Artificial Intelligent Systems

(BMA-EL-IZB-LJ-RE 1. year 2024/2025)

Introduction to Prolog

Simon Dobrišek

Lecture Topics

- What is Prolog?
- Prolog syntax and semantics
- Writing Prolog programs
- Prolog interpreter system
- Tracing Prolog programs
- Prolog programming tips and examples

What is Prolog?

- Prolog is a declarative computer programming language.
- It differs from traditional imperative/procedural languages.
- Programmers translate a description of a problem from a natural language into a formal notation.
- Prolog clauses are statements about what is true about a problem, instead of instructions how to accomplish the solution.
- The Prolog system uses the clauses to work out how to accomplish the solution by searching through the space of possible solutions.

Imperative/Procedural Programming

- Imperative programming is a programming paradigm that uses statements that change a program's state.
- An imperative program consists of commands for the computer to perform.
- Imperative programming focuses on describing how a program operates.
- Procedural programming is a type of imperative programming in which the program is built from one or more procedures (also known as subroutines or functions).
- A program in a procedural programming language is:

Program = Algorithm + Data Structures

Declarative Programming

- Declarative programming is a non-imperative style of programming in which programs describe their desired results without explicitly listing commands or steps that must be performed.
- Prolog is a declarative programming language and has its roots in the first-order logic.
- In Prolog, program logic is expressed in terms of relations, and a computation is initiated by running a query over these relations.
- A program in Prolog is (according to Kowalski):

Program = Logical Statements + Execution Control

Characteristics of Prolog

- Prolog approximates the first-order logic.
- Prolog programs are sets of Horn-like clauses.
- Logical inference is implemented using resolution.
- Search is made by backtracking with variable unifications.
- Variables are unknowns and not memory locations.
- Prolog does not distinguish between inputs and outputs - it solves relations/predicates.

Horn Clauses

- Horn clauses are the basis of logic programming, where it is common to write definite clauses in the form of an implication:

$$(p \wedge q \wedge \ldots \wedge t) \rightarrow u$$

- In logic programming, a definite clause behaves as a goal-reduction procedure and the Horn clause behaves as the procedure:

to show u, show p and show q and ... and show t.

- To emphasize this reverse use of the clause, it is often written in the reverse form:

$$u \leftarrow (p \wedge q \wedge \ldots \wedge t)$$

- In Prolog this is written as:

```
u :- p, q, ..., t.
```

Horn Clauses

- The Horn clause in Prolog syntax:

```
conclusion :-
        condition1,
        condition2,
        ...
        conditionn.
```

  is called a rule.

- A clause without conditions is called a fact.
- With facts and rules we describe relations between objects.
- In a dialogue with the Prolog interpreter, the user is writing rules and facts and queries for the validity of specific statements.
- If the interpreter is able to confirm the validity of the given statement then it is registered as the true statement.

The First Example Program

- An example program, consisting of three facts and one rule :

```
child(emily,linda).
child(david,linda).
child(linda,mary).

parent(X,Y) :-
       child(Y,X).
```

- The Prolog interpreter prompts with:

```
?-
```

- Now we can verify facts with the interpreter.

```
?- child(linda,mary).
  true

?- child(emily,mary).
  false
```

The First Example Program

- We can also verify partial facts using variables (unknowns):

```
?- child(X,mary).
  X = linda
  true
?- child(david,X).
  X = linda
  true
?- child(X,Y).
  X = david
  Y = linda;
  X = emily
  Y = linda;
  X = linda
  Y = mary;
  false
```

Prolog Syntax and Semantics

- Prolog sentences end with a dot.
- Prolog is dynamically typed - it has a single data type, the term, which has several subtypes: atoms, numbers, variables and compound terms.
- An atom is a general-purpose name with no inherent meaning; Atoms containing spaces or certain other special characters must be surrounded by single quotes.
- Variables are denoted by a string consisting of letters, numbers and underscore characters, and beginning with an upper-case letter or underscore.
- A variable can become instantiated (bound to equal a specific term) via unification.
- A single underscore (_) denotes an anonymous variable and means "any term".

Prolog Syntax and Semantics

- A compound term is composed of an atom called a "functor" and a number of "arguments", which are again terms.
- Compound terms are ordinarily written as a functor followed by a comma-separated list of argument terms, which is contained in parentheses.
- The number of arguments is called the term's arity. An atom can be regarded as a compound term with arity zero.
- Special cases of compound terms are lists and strings.
- Lists are finite sequences of elements. The brackets are the beginning and the end of the list, and the commas separate the various elements. A list can contain all sorts of elements, and different types of element can occur in the same list.
- Sequences of characters surrounded by quotes (strings) are equivalent to a list of (numeric) character codes.

Types of Prolog Clauses

- Prolog clauses can be facts, rules, commands or questions.
- A rule is composed of a head predicate and a body, separated by the sign `:-` and terminated by a dot.

```
head :- body.

stupid :- handsome.

trip :-
       sunny,
       car.

parent (X, Y) :- child(Y, X).
```

- A rule is read as "Head is true if Body is true“.

Rule Clauses

- A rule's body consists of calls to predicates that are called the rule's goals.
- The built-in predicate `,/2` (meaning a 2-arity operator with name `,`) denotes conjunction of goals, and `;/2` denotes disjunction.
- Conjunctions and disjunctions can only appear in the body, not in the head of a rule.

```
beats(X,Y) :-
      resigned(Y,X);
      (points(X,Tx),
      points(Y,Ty),
      Tx > Ty).
```

Fact Clauses

- Rules clauses with empty bodies are called facts.

```
raining.                % It is raining!
child(david,linda). /* David is Linda's child! */
equals(X,X).
cardinality([],0).
tracing_on.
```

- A fact is equivalent to a rule with a body that is always true:

```
cat(tom).
cat(tom) :- true.
```

Commands and Questions

- A rule clauses without a head are commands.

```
:- goal1,goal2;goal3,goal4.
:- consult('program.pl').
:- write('Hello world!').
:- start.
```

- A question in Prolog is similar to a command:

```
?- raining.
?- raining,trip.
?- child(david,linda).
?- child(david,X).
?- child(X,Y).
?- old(david,X), X > 10.
```

Arithmetic in Prolog

- Prolog provides a number of basic arithmetic tools for manipulating numbers.
- Posing the following queries yields the following responses:

```
?- 8 is 6+2.
  yes

?- 3 is 6/2.
  yes

?- 1 is mod(7,2).
  yes
```

- More importantly, arithmetic questions can have variables:

```
?- X is 6*2.
  X = 12

?- R is mod(7,2).
  R = 1
```

Arithmetic in Prolog

- An example program of Euclid's method for finding the greatest common divisor.
- A program in the programming language C:

```
int divisor(int a, int b)
{
        while(a != b)
                if(a > b)
                        a -= b;
                else
                        b -= a;
        return a;
}
```

Arithmetic in Prolog

- A program in Prolog:

```
divisor(A,A,A).

divisor(A,B,Divisor) :-
        A > B,
        A1 is A - B,
        divisor(A1,B,Divisor).

divisor(A,B,Divisor) :-
        divisor(B,A,Divisor).
```

- Verifications of complete and incomplete facts:

```
?- divisor(24,30,X).
  X = 6

?- divisor(24,30,6).
  true

?- divisor(24,30,5).
  ... ⌛
```

Arithmetic in Prolog

- An improved program in Prolog:

```
divisor(A,A,A).

divisor(A,B,Divisor) :-
        A > B,
        A1 is A - B,
        divisor(A1,B,Divisor).

divisor(A,B,Divisor) :-
        B > A,
        B1 is B - A,
        divisor(B1,A,Divisor).
```

- Verifications of the false facts:

```
?- divisor(24,30,5).
false
```

Literal, Predicate and Operator

- In computer science, a literal is a notation for representing a fixed value in source code.
- A literal in Prolog is headed by a predicate symbol followed by a series on n arguments/objects in parentheses, separated by commas:
- The predicate symbol determines the relation between objects.
- The number of arguments ($n \geq 0$) defines the arity of the predicate.
- A Predicate is uniquely determined by the predicate name and its arity.

Prolog predicates

- In the following rule,

```
beats(X,Y) :-
        resigned(Y,X);
        (points(X,Tx),
        points(Y,Ty),
        Tx > Ty).
```

- There are predicates `beats/2`, `resigned/2`, `points/2`, and `>/2`.
- The predicate `>/2` is an operator that is written between the arguments for the sake of readability.
- The built-in predicates/operators `,/2` denotes conjunction of goals, and `;/2` denotes disjunction.
- The predicate name must be an atom, and the arguments are any valid Prolog terms.

Prolog Procedures

- Procedure consists of all the clauses that defines the same predicate.
- The following Prolog program

```
mother(linda,david).
mother(linda,emily).
mother(mary,linda).

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

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

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

contains three procedures for the predicates `mother/2`, `parent/2` and `grandmother/2`.

Prolog Terms

<table>
<tr><th colspan="3">Terms</th></tr>
<tr><th colspan="2">Simple terms</th><th>Structures</th></tr>
<tr><th>Variables</th><th colspan="2">Constants</th><td rowspan="2"><code>person(PIN,name(Name,
Surname),date(Day,
Month, Year))</code></td></tr>
<tr><td><code>X
Result
_x1
H_A
O1

_</code></td><th>Atoms</th><th>Numbers</th></tr>
<tr><td></td><td><code>david
x_13
:-
<->
'David'
'A=B'</code></td><td><code>0
9
21
3.14</code></td></tr>
</table>

Prolog Operators

- Operators are functors/predicates with one or two arguments written without the use of brackets.
- A single-arity operator can be written before the argument (prefix) or following the argument (postfix).
- A double-arity operator is written between the arguments (infix).
- Operators increase the readability of Prolog programs.

```
+(+(*(5,*(X,X)),*(100,X)),/(10,+(X,1)))).

5*X*X + 100*X + 10/(X+1).
```

Translating Natural Language Statements to Prolog Clauses.

- *"Linda plays the piano".*

```
plays(linda,piano).
```

- *"Nancy plays all the instruments that have a keyboard."*

```
plays(nancy,Instrument) :-
        has(Instrument,keyboard).
```

- *"Does David play an instrument?"*

```
plays(Person) :-
        instrument(Instrument),
        plays(Person,Instrument).

?- plays(david).
```

- *"Which instrument does James play?"*

```
?- plays(james, Instrument).
```

Translating Natural Language Statements to Prolog Clauses.

- *"James plays all the instruments that Nancy or Linda play."*

```
plays(james,Instrument) :-
        instrument(Instrument),
        (plays(nancy,Instrument);
        plays(linda,Instrument)).
```

- *"Who knows how to play the guitar and the harmonica?"*

```
?- plays(Person, guitar),plays(Person, harmonica).
```

- *"Is there anybody who knows how to play the guitar and the harmonica?"*

```
?- plays(_, guitar),plays(_, harmonica).
```

Ambiguities in Prolog Programs.

- Get rid of the ambiguity:

```
child(leo,nancy).
child([leo,peter],nancy).
child(nancy,2).              % Nancy has two children.
```

- Different predicates should be used:

```
child(leo,nancy).
children([leo,peter],nancy).
number_of_children(nancy,2).
```

Interpretation of Prolog Clauses

- Translate the following Prolog clauses to natural language sentences.

```
likes('Linda','David').
```

"Linda likes David."

```
symbol(==>,implication).
```

"'==>' is the symbol for logical implication."

```
symbol(&,conjuction).
```

"'&' is the symbol for logical conjunction."

```
symbol(V,disjunction).
```

"Whatever it is the symbol for logical disjunction."

```
symbol(*,Star).
```

"The asterisk is a sign of anything."

Interpretation of Prolog Clauses

- Translate the following Prolog clauses to natural language sentences.

```
spend(Linda,1000) :- income(Linda,INC), INC > 1000.
```

"Somebody can spend 1000,
if somebody's income is greater than 1000."

```
spend(david,_100) :- income(david,_200), _200 > _100.
```

"'David can spend less than he is paid."

```
likes(Linda,David).
```

"Everybody likes everybody."

```
?-likes(Linda,David).
```

"Who likes who?"

Interpretation of Prolog Clauses

- Translate the following Prolog clauses to natural language sentences.

```
?-likes(Linda,_).
```

"Who likes somebody?"

```
?-likes(_,_).
```

"Does somebody like somebody?"

```
?-not(likes(Linda, 'David')).
```

"Does nobody like David?"

Readability of Prolog Clauses

- Improve the readability of the following Prolog clauses.

```
grandmother(X,Y) :-
        mother(X,Z),(mother(Z,Y);father(Z,Y)).


%-----------------------------------------
grandmother(Mother,Grandchild) :-
        mother(Mother,Child),
        parent(Child,Grandchild).
parent(Parent,Child) :-
        mother(Parent,Child);
        father(Parent,Child).
```

Readability of Prolog Clauses

- Improve the readability of the following Prolog clauses.

```
way_home(Office,Home) :- dog(Office,Station1),
        (train(Station1,Station2);
        bus(Station1,Station2)),
        dog(Station2,Home).


%-----------------------------------------
way_home(Office,Home) :-
        dog(Office,Station1),
        transport(Station1,Station2),
        dog(Station2,Home).
transport(Station1,Station2) :-
        train(Station1,Station2);
        bus(Station1,Station2).
```

Prolog Lists

- Prolog lists are specific data structures in the form of ordered sets of elements.
- The order of elements is important.
- Every list has a head and a tail.
- The head is the first list element and the tail is the list without the first element.

```
[obj1,obj2, ... objn]          % Basic form
[obj1 | [obj2, ... objn]]      % With separate head and tail
[]                             % An empty list
[o1, o2, o3 | [o4, ... on]] % The generalized use of |
```

Lists in Prolog

- Examples of lists with separate heads and tails.

```
[linda,david,john]          [linda | [david,john]]
[1,2,3,5|X]                 [1 | [2,3,5|X]]
[X|Y]                       [X | Y]
[f(city,[1,2,3])]           [f(city,[1,2,3]) | []]
```

- Breaking the list on its head and tail.

```
[obj1,obj2, ... objn] = [Head|Tail].
```

- The generalized use of the operator `|`.

```
[a,b,c] = [a|[b,c]] = [a|[b|[c]]] = [a|[b|[c|[]]]]
[a,b,c] = [a,b|[c]] = [a,b,c|[]] = [a|[b,c|[]]]
```

Lists

- Break the following lists into heads and tails.

```
[a,b,c] = [H|T].
  H = a
  T = [b, c]

[[a,b,c],e|[d,f]] = [[H|T1]|T2].
  H = a
  T1 = [b, c]
  T2 = [e, d, f]

[] = [H|T].
  false

[X,[Y]] = [H|T].
  X = H
  T = [[Y]]
```

Strings in Prolog

- Prolog strings are lists of non-negative integers.
- These integers represent string character codes according to the used coding table (ASCII, UNICODE, …).
- Prolog does not distinguish between

```
"Prolog"      [80,114,111,108,111,103]
```

- The predicate/procedure `name(Atom,List)` enables the conversion of an atom into a list of character codes.

```
?- name('Hello!',S).
  S = [72, 101, 108, 108, 111, 33]

?- name(A,"Hello!").
  A = 'Hello!'

?- name(A,[112,114,111,108,111,103]).
  A = prolog
```

CONSTRUCTOR

- The constructor `'=..'` is the operator that enables the construction of arbitrary Prolog structures.
- The head of the list becomes a predicate/functor of the structure with the arguments in the tail of the list.

```
?- X =.. [head,o2,o3,o4].
  X = head(o2, o3, o4)

?- functor(o1,o2,o3) =.. X.
  X = [functor,o1,o2,o3]
```

- An example of the use of the constructor:

```
?- X =.. [address,'Slovenska',5],
   Y =.. [person,'David',X].

  X = adress('Slovenska',5)
  Y = person('David',address('Slovenska',5))
```

Unification in Prolog

- The way in which Prolog matches two terms is called unification.
- When two terms are matched it is checked if they can be made to represent the same structure.
- For example, the Prolog knowledge base contains the single clause

```
mother(linda,david).
```

and one gives the query

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

- One would expect `X` to be instantiated to `linda` and `Y` to `david` when the query succeeds.
- It is said that the term `mother(X,Y)` unifies with the term `mother(linda,david)` with `X` bound to `linda` and `Y` bound to `david`.

Unification

- The unification algorithm in Prolog follows the following rules:
  - Two constants unifies if they are identical.
  - Two structures unifies if they have the same functor (predicate) and arity, and if all their corresponding arguments unify as well.
  - A (non-instantiated) variable can unify and then instantiate to any term in both directions.
  - Two (non-instantiated) variables unify and become an identical variable.
- The operator = is the operator that triggers the unification of the two terms on its each side.

UNIFICATION

- For example, the following terms unify and the variables are instantiated as it is shown below.

```
?- X = 1 + 2.
   X = 1 + 2

?- X + Y = 1 + 2.          % +(X,Y) = +(1,2).
   X = 1
   Y = 2

?- f(X) = f(g(1)).
   X = g(1)

?- X = f(X).
   X = f(f(f(f(f(f(f(f( .... ⌛
```

Prolog Unification

- How the following terms are unified?

```
?- date(D,M,1997) = date(D1,june,Y1).
  D = D1 M = june Y1 = 1997

?- triangle(t(1,1),A,t(2,3)) = triangle (X,t(4,Y),t(2,Y)).
  Y = 3 X = t(1, 1) A = t(4, 3)

?- X = [a,b,c].
  X = [a,b,c].

?- [a,b|X] = [a,b,c].
  X = [c] % Because [a,b,c] = [a,b|[c]].

?- [X|[Y]] = [a,b,c].
  false % Because [a,b,c] = [a|[b,c]] in Y != b,c

?- [X|Y] = [a].
  X = a Y = []              % Because [a] = [a|[]]

?- [2|X] = X.
  X = [2,2,2,2,2, ... ⌛  % Because X = [2|[2|[2 ...
```

Arithmetic in Prolog

- Prolog is not the programming language of choice for carrying out heavy-duty mathematics, however, provide arithmetical capabilities.
- The operator `'is'` is the operator that triggers the evaluation of the right-hand side arithmetic terms (expressions).
- The usual numbers and built-in fundamental mathematical functions may appear in the arithmetic expressions, e.g.,

```
?- X = 1 + 2.
  X = 1 + 2

?- X is 1 + 2.
  X = 3

?- 3 = 1 + 2.
  false

?- 3 is 1 + 2.
  true
```

Arithmetic in Prolog

- Other built-in Prolog arithmetic infix operators are:

Operator	Meaning
>	greater than
<	less than
<=	greater than or equal to
>=	less than or equal to
=:=	equal values
=/=	not equal values

- The operator =:= must be distinguished from the operator == that is used for comparing non-arithmetic terms.

```
?- 2*3 =:= 7-1.
  true

?- 2*3 >= 7-1.
  true

?- X == 1 + 2.
  false              % Non-arithmetic comparison!
```

Arithmetic in Prolog

- What are the results of the following queries?

```
?- 2*3 = 7-1.
  false        % *(2,3) does not unify to -(7,1)!

?- 2*3 is 7-1.
  false        % *(2,3) does not unify to 6!

?- 6 is 7-1.
  true

X is 17 * 3 - 5 + log(1.0).
  X = 46

?- 5 is X - 2.
  [WARNING: Unbound variable in arithmetic expression]

?- 15 is 3 * 5.
  true
```

Arithmetic in Prolog

- What are the results of the following queries?

```
?- X = 1 + 2.
  X = 1 + 2

?- X =:= 1 + 2.
  false                % X is not instantiated!

?- X = 1 + 2, 1 + 2 * 3 = X * 3.
  false                % 1 + 2 * 3 = (1 + 2) * 3

?- X = 5 + 3, Y is 2 * X.
  X = 5 + 3 Y = 16

?- X = a + Y, Y = b + Z, Z = c + d.
  Z = c + d
  Y = b + (c + d)
  X = a + (b + (c + d))
```

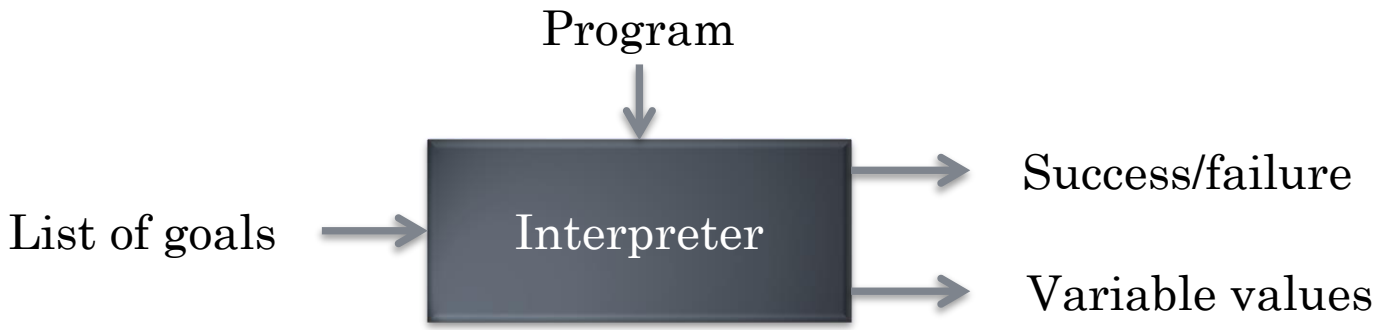
The Scope of variables and Clauses

- The scope of a variable name is the clause in which it occurs - there are no "global" variables.
- All procedures are global.
- The order of how procedures are defined is not relevant, important is the order of the clauses within procedures.
- The anonymous variable is special and it is local to itself, e.g.,

```
address(Name,Street) :-
      person(Name,_,_,Street).

?- child(_,david).   % Has David a child?

?- child(_,_).       % Has somebody a child?

?- plays(_,guitar),plays(_,piano).
```

Prolog Interpreter System

- The inputs to a Prolog interpreter are: a list of goals (a query) and a program with defined procedures, rules and facts.



- A successful accomplishment of the goals includes the values of the variables from the query that have been successfully instantiated.
- The values of the anonymous variables are not printed out.
- The interpreter tries to accomplish the goals in the input query in accordance with specific inference rules.

Prolog Interpreter Rules

- A Prolog program is first consulted and the facts and rules in the program are loaded into the knowledge database.
- The interpreter considers the input goals (query) from left to right
- It searches the knowledge database from top to bottom to find any clause which head unifies with the considered goal.
- If there is such a clause then its variables (if any) are renamed to preserve the integrity of the program.
- The head of the clause is unified with the goal and its body is added to the beginning of the list of goals.

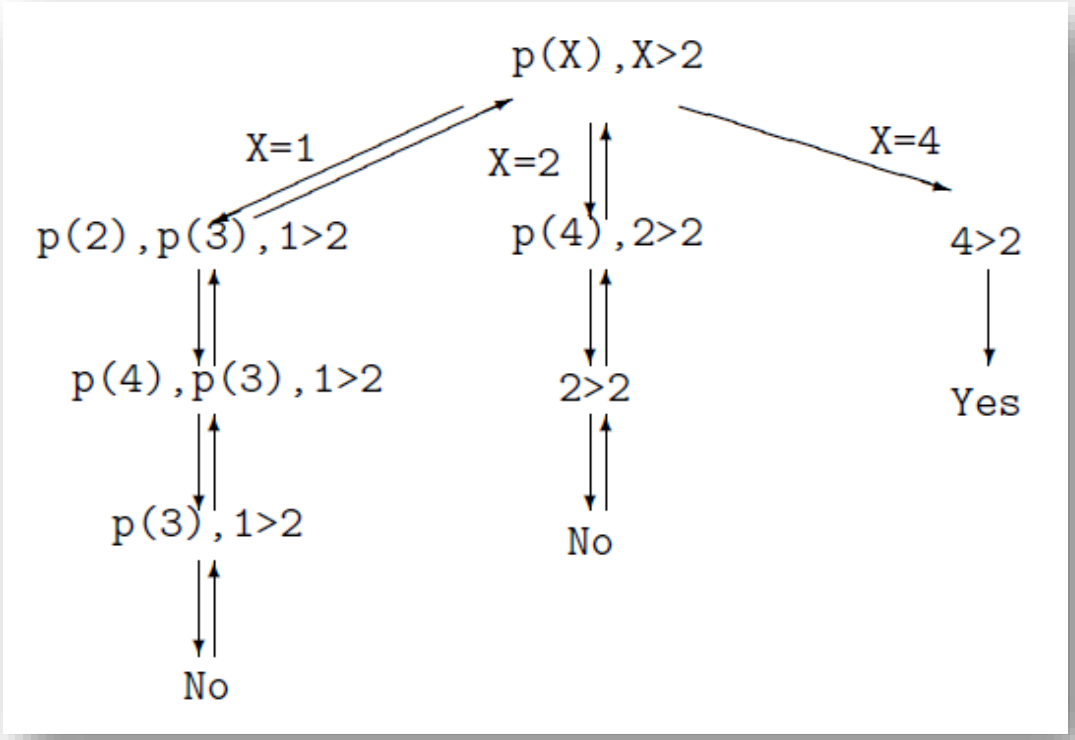
Prolog Interpreter Rules

- If the interpreter fails to accomplish the goal then it returns to the previous goal that has already been unified and tries to unify it again with the next clauses in the program.
- If the first goal in the list of goals cannot be unified then the interpreter returns false.
- If the list of goals is empty then it returns true as well as the instantiated variables from the input query.
- If the first goal can always be unified, but new and new goals are added to the list then the interpreter can loop endlessly.

```
work :- work.

?- work.
```

- The search process of the Prolog interpreter can be seen as a depth-first search.

```
p(1) :- p(2),p(3).      %%%%%%%%%%%%%%%%%%%%%%%%
p(2) :- p(4).   % Program (procedure)! %
p(4).                   %%%%%%%%%%%%%%%%%%%%%%%%

?- consult('program.pl').      % Consulting program

?- p(X),X>2.                   % Query
```



Tracing Prolog Programs

- The execution of Prolog programs can be traced in order to find semantic errors in the program.
- The Prolog built-in procedures `trace` and `spy` can be used for this purpose.

Tracing Example

```
child(david,linda).
child(nancy,linda).
parent(X,Y) :- child(Y,X).

?- spy(parent/2),trace,parent(linda,X).
   * Call: parent(linda, G1248) ?
       Call: child(G1248, linda) ?
       Exit: child(david, linda) ?
   * Exit: parent(linda, david) ?
  X = david ;
       Redo: child(G1248, linda) ?
       Exit: child(nancy, linda) ?
   * Exit: parent(linda, nancy) ?
  X = nancy ;
No
```

Tracing Example

```
child(david,linda).
child(nancy,linda).
parent(X,Y) :- child(Y,X).

?- spy(parent/2),trace,parent(james,X).
    * Call: parent(james, G1272) ?
        Call: child(G1272, james) ?
        Fail: child(G1272, james) ?
    * Redo: parent(james, G1272) ?
    * Fail: parent(james, G1272) ?
No
```

Programming Tips and Examples

- Many programming tasks involve repeated performance of some operation either over a whole data-structure, or until a certain point is reached.
- Such tasks are in Prolog typically implemented by recursions.
- In a recursive procedure, some boundary/stopping rules are first defined following by the rules that refer to themselves, e.g.,

```
on_route(milan).

on_route(Place1):-
    move(Place1,Mean,Place2),
    on_route(Place2).

move(home,bus,ljubljana).
move(ljubljana,train,trieste).
move(trieste,plane,milan).
```

Prolog Recursions

```
fak(0,1).

fak(N,F) :-                 %%%%%%%%%%%%%%%%%%%%%%%%%%
      N > 0,                % Factorial Calculator! %
      N1 is N - 1,          %%%%%%%%%%%%%%%%%%%%%%%%%%
      fak(N1,F1),
      F is N*F1.

?- fak(2,F).                % fak(+N,-F)/2
```

input output arity

```
work :-                                 %%%%%%%%%%%%%%%%
      write('Your command: '),          % Interactive  %
      read(Command),                    % Recursion    %
      (Command = terminate;             %%%%%%%%%%%%%%%%
      execute(Command), work).
```

Interactive Recursion Example

```
work :-
        write('Your command: '),
        read(Command),
        decide(Command).

decide(terminate).
decide(Command) :-
        execute(Command),
        work.

execute(Command) :-
        write('You typed: '),
        write(Command),
        name(NL,[10]), write(NL).
```

Recursions on Lists

```
% Definition delete(+Element,+List1,-List2).
% Procedure delete/3 deletes an element from the list.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

delete(_,[],[]).
delete(E,[E|T],T).
delete(E,[H|T],[H|T1]) :-
       delete(E,T,T1).

?- delete(1,[0,1,2],S).
```

```
% Definition not_element(?Element,+List).
% Procedure not_element/2 checks
% if an element is not on the list.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

not_element(_,[]).
not_element(X,[H|T]) :-
       X \== H,
       not_element(X,T).

?- not_element(4,[0,1,2]).
```

Automatic Backtracking Control

- Automatic backtracking that is part of Prolog's deduction search process is not always efficient, as time can be wasted by exploring possibilities that lead nowhere.
- A built-in Prolog predicate ! (the exclamation mark), called cut, offers a direct way of exercising control over the way Prolog searches for solutions.
- The cut ! is a goal which always succeeds, but cannot be backtracked again.
- It prevents the goal in the head of a prolog clause to be satisfied more than once during the search process.
- The cut should be used sparingly, as there is a temptation to insert cuts experimentally into code that is not working correctly.

Automatic Backtracking Control

- If Prolog finds a cut in a rule, it will not backtrack on the choices it has made.
- For instance, if it has chosen `element` for the variable `X` and encounters a cut, Prolog will consider `element` as the only option for `X`, even if there are other possibilities in the knowledge database.

```
% Definition element(?Element,+List).
% Procedure element/2 checks the presence
% of an element on the list.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

element(X,[X|_]) :- !.
element(X,[_|T]) :-
       element(X,T).

?- element(f(X),[f(1),f(2),f(3)]).
  X = 1;
  No
```

Bi-directionality of arguments

- Most predicates in Prolog have arguments that can be input, output, or input/output, depending how the predicate is used.
- This mostly very useful characteristic is sometimes unwanted.

```
element(X,[X|_]).
element(X,[_|T]) :-
      element(X,T).

?- element(a,[b,c,X,d]).
      X = a
```

```
element(X,[X1|_]) :- X == X1.
element(X,[_|T]) :-
      element(X,T).

?- element(a,[b,c,X,d]).
No.
```

Asserting and Retracting Clauses

- The procedure `assert` adds a clause to the Prolog program (knowledge base).
- The procedure `retract` removes a clause to the Prolog program.
- If the argument of the `retract` procedure contains a variable then the first clause that unifies with the argument is removed from the program.

```
?- assert((pleasant :- sunny)), assert(sunny).
  Yes

?- pleasant.
  Yes

?- retract(child(linda,X)).
  X = david;
  No
```

Input/Output and Files

```
?- consult(File).    % Consulting the program in a file
?- open/4.           % Opening a file
?- close(File).      % Closing a file

?- see(File).   % Redirecting the input to a file
?- see(user).   % Redirecting the input to the user
?- seen.        % Closing and redirecting the input to the user
?- seeing(X).   % Information about the input

?- tell(File). % Redirecting the output to a file
?- tell(user). % Redirecting the output to the user
?- told.       % Closing and redirecting the output to the user
?- telling(X). % Information about the output

?- get(C).    % Unifying of C with the input character
?- put(C).    % Writing a character C to the output
?- read(I).   % Unifying I with the input line
?- write(I).  % Writing I to the output
```

An Input/Output Interaction Example

```
find_parent :-
        write('Whose parent are you looking for?'),
        read(Child),
        parent(Parent,Child),
        write('The parent of ',write(Child),
        write(' is ',write(Parent),
        write('!'),nl.
```

References and Further Readings

- Prolog Programming for Artificial Intelligence (4th Edition)(Bratko, 2011).
- http://www.swi-prolog.org/
- http://www.learnprolognow.org/

Questions

- What are the main differences between Prolog and other programing languages?
- Give an example of a Horn clause in Prolog
- Describe the key features of the Prolog syntax.
- What defines a procedure in a Prolog program?
- Convert an example of a natural language statement into a Prolog clause.
- How does Prolog answer questions?
- What are the inference rules of the Prolog interpreter system?