Prolog Introduction

Console Commands

	command	Description
1	`consult(”filename”).`	load file
2	`make.`	load file
3	`reload.`	reload file
4	`trace.`	debugger
5	`notrace.`	quit debugger
6	`apropos(command_name).`	help for command
7	`;`	get next result

Trace Debug Commands

	command	Description

	command	Description
1	`u`	up (sale di un livello nella ricorsione, esce dalla chiamata)
2	`s`	skip (non scende nell'albero di ricorsione)
3	`a`	abort execution
4	`h`	help
5	`e`	exit
6	`enter`	go into (scende nelle sottochiamate)

Operators

Logic Operators

Operator	Meaning
`X > Y`	X is greater than Y
`X < Y`	X is less than Y
`X >= Y`	X is greater than or equal to Y
`X =< Y`	X is less than or equal to Y
`X =:= Y`	the X and Y values are equal
`X =\= Y`	the X and Y values are not equal

Arithmetic Operators

Operator	Meaning
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`**`	Power
`//`	Integer Division
`mod`	Modulus

Loops

In place of for/while loops Prolog only uses recursion to execute code multiple times.

```
% caso base
count_to_10(10) :-
  write(10),
  nl.

% caso generale
count_to_10(X) :-
  write(X),
  nl,
  Y is X + 1,
  count_to_10(Y).
```

Decision Making

Decision making in prolog is done via conditions placed inside the predicates body, if the condition on the constants evaluates to true then the execution is continued, otherwise prolog looks for another branch of execution that matches the conditions.

```
% If-Then-Else statement

gt(X,Y) :-
  X >= Y,
  write('X is greater or equal').
gt(X,Y) :-
  X < Y,
  write('X is smaller').


% If-Elif-Else statement

gte(X,Y) :-
  X > Y,
  write('X is greater').
gte(X,Y) :-
  X =:= Y,
  write('X and Y are same').
gte(X,Y) :-
  X < Y,
  write('X is smaller').
```

AND / OR

Conjunction

Conjunction (AND logic) can be implemented using the comma (,) operator. So two predicates separated by comma are joined with AND statement. Suppose we have a predicate, **parent(jhon, bob)**, which means “Jhon is parent of Bob”, and another predicate, **male(jhon)**, which means “Jhon is male”. So we can make another predicate that **father(jhon,bob)**, which means “Jhon is father of Bob”. We can define predicate **father**, when he is parent **AND** he is male.

Disjunction

Disjunction (OR logic) can be implemented using the semi-colon (;) operator. So two predicates separated by semi-colon are joined with OR statement. Suppose we have a predicate, **father(jhon, bob)**. This tells that “Jhon is father of Bob”, and another predicate, **mother(lili,bob)**, this tells that “lily is mother of bob”. If we create another predicate as **child()**, this will be true when **father(jhon, bob)** is true **OR** **mother(lili,bob)** is true.

```
parent(jhon,bob).
parent(lili,bob).

male(jhon).
female(lili).

% Conjunction Logic
father(X,Y) :- parent(X,Y), male(X).
mother(X,Y) :- parent(X,Y), female(X).

% Disjunction Logic
child_of(X,Y) :- father(X,Y); mother(X,Y).
```

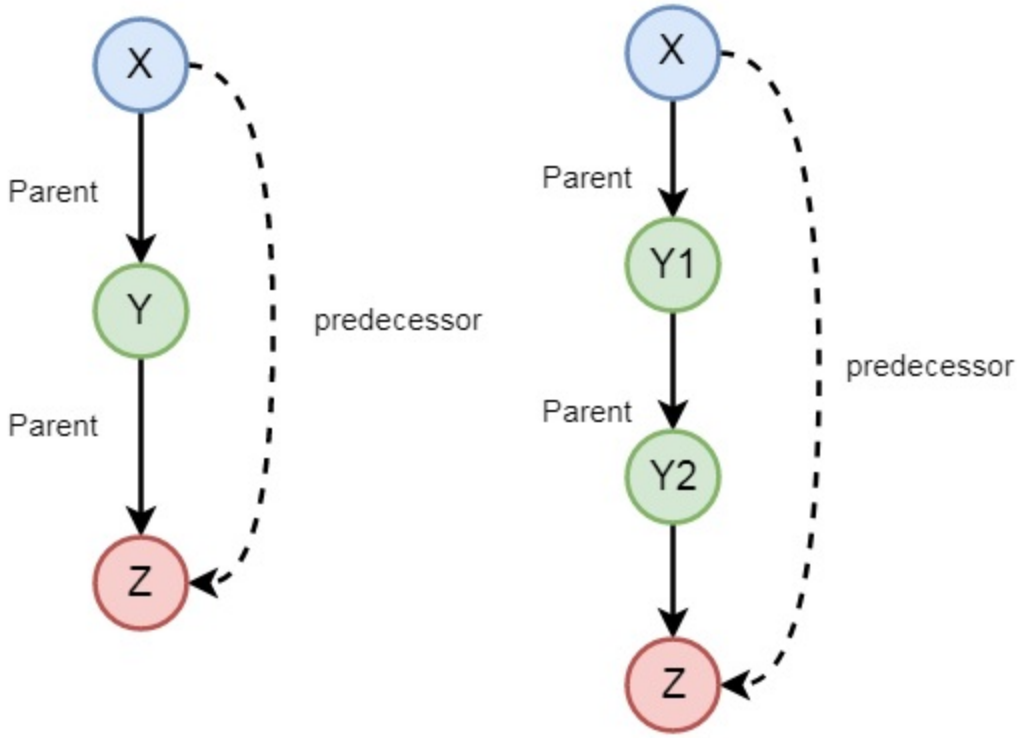
List

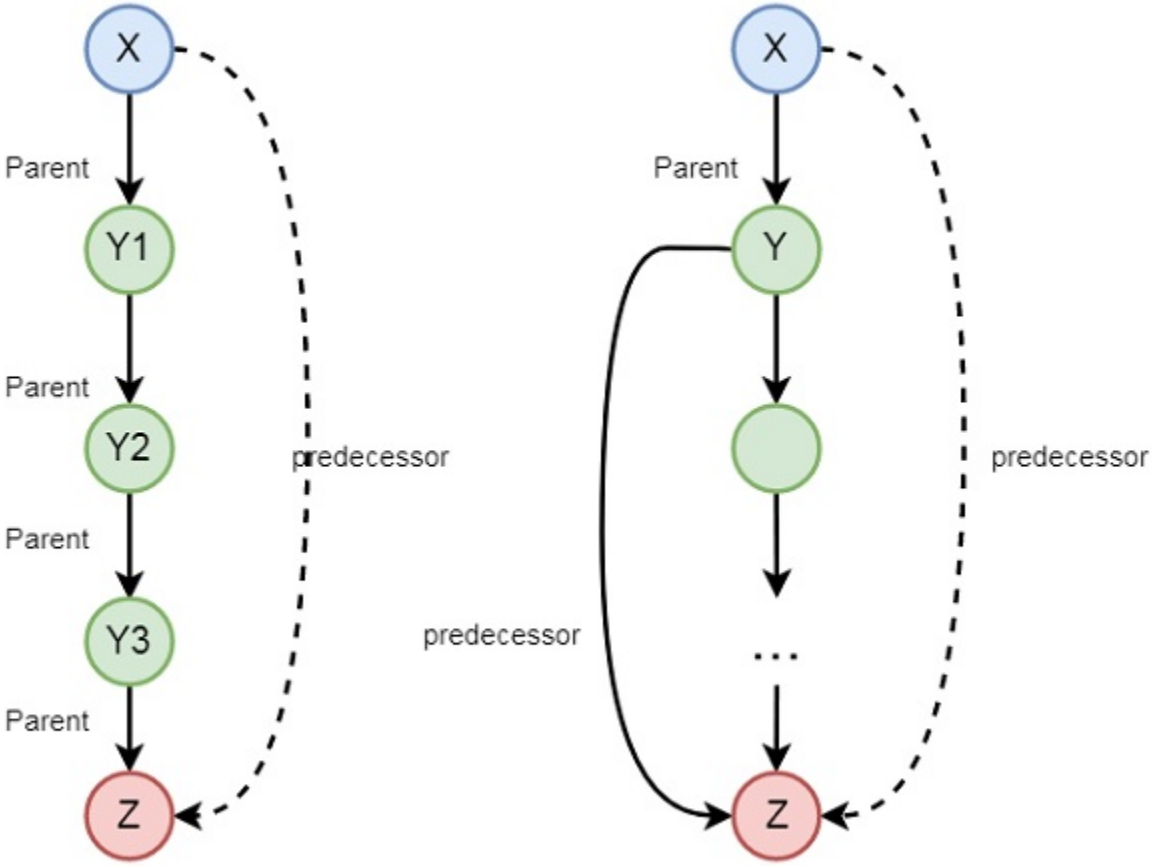
https://www.tutorialspoint.com/prolog/prolog_lists.htm

Recursion

Recursion is a technique in which one predicate uses itself (may be with some other predicates) to find the truth value.

There may be some other examples also, so let us see one family example. So if we want to express the predecessor logic, that can be expressed using the following diagram





So we can understand the predecessor relationship is recursive.

We can express this relationship using the following syntax:

- predecessor(X, Z) :- parent(X, Z).
- predecessor(X, Z) :- parent(X, Y), predecessor(Y, Z).

Negation via Fail

Here we will perform failure when condition does not satisfy. Suppose we have a statement, “Mary likes all animals but snakes”, we will express this in Prolog.

It would be very easy and straight forward, if the statement is “Mary likes all animals”. In that case we can write “Mary likes X if X is an animal”. And in prolog we can write this statement as, likes(mary, X) := animal(X).

In prolog we can write this as:

- likes(mary,X) :- snake(X), !, fail.
- likes(mary, X) :- animal(X).

The ‘fail’ statement causes the failure. Now let us see how it works in Prolog.

```
animal(dog).
animal(cat).
animal(elephant).
animal(tiger).
animal(cobra).
animal(python).

snake(cobra).
snake(python).

likes(mary, X) :- snake(X), !, fail.
likes(mary, X) :- animal(X).
```

built-in-predicates

Predicate	Description
`var(X)`	succeeds if X is currently an un-instantiated variable.
`novar(X)`	succeeds if X is not a variable, or already instantiated
`atom(X)`	is true if X currently stands for an atom
`number(X)`	is true if X currently stands for a number
`integer(X)`	is true if X currently stands for an integer
`float(X)`	is true if X currently stands for a real number.
`atomic(X)`	is true if X currently stands for a number or an atom.
`compound(X)`	is true if X currently stands for a structure.
`ground(X)`	succeeds if X does not contain any un-instantiated variables.

```
| ?- var(X).

yes
```

```
| ?- X = 5, var(X).

no
| ?- var([X]).

no
| ?-
```

Mathematical Predicates

Predicates	Description
`random(L,H,X).`	Get random value between L and H
`between(L,H,X).`	Get all values between L and H
`succ(X,Y).`	Add 1 and assign it to X
`abs(X).`	Get absolute value of X
`max(X,Y).`	Get largest value between X and Y
`min(X,Y).`	Get smallest value between X and Y
`round(X).`	Round a value near to X
`truncate(X).`	Convert float to integer, delete the fractional part
`floor(X).`	Round down
`ceiling(X).`	Round up
`sqrt(X).`	Square root

Tree Data Structure

https://www.tutorialspoint.com/prolog/prolog_tree_data_structure.htm

Exercises

https://www.tutorialspoint.com/prolog/prolog_basic_programs.htm