Prolog

Prolog was invented in the early 1970s by Alan Colmerauer and his colleagues in Marseille: Their major interest was Natural Language Processing. The deductive mechanism behind Prolog is based on Robert Kowalski's work on refinements of resolution (SLD) for Horn clauses.

- Monmouth has Sicstus Prolog developed by the Swedish Institute of Computer Science. `http://www.sics.se`
- An excellent Prolog - Amzi Logic Explorer free for PCS(Both Linux and Windows). `http://www.amzi.com`

References

- Learn Prolog Now! by Patrick Blackburn, Johan Bos and Kristina Striegnitz
- Chapters 5 and 6 of Brachman and Levesque
- Introduction to Programming in Prolog by Danny Crookes. NewYork: Prentice Hall 1988
- Programming in Prolog by W.F.Clocksin and C.S. Mellish Fourth Edition. Berlin: Springe-verlag 1994

Horn Clauses

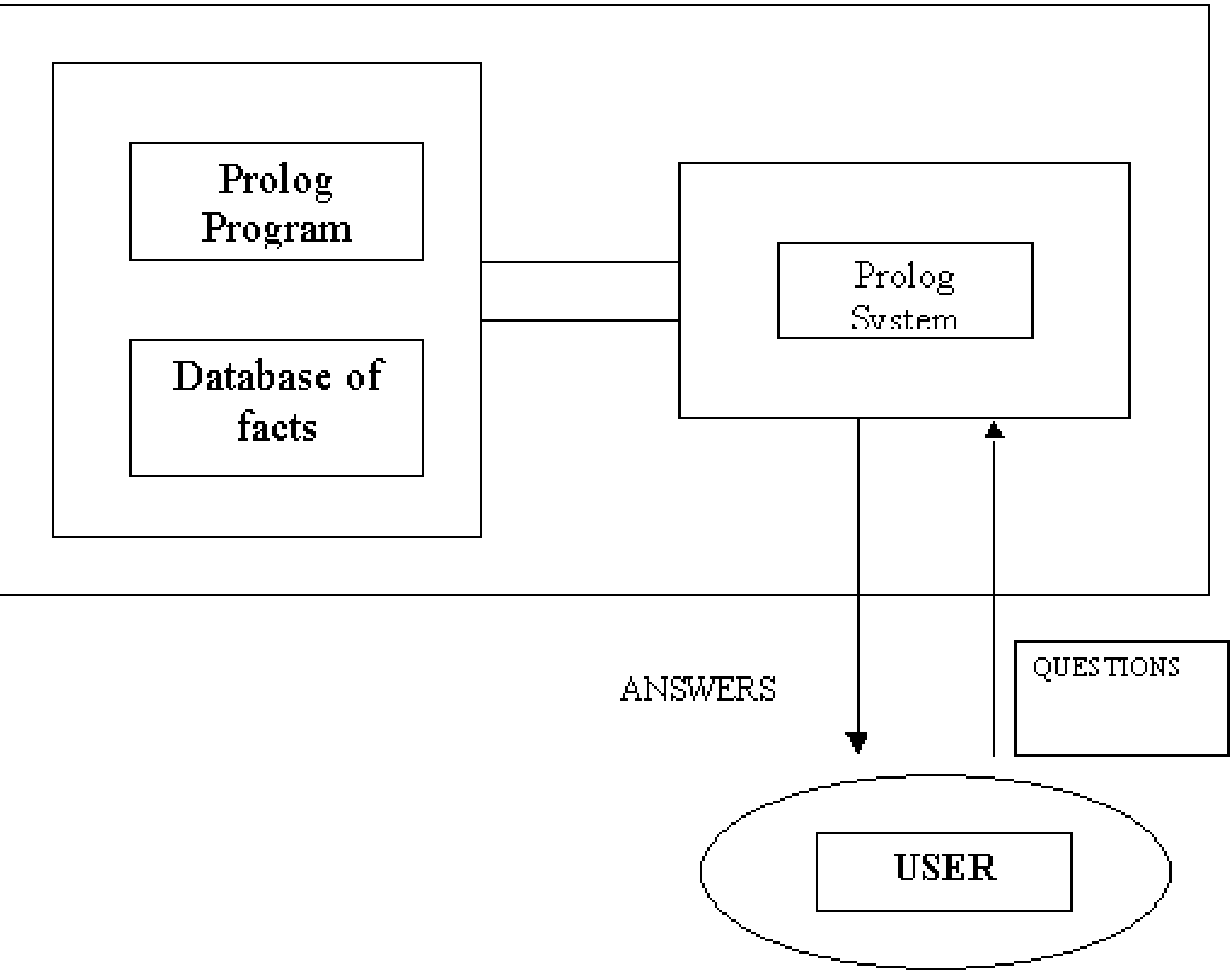
Horn Clauses are clauses that have at most one positive literal. If there is one positive literal, then the clause is a rule whose consequent is the single positve literal and whose antecedent is a conjunction of positive literals.

- Rules `H :- B1,....Bn`
- A fact is a single postiive literal.
  Facts `H :-`
- A goal (query) is a conjunction of negative literals.
  Goals `:- B1,...Bn`

Interaction

The user submits questions to the prolog system and receives answers based on information contained in the database of facts, and the rules that have been loaded into the prolog system.

Interaction



A Sample Database

```
has_vacancy(harvard, secretary).
has_vacancy(prentice_hall, author).
has_vacancy(ibm, salesman).
has_vacancy(hertz, driver).
has_vacancy(nasa, programmer).
has_vacancy(prentice_hall, secretary).

trained_as(michael, programmer).
trained_as(fred, taxidermist).
trained_as(mary, driver).
trained_as(joe, secretary).
trained_as(michael, salesman).
trained_as(elizabeth, secretary).
```

DB Continued

```
accurate(elizabeth).
accurate(mary).
accurate(michael).
accurate(fred).
outgoing(michael).
outgoing(mary).
outgoing(elizabeth).
co_ordinated(joe)
hard_working(mary)
hard_working(joe).
hard_working(michael).

literate(michael).
clear_thinking(elizabeth).
clear_thinking(michael))
intelligent(mary).
imaginative(michael).
```

Queries

- `?-clear_thinking(elizabeth).`
- `?-clear_thinking(fred).`
- `?-clear_thinking(X)`
- `?-imaginative(X), hard_working(X).`

Queries Continued

Note that if there is more than one object satisfying the query, the user can type a semicolon (;) afer the answer and prolog will search for another binding for the variables. This can continue until prolog can not find another binding. It will then return `no`.

But Prolog can do much more than mere retrieval of facts!

Prolog Rules

NASA might employ someone if that person is clear_thinking and reliable.

```
might_employ(nasa, X) :-
      clear_thinking(X),
      accurate(X).
?- might_employ(nasa, elizabeth).
Yes

?- might_employ(nasa,fred).
No

? might_employ(nasa, X).
  X=elizabeth;
  X=michael;
No
```

Rules Continued

If the above rule is added to the database other plausible rules are:

```
acceptable(Candidate, Employer, Skill) :-
      has_vacancy(Employer, Skill),
      trained_as(Candidate, Skill).

acceptable(Candidate, Emplooyer, Skill) :-
      has_vacancy(Employer, Skill),
      \+(trained_as(Candidate, Skill),
      could_be_trained_as(Candidate, Skill)
could_be_trained_as(X, secretary) :-
      accurate(X),
      literate(X),
      outgoing(X).
could_be_trained_as(X, programmer) :-
      clean_thinking(X),
      accurate(X),
      intelligent(X).
could_be_tranined_as(X, driver):-
      co_ordinated(X)
      hard_working(X),
```

Examples

```
?-could_be_trained_as(michael, secretary).

?-could_be_trained_as(mary, programmer).
```

Negation

Note that the

```
\+
```

is the negation operator in Sicstus Prolog. In Amzi prolog the negation operator is the standard `not` as in `not (member(X, [a,b,c]))`

Using Prolog

Type your program into a file and save it. Save it with the suffix `pl` as in `kb.pl`. Then enter prolog.

```
?- listing.
```

```
?- [kb2].
```

```
?- listing.
?- halt.
```

A Family

```
male(philip). male(charles). female(liz).
child_of(charles, philip).
child_of(charles, liz).
parent_of(philip,charles).
parent_of(liz,charles).
father_of(X,Y):- parent_of(X,Y),
                 male(X)
```

Descendant

Consider the problem of trying to specify the concept of descendant.

```
descendant_of(X,Y) :- child_of(X,Y).

descendant_of(X,Y) :- grandchild_of(X,Y).

descendant_of(X,Y) :- great_grandchild_of(X,Y).

grandchild_of(X,Y) :- child_of(X,Z),
                      child_of(Z,Y)

great_grandcdhild_of(X,Y) :- child_of(X,Z),
                             grandchild_of(Z,Y).


great_great_grandchild_of(X,Y) :- child_of(X,Z),
                      great_grandchild_of(Z,X).
```

Tedious !, Incomplete ! **descendants** of Y are Y's children, along with their **descendants**

Recursive Rules

But with recursive rules this is easy.

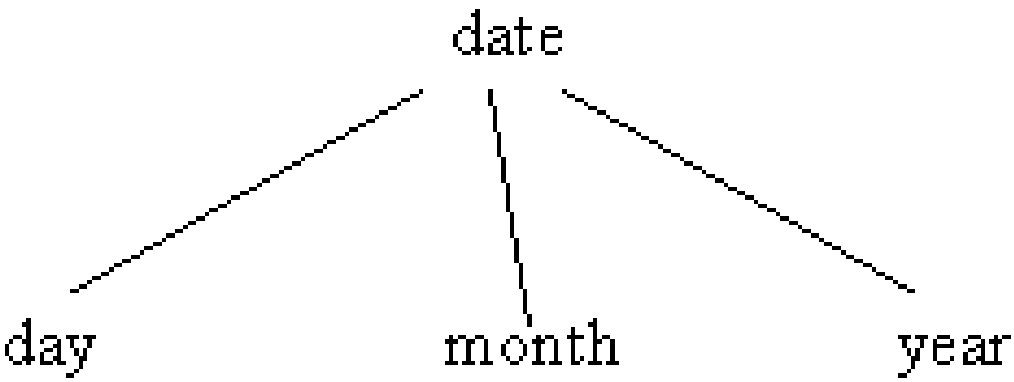
X is a **descendant** of Y *either if* X is a child of Y, *or if* X is a **descendant** of a child of Y.

```
descendant_of(X,Y) :- child_of(X,Y).
descendant_of(X,Y) :- child_of(C,Y),
                      descendant_of(X,C).
?- descendant_of(X, elizabeth).
```

Structured Objects

Use of term structure enables one to fully utili
relatively simple expressivity of Prolog.

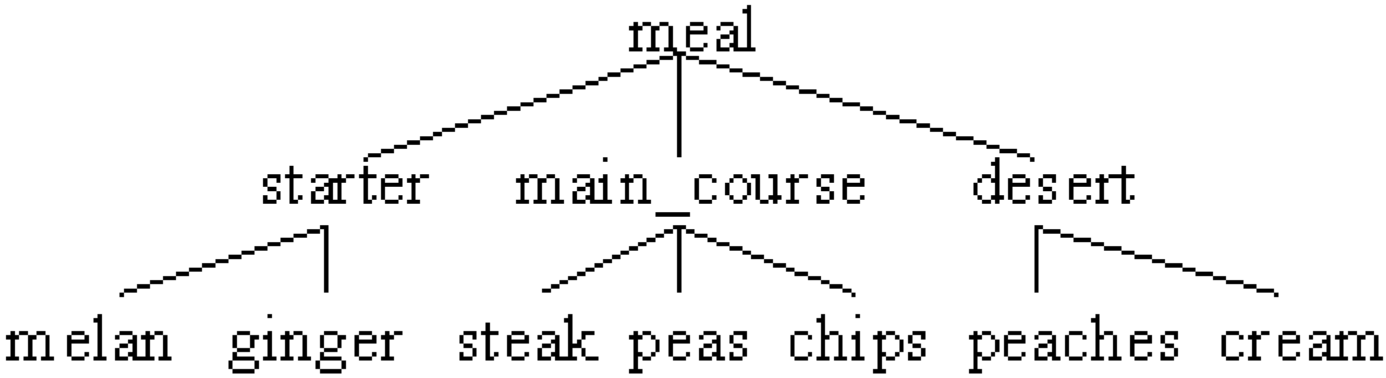
`Object-kind(component1, component2, ......)`



1. date(Day,Month, Year)
   date(31, january, 1988)
   date(25, december,1990)

Structured Objects

2. meal(starter, main_course, desert)




```
main_course(steak, peas,chips)
meal(starter(melon, ginger),
     main_course(steak, peas, chips),
     desert(peaches, cream))
```

Structured Objects Continued

```
3.
book( Author, Title, Classifcation)
book( shakespeare, macbeth, qt-13....)


date_of_birth(Person, Date)
date_of_birth( fred, date(1, february, 1959)).
date_of_birth( shakespeare, date(26, april, 1564
?- date_of_birth( shakespeare, D).
D= date(26, april, 1564)).
?- date_of_birth( P, date(26, april,  1564)).
P=shakespeare
```

Example: Library Catalogue

```
in_library( book( melville,
                  moby_dick,
                  4r_14_s8)).

in_library( book ( shakespeare,
                   romeo_and_juliet,
                   4r_49_s35)).

on_loan( Book, Borrower, Due_date).

on_loan( book (melville,
               moby_dick, 4r_14_s8),
               robinson,
               date(21, november, 1988)).

on_loan(book(shakespeare,
             romeo_and_juliet, 4r_49_s25),
             wilson,
             date( 7, september, 1988)).
```

Lists

But the most important structured object of all is the List – treated specially in Prolog.

```
[ ] - empty list
[ tennis, baseball, sailing, reading, judo ]
[ computing, programming, prolog, AI ]
[ tennis | X ]
```

Lists Cont

(compare as : head , tail) The head of the list above is tennis and X is the tail. Example:

```
all_rich( List )

A list is all rich if
  either
     the list is empty
  or
     the list has the structure [ Person1 | Tail
             and
     Person1 is rich
             and
    Tail is all rich.

all_rich([]).

all_rich([Person1 | Tail]) :- rich(Person1),
                              all_rich(Tail ).
```

Example: Member

A very simple program defines the member relatio

Definition of Member

```
member(X, [X | _]).
member(X, [_ | Y]) : - member(X,Y).
?- member(d, [a,b,c,d,e,f,g]).
YES
?- member(2, [3,a,4,f]).
NO
```

Example: Append

Another very simple function definds the append

```
append([],L,L).
append([ X|L1], L2, [X | L3])
            : - append( L1, L2, L3).
?- append(X,Y,[a, b,c]).
X= []
Y= [a, b, c]?;

X= [ a ];
Y= [b, c]?;

X= [ a, b],
y= [ c ]?;

X= [ a, b ,c],
Y= []?;

no
```

Arithmetic

```
variable is expression
?- X is 2 * 8 + 5.
X=21
?- X is 12, X is 10.
no.
?- X is 12, Y is 3 * X -1.
X=12, Y=35.
```

DB Continued

```
in_range(N, Lower, Upper)
              :- N >= Lower, N <= Upper.

?- in_range(10, 1 ,100)
Yes
?- in_range(0, 1,10)
No.

Sum of a List

Sum([ ],0 )
Sum([Head | Tail ], S) :- Sum(Tail, T),
                           S is Head + T.
```

Cut!

```
foo :- a, b, c, !, d, e, f
```

When a cut is encountered as a goal, the system thereupon becomes committed to all choices made since the parent goal was invoked. All other alternatives are discarded. Hence an attempt to re-satisfy any goal between the parent goal and the cut goal will fail.

```
facility(Pers, Fac):-
        book_overdue(Pers, Book),
        !,
        basic_facility(Fac).
```

Cut Continued

```
facility(Pers, Fac):- general_facility(Fac).
basic_facility(references).
basic_facility(enquiries).
additional_facility(borrowing).
additional_facility(inter_library_loan).
general_facility(X) :- basic_facility(X).
general_facility(X) :- additional_facility(X).
book_overdue('C.Watzer', book10089).
book_overdue('R.Scherl', book29907).
client('A. Sones').
client('R.Scherl').
?-client(X), Facility(X,Y).
```