CS 420 Advanced Programming Languages
Fall Semester, 2022
Doc 20 Prolog 2
Nov 8, 2022

SWI Prolog

https://www.swi-prolog.org

Free download for

- MacOS
- Windows
- Link

More Consult

?- consult(prime).
  Load and compile prime.pl

?- [prime].
  Short cut for consult(prime)

?- make.
  load and recompiles files that have changed

?- edit(is_prime).
  Opens an editor on the file(s) containing is_prime

?- listing(is_prime).
  Shows the current definition of is_prime

abolish

?- abolish(is_prime/1).

Removes is_prime/1 from the interpreter

modules

```
% is_prime(P) :- P is a prime number
%    (integer) (+)

:- module(prime,[is_prime/1]).

is_prime(2).
is_prime(3).
is_prime(P) :- integer(P), P > 3, P mod 2 =\= 0, \+ has_factor(P,3).

% has_factor(N,L) :- N has an odd factor F >= L.
%    (integer, integer) (+,+)

has_factor(N,L) :- N mod L =:= 0.
has_factor(N,L) :- L * L < N, L2 is L + 2, has_factor(N,L2).
```

using has_factor

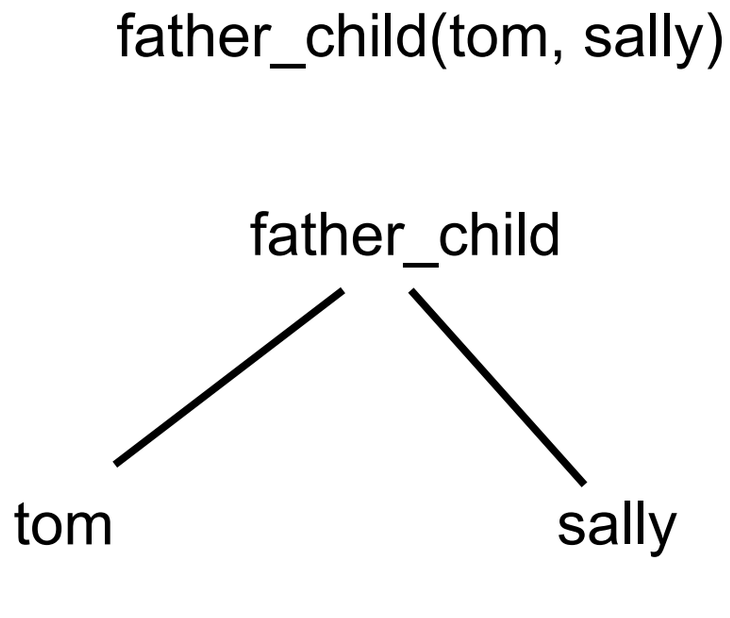
```
?- [prime].
% prime compiled into prime 0.00 sec, 1,756 bytes
true.

?- is_prime(4).
false.

?- has_factor(10,3).
Correct to: "prime:has_factor(10, 3)"? yes
true .

?-
```

Some IO

read & write

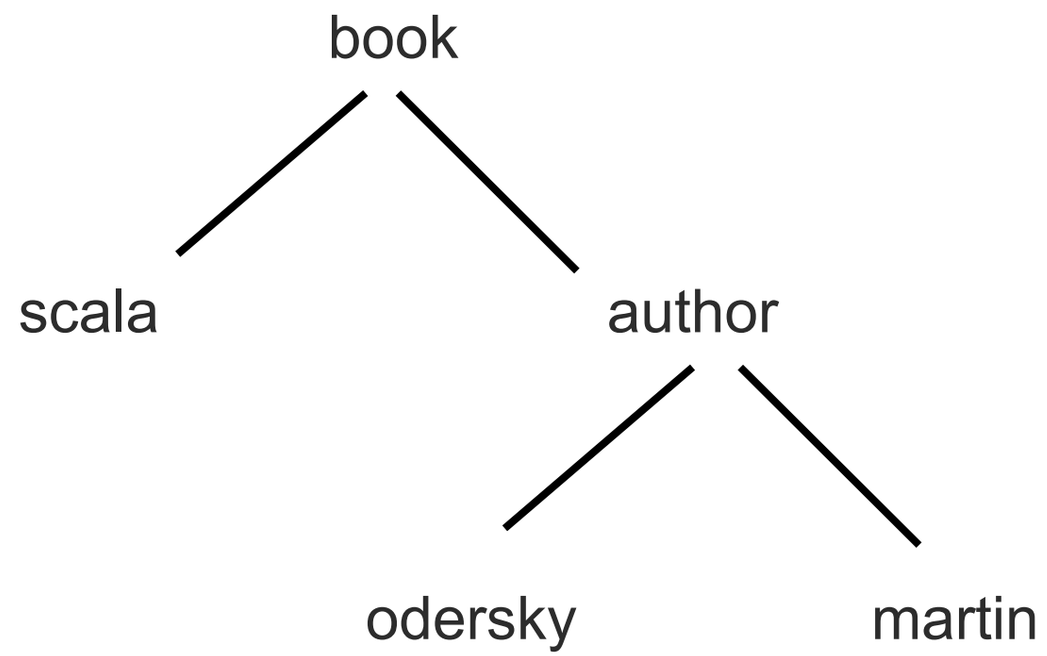
```
?- write(cat).
cat
true.

?- write(cat),nl,write(dog).
cat
dog
true.

?- read(X).
|: mat.
X = mat.

?- read(X), write(X).
|: sat.
sat
X = sat.
```

Structure



Complex Structure

book(scala,author(odersky,martin))



Some Book Facts

```
book(scala,author(odersky,martin),publisher(artima)).
book(stripes,author(daoud,frederic),publisher(pragmaticProgramers)).
book(masteringDojo,author(riecke,craig),publisher(pragmaticProgramers)).
book(programingGroovy,author(subramaniam,venkat),publisher(pragmaticProgramers)).
```

Book Questions

?- book(scala,X,Y).
X = author(odersky, martin),
Y = publisher(artima).

?- book(scala,X,_).
X = author(odersky, martin).

?- book(Title,author(odersky,_),_publisher).
Title = scala,
_publisher = publisher(artima) .

?- book(Title,_,publisher(pragmaticProgramers)).
Title = stripes ;
Title = masteringDojo ;
Title = programingGroovy.

?-

Matching

Atom match themselves

Variables match anything

_ is don't care variable

Don't Care Example

?- book(scala,X,X).
false.

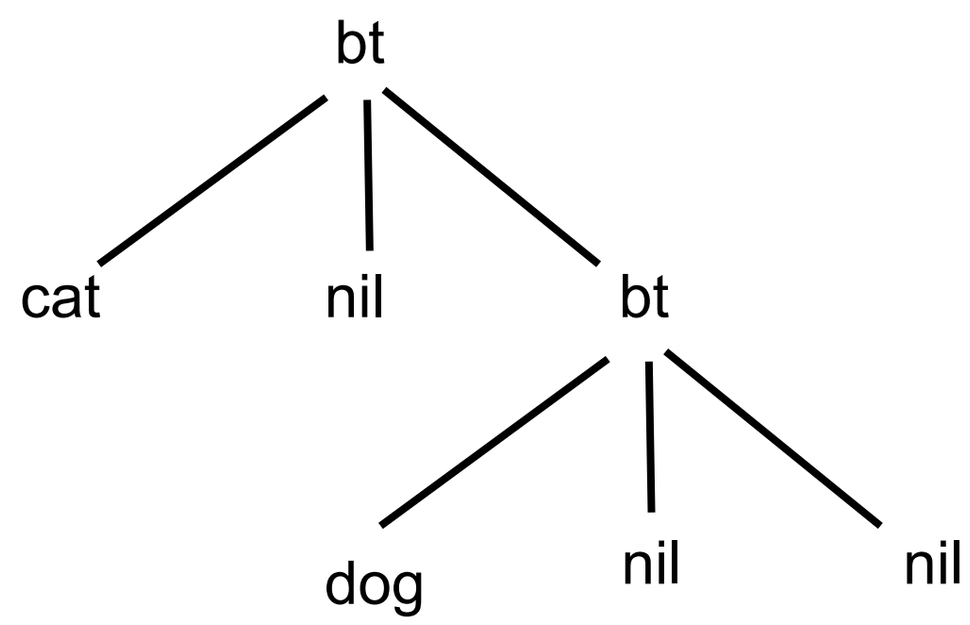
?- book(scala,_,_).
true.

?-

Binary Tree

bt(value,leftTree,rightTree)

bt(cat,nil, bt(dog,nil,nil))



Binary Tree Example

```
istree(nil).
istree(bt(_,L,R)) :- istree(L), istree(R).
% bt(value,leftTree,rightTree)
```

```
?- istree( bt(a,nil,nil) ).
true.

?- istree( bt(a, bt(c,nil,nil), bt(d,nil,nil))).
true.

?- istree( bt(a, bt(c,nil,nil), bt(d,nil, bt(e,nil,nil)))).
```

Lists

.(head,tail)	[head|tail]	[] empty list
.(a,[])	[a]	tails of lists are lists
.(a,.(b,[])	[a,b]	
.(a,.(b,.(c,[])))	[a,b,c]	

Some Lists

[this, is, a, simple, list]

[this, list, has, [a, nested, list], in, it]

[one, [can, nest, [lists, as, deep], as], one, [needs, to]]

Heads and Tails

[Head | Tail]

	Head	Tail
[Head | Tail] = [a, b, c]	a	[b, c]
[Head | Tail] = [the, [cat, sat]]	the	[[cat, sat]]
[Head | Tail] = [the, [dog, sat], down]	the	[[dog, sat], down]
[Head | Tail] = [[the, man], sat, down]	[the, man]	[sat, down]

Some Matches

[X, Y, Z] = [the, red, hat]	X= the, Y= red, Z = hat
[the, red, hat] = [X, Y, Z]	X= the, Y= red, Z = hat
[cat] = [X | Y]	X = cat, Y = []
[X, Y | Z] = [the, red, hat]	X= the, Y= red, Z = [hat]
[[the, Y] | Z] = [[X, hat], [is, here]]	X = the Y= hat Z = [[is, here]]

Contains

```
% contains(element, List)  true if element is in the list

contains(X, [X|_]).
contains(X, [_| Y]) :- contains(X,Y).
```

```
?- contains(10,[1,2,3,4,5,6,7,8,9,10]).
true ;
false.

?- contains(X,[1,2,3,4,5,6,7,8,9,10]).
X = 1 ;
X = 2 ;
X = 3 ;
```

Sum

```
sum([],Sum) :- Sum = 0.
sum([H|T],Sum) :- sum(T,SublistSum), Sum is SublistSum + H.
```

```
?- sum([1,10,2],S).
S = 13.
```

Append

```
append([],L, L).
append([X|L1], L2, [X| L3]) :- append(L1, L2, L3).
```

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

?- append(X, [c, d], [a, b, c, d]).
X = [a, b] ;
false.

?- append([a, b], [c, d], [a, b, c, d]).
true.
```

Problems

Find the last element of a list

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

Find the K'th element of a list

?- elementAt(X, [a, b, c, d], 3).
X = c

Find the length of a list

Reverse a list

Problem

```
mother_child(susan, sally).
mother_child(susan, matt).

father_child(tom, sally).
father_child(tom, erica).
father_child(tom, pete).
father_child(mike, tom).
```

```
sibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

Backtracking

```
mother_child(susan, sally).
mother_child(susan, matt).

father_child(tom, sally).
father_child(tom, erica).
father_child(tom, pete).
father_child(mike, tom).
```

```
sibling(X, Y)  :- parent_child(Z, X), parent_child(Z, Y), \+ X = Y.

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y)
```

Backtracking & Database

Each time a rule/fact matches Prolog keeps track of where in the database the match occurred

Variables

C/C++/Java variables

- Point to a memory location
- Can change the value of a variable
- X = X + 1

Mathematical Variables

- Represent value(s) that make equations true
- X = X + 1 has no solution

Prolog variables are like mathematical variables

Backtracking & Unification

In backtracking Prolog tries out various values for variables

Cut !

```
mother_child(susan, sally).
mother_child(susan, matt).

father_child(tom, sally).
father_child(tom, erica).
father_child(tom, pete).
father_child(mike, tom).
```

```
sibling(X, Y) :- parent_child(Z, X), !, parent_child(Z, Y), \+ X = Y.

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

Cut !

Once a cut is reach in a rule

Prolog will not try to re-satisfy any goal between parent goal and cut

Cuts

Reduce the number of paths searched

Reduce bookkeeping needed for backtracking

What Happens Here?

```
mother_child(susan, sally).
mother_child(susan, matt).

father_child(tom, sally).
father_child(tom, erica).
father_child(tom, pete).
father_child(mike, tom).
```

```
related(X,Y) :- father_child(_, X), sibling(X,Y).

sibling(X, Y) :- parent_child(Z, X), !, parent_child(Z, Y), \+ X = Y.

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

Common Reasons for using Cut

Tell Prolog that it has found the correct rule

Tell Prolog to fail a goal without trying other solutions (! , fail)

Tell Prolog it has found a correct solution and stop looking for more

What Happens Here?

```
sumTo(1,1).
sumTo(N,Sum) :-
   N1 is N -1,
   sumTo(N1,Sum2),
   Sum is Sum2 + N.
```

```
?- sumTo(5,X).
X = 15 ;
```

```
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 0.9Gb, global: 77.8Mb, trail: 0K
ERROR:   Stack depth: 10,189,004, last-call: 0%, Choice po
ERROR:   In:
ERROR:     [10,189,004] user:sumTo(-10188989, _2040432
ERROR:     [10,189,003] user:sumTo(-10188988, _2040434
ERROR:     [10,189,002] user:sumTo(-10188987, _2040436
ERROR:     [10,189,001] user:sumTo(-10188986, _2040438
ERROR:     [10,189,000] user:sumTo(-10188985, _2040440
ERROR:
ERROR: Use the --stack_limit=size[KMG] command line opt
ERROR: ?- set_prolog_flag(stack_limit, 2_147_483_648). to
```

Found the correct rule - so stop

```
sumTo(1,1) :- !.
sumTo(N,Sum) :-
    N1 is N -1,
    sumTo(N1,Sum2),
    Sum is Sum2 + N.
```

```
?- sumTo(5,X).
X = 15.
```

Replacing ! with \+

```
sumTo(1,1).
sumTo(N,Sum) :-
   \+(N = 1),
   N1 is N -1,
   sumTo(N1,Sum2),
   Sum is Sum2 + N.
```

!, fail

```
average_taxpayer(X) :- foreigner(X), !, fail.

average_taxpayer(X) :-
    spouse(X, Y),
    gross_income(Y, Income),
    Income > 300000,
    !, fail.

average_taxpayer(X) :-
    gross_income(X, Income),
    20000 < Income, Income < 200000.
```

Replace !, fail with \+

```
average_taxpayer(X) :-
    \+ foreigner(X),
    \+(spouse(X, Y), gross_income(Y, SpouseIncome), SpouseIncome > 300000),
    gross_income(X, Income),
    20000 < Income, Income < 200000.

gross_income(X,Y) :-
    \+ (receives_pension(X, Pension), Pension < 20000),
    gross_salary(X, Z),
    investment_income(X,W),
    Y is Z + W.
```

Found a Correct Solution - So Stop

```
is_integer(0).
is_integer(X) :- is_integer(Y), X is Y + 1.


divide(Numerator,Denominator, Result) :-
    is_integer(Result),
    Product is Result * Denominator,
    ProductNext is (Result + 1) * Denominator,
    Product =< Numerator, ProductNext > Numerator,
    !.
```