** the program examples on that page
  - Try to invent your own problems and solve them

- **Take breaks to think about how to think in Prolog**
  - Prolog is very different from C or Java (or Lisp)
  - Ask questions – in the labs & lectures – discuss problems!
Program example (Endriss Ch. 1)

see http://www.cs.kau.se/cs/education/courses/dvgc01/PROLOGINFO/plcode/animals.pl

bigger(elephant, horse). % facts
bigger(horse, donkey).
bigger(donkey, dog).
bigger(donkey, monkey).

/********* rules****************/
is_bigger(X, Y) :- bigger(X, Y). % NB tail recursive

/********* tests****************/
?- bigger(elephant, horse). % true – fact exists
?- bigger(elephant, wolf). % false – no rule
?- bigger(horse, monkey). % false – no fact exists ... but
?- is_bigger(horse, monkey). % true – via deduction

is_bigger checks (i) the facts then (ii) tries to deduce the answer
bigger(elephant, horse).
bigger(horse, donkey).
bigger(donkey, dog).
bigger(donkey, monkey).

is_bigger(X, Y) :-
bigger(X, Y).

is_bigger(X, Y) :-
bigger(X, Z),
   is_bigger(Z, Y).

bigger(X, Y).
X = elephant, Y = horse;
X = horse, Y = donkey;
X = donkey, Y = dog;
X = donkey, Y = monkey.

is_bigger(X, Y).
X = elephant, Y = horse;
X = horse, Y = donkey;
X = donkey, Y = dog;
X = donkey, Y = monkey;
X = elephant; Y = donkey;
X = elephant, Y = dog;
X = elephant, Y = monkey;
X = horse, Y = donkey;
X = horse, Y = dog;
X = horse, Y = monkey;
false.
Prolog – call/exit/fail/redo

• These can be viewed as

CALL ➔ EXIT means a predicate has succeeded
CALL ➔ FAIL means a predicate has failed
REDO: repeat until all possibilities have been found
if more rules exist try these in turn until the process FAILs

• CALL / EXIT is “similar” to procedural programming
• REDO / FAIL is unique to Prolog
Use trace to see what happens

?- trace.

\textbf{is\_bigger(X, Y) :- bigger(X, Y).}  // via 1\textsuperscript{st} rule

trace.

[trace] ?- \textbf{is\_bigger(X,Y)}.

\textbf{Call: } (6) \textbf{is\_bigger}(_G3078, _G3079) ? creep  // press return

\textbf{Call: } (7) \textbf{bigger}(_G3078, _G3079) ? creep

\textbf{Exit: } (7) \textbf{bigger}(elephant, horse) ? creep

\textbf{Exit: } (6) \textbf{is\_bigger}(elephant, horse) ? creep

\textbf{X = elephant,}

\textbf{Y = horse ;}  // enter ;

\textbf{Redo: } (7) \textbf{bigger}(_G3078, _G3079) ? Creep

\textbf{Exit: } (7) \textbf{bigger}(horse, donkey) ? creep

\textbf{Exit: } (6) \textbf{is\_bigger}(horse, donkey) ? creep

\textbf{X = horse,}

\textbf{Y = donkey ;}

\textbf{Etc.}
Use trace to see what happens

\[
is\_bigger(X, Y) :- \text{bigger}(X, Z), \text{is\_bigger}(Z, Y). \quad // \text{via 2nd rule}
\]

Redo: (6) \text{is\_bigger}(_G3078, _G3079) ? creep // \(X = \text{elephant}\)

Call: (7) \text{bigger}(_G3078, _G3164) ? creep // \(NB X Z\)

Exit: (7) \text{bigger}(\text{elephant, horse}) ? creep // \(Z = \text{horse}\)

Call: (7) \text{is\_bigger}(\text{horse, _G3079}) ? creep // \(NB Z Y\)

Call: (8) \text{bigger}(\text{horse, G_3079}) ? creep // \(NB Z Y \text{ rule 1}\)

Exit: (8) \text{bigger}(\text{horse, donkey}) ? creep // \(NB Z Y \text{ rule 1}\)

Exit: (7) \text{is\_bigger}(\text{horse, donkey}) ? creep // \(NB Z Y\)

Exit: (6) \text{is\_bigger}(\text{elephant, donkey}) ? Creep // \(NB X Y\)

\(X = \text{elephant, Y = donkey}; \quad // \text{Etc.}\)

This will eventually fail on trying to find solutions to
\[\text{is\_bigger(dog, Y) and is\_bigger(monkey, Y)} \rightarrow \text{no facts! Try it!}\]
...one little “problem”

?- is_bigger(X, monkey). % which animals are > monkey?
X = donkey ; % press ‘;’ for more answers
X = elephant ;
X = horse ;
false. % no more answers.

This can become slightly tedious! Especially if the list is long.
The solution is findall/3

?- findall(X, is_bigger(X, monkey), L).
L = [donkey, elephant, horse].
returns the answer as a list
Program example (Endriss Ex 1.3)

see [http://www.cs.kau.se/cs/education/courses/dvge01/PROLOGINFO/plcode/family.pl](http://www.cs.kau.se/cs/education/courses/dvge01/PROLOGINFO/plcode/family.pl)

female(mary).
female(sandra).
female(juliet).
female(lisa).
male(peter).
male(paul).
male(dick).
male(bob).
male(harry).

parent(bob, lisa).
parent(bob, paul).
parent(bob, mary).
parent(juliet, lisa).
parent(juliet, paul).
parent(juliet, mary).
parent(peter, harry).
parent(lisa, harry).
parent(mary, dick).
parent(mary, sandra).
Program example (Endriss Ex 1.3)

see [Link to the example](http://www.cs.kau.se/cs/education/courses/dvgc01/PROLOGINFO/plcode/family.pl)

```prolog
mother(X, Y) :- parent(X, Y), female(X).
father(X, Y) :- parent(X, Y), male(X).
malediff(X, Y) :- male(X), X \= Y.
pbrother(X, Y) :- parent(Z, X), parent(Z, Y), malediff(X, Y).
domother(X, Y) :- mother(A, X), mother(A, Y), malediff(X, Y).
dofather(X, Y) :- father(A, X), father(A, Y), malediff(X, Y).
abrother(X, Y) :- domother(X, Y), dofather(X, Y).
obrother(X, Y) :- domother(X, Y); dofather(X, Y).

descendant(X, Y) :- parent(X, Y).
descendant(X, Y) :- parent(X, Z), descendant(Z, Y).
display([]).
display([H|T]) :- write(H), nl, display(T).
sbrother(X, Y, [X,Y]) :- parent(Z, X), parent(Z, Y), malediff(X, Y).
siblings(Q) :- findall(Z, sbrother(_,_,Z), L), sort(L,Q).
```
Program example (Endriss Ex 1.3)

see [http://www.cs.kau.se/cs/education/courses/dvgc01/PROLOGINFO/plcode/family.pl](http://www.cs.kau.se/cs/education/courses/dvgc01/PROLOGINFO/plcode/family.pl)

Family database

- Such examples are fairly typical of introductory Prolog
- Most of the relationships are straightforward however there are a few points to note
- `?- pbrother(X,Y).` will give duplicate answers since if a person has both a mother and a father, Prolog will give two answers.
  
  \[
  \begin{align*}
  X &= \text{paul}, \ Y = \text{lisa}; \\
  X &= \text{paul}, \ Y = \text{mary}; \\
  X &= \text{paul}, \ Y = \text{lisa}; \\
  X &= \text{paul}, \ Y = \text{mary}; \\
  X &= \text{dick}, \ Y = \text{sandra}; \\
  \text{false}.
  \end{align*}
  \]
- One solution might be to write a remove duplicates predicate

- Note `malediff(X, Y) :- male(X), X \neq Y.` which says that if X is a brother to Y, X must be male and not the same person as Y.
Program example (Endriss Ex 1.3)

see http://www.cs.kau.se/cs/education/courses/dvpc01/PROLOGINFO/plcode/family.pl

- Checking just the father for example will not find all results since dick and sandra do not have a father

Note the definitions for abrother (AND ‘.’) and obrother (OR ‘;’)  

```
abrother(X, Y) :- domother(X, Y), dofather(X, Y).
obrother(X, Y) :- domother(X, Y); dofather(X, Y).
```

- ?- abrother(X,Y). will not find X = dick, Y = sandra; since dick and sandra do not have a father (the AND requires both)
- ?- obrother(X,Y). Will give the same answer as pbrother (above) since an OR (‘;’) has been used (mother or father)
Program example (library DB)

The facts

book( title([artificial, inteligence]),
    author([patrick, henry, winston]),
    class([technical, ai])).

book( title([common, lisp]),
    author([guy, steele]),
    class([technical, lisp])).

book( title([moby, dick]),
    author([herman, melville]),
    class([fiction])).

book( title([tom, sawyer]),
    author([mark, twain]),
    class([fiction])).

book( title([the, black, orchid]),
    author([rex, stout]),
    class([fiction, mystery])).
Program example (library DB)

• What information do we need?
  • total contents, titles, authors, classes (?), fiction, non-fiction
• How should the rest of the program be designed?

---

display(X) :- tab(3), write(X), nl. % display predicate

listall(_,_,_) :- book(X,Y,Z), nl, display(X), display(Y), display(Z).

Design

• find each book and display the details (title, author, class)
• since the body of the predicate displays the book, listall need not have any named parameters – the “don’t care” symbol (‘_’) is used
• for listall(X,Y,Z), Prolog would display each field again since the query “asks” for matches
Program example (library DB)

- Note: in listall (above, previous slide) for the first book
  - X is title([artificial, intelligence]) % structures
  - Y is author([patrick, henry, winston])
  - Z is class([technical, ai])

```
listtitle(X) :- book(title(X),__,_), display(X).
listauthor(X) :- book(__,author(X),__), display(X).
listclass(X) :- book(__,__,class(X)), display(X).
```

Now
- X is [artificial, intelligence] % lists
- X is [patrick, henry, winston]
- X is [technical, ai]

These predicates extract the title, author and class information from the “fields” of the “record”. In OO terminology, the attributes of the object book.
Program example (library DB)

listlastname(X,Y) :- book(_,author(X),_), last(X,Y), display(Y).

this uses author to extract the author information
for the first book X is [patrick, henry, winston]
X is then passed to last/2 which returns Y as winston
Y is then displayed
Program example (library DB)

listfiction(X,Y,Z) :- book(title(X), author(Y), class(Z)), infiction(X,Y,Z).
infiction(X,Y,Z) :- member(fiction, Z), display(X), display(Y), nl.

listnonfiction(X,Y,Z) :- book(title(X), author(Y), class(Z)), notinfiction(X,Y,Z).
notinfiction(X,Y,Z) :- not(member(fiction, Z)), display(X), display(Y), nl.

Design
• checks to see if a book is classed as “fiction” – or not
• uses member/2 to decide if ‘fiction’ is in the list Z
• uses not(member/2) to decide if ‘fiction’ is not in the list Z
• a “side effect” is that the title (X) and author (Y) are displayed
Program example (library DB)

Using `findall/3` – `findall(X, predicate(...), L).

library    :- findall(_, listall(_,_,_), _).
fiction    :- findall(_, listfiction(_,_,_), _).
nonfiction :- findall(_, listnonfiction(_,_,_), _).
titles     :- findall(X, listtitle(X), _).
authors    :- findall(X, listauthor(X), _).
lastnames  :- findall(Y, listlastname(_,Y), _).
classes    :- findall(X, listclass(X), _).

Note the use of the “don’t care” character (‘_’) – why?
Program example (library DB)

Using `findall/3` – `findall(X, predicate(...), L).

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>library</td>
<td>list all the books in the library</td>
</tr>
<tr>
<td>fiction</td>
<td>list all the fiction books in the library</td>
</tr>
<tr>
<td>nonfiction</td>
<td>list all the non-fiction books in the library</td>
</tr>
<tr>
<td>titles</td>
<td>list the titles of all the books in the library</td>
</tr>
<tr>
<td>authors</td>
<td>list the authors of all the books in the library</td>
</tr>
<tr>
<td>lastnames</td>
<td>list the authors’ last names ...</td>
</tr>
<tr>
<td>classes</td>
<td>list the classes of all the books in the library</td>
</tr>
</tbody>
</table>
Program example (library DB)

Testing the system
  • the predicates library... classes are meant for interactive use
  • to test all the predicates in the program, another predicate – showall, may be defined (including display information)

```
showall :-
    write('Library:'), nl, library, nl,
    write('Fiction:'), nl, fiction, nl,
    write('Nonfiction:'), nl, nonfiction, nl,
    write('Titles:'), nl, titles, nl,
    write('Authors:'), nl, authors, nl,
    write('Lastnames:'), nl, lastnames, nl,
    write('Classes:'), nl, classes, nl.
```

20/04/2017 22:10 DFR - Prolog Intro
Program example (library DB)

Library:

title([artificial, intelligence])
author([patrick, henry, winston])
class([technical, ai])

title([common, lisp])
author([guy, steele])
class([technical, lisp])

title([moby, dick])
author([herman, melville])
class([fiction])

title([tom, sawyer])
author([mark, twain])
class([fiction])

title([the, black, orchid])
author([rex, stout])
class([fiction, mystery])

Titles:
[artificial, intelligence]
[common, lisp]
moby, dick]
tom, sawyer]
[the, black, orchid]

Authors:
[patrick, henry, winston]
guy, steele]
[herman, melville]
[mark, twain]
xen, stout]

Lastnames:
winston
steele
melville
twain
twain

Classes:
[technical, ai]
[technical, lisp]
[fiction]
[fiction]
[fiction, mystery]
Program example (library DB)

Testing the system

• The output from showall can be redirected to a file

---
testall :- tell('library.out'), showall, told, halt.
---

tell open a file for write (redirect the output stream)
told close the file (output is redirected to default)
halt exit the Prolog system
---
# Program example (library DB)

**Testing the system** - after **halt**. We are back in Linux – one way is

```bash
echo "*** saving previous test output ***"  # ./runlib

cp library.out library.old

echo "*** running library program ***"

echo "*** enter testall. & return ***"

prolog -s library.pl -q  # start Prolog

-------------

echo "*** checking library test ***"  # ./checklib

echo "*** checking result files ***"

ls -lt library.old

ls -lt library.out

echo "*** comparing test results (diff) ***"

diff library.out library.old

echo "*** end of library check ***"
```

---

20/04/2017

22:10 DFR

- Prolog Intro

28
Library DB –Lisp version

(defun make-book (title author class)
  (list (list 'title title)
        (list 'author author)
        (list 'class class)))

(defun baw (book author)
  (if (eql 'author (first (first book)))
      (cons (list 'author author) (rest book))
      (list 'author author)
    (cons (first book) (baw (rest book) author))))

(defun book-title (book)
  (second (assoc 'title book)))

(defun book-author (book)
  (second (assoc 'author author)))

(defun book-class (book)
  (second (assoc 'class book)))

(defun fictionp (book)
  (member 'fiction (book-class book)))

(mapcar #'<function> books)
(remove-if #'<predicate> books)
(find-if #'<predicate> books)
(count-if #'<predicate> books)

; apply <function> to each element
; filter
; find
; count
Library DB – Lisp version

Prolog – book structure

```
book (title([artificial, intelligence] )
   author([patrick, henry, winston] )
   class([technical, ai] )
)
```

Lisp – book structure

```
(setf book-ex2
   '\( (title (Artificial Intelligence )) ; Title
     (author (Patrick Henry Winston)) ; Author
     (class (Technical AI )) ) ; Class
)
```