Artificial Intelligence Lab
Computer Science CC13P Study Material

Part-I: Even, Sum, Squares, GCD, Factorial, Fibonacci, ${}^nC_r$

Anupam Pattanayak[1]
Assistant Professor,
Department of Computer Science,
Raja N. L. Khan Women's College (Autonomous),
Midnapore, West Bengal

April 8, 2020

[1]anupam.pk@gmail.com

Contents

1

Prolog Programs - I

In class, we have seen the programs for family tree. Now we will see the programs that have been given in syllabus. we will refer the book by Brakto[1], and the book by Bramer[2] for our programs. We use the *swipl* in Linux. All these programs have been compiled and run in my system.

1.1 Program for Even Number

We will first see how to write a program to check if a number is even. Observe the program given below:

```
% program for checking if  a number is even.
% File Name: even.pl
checkeven(N):- M is N//2, N=:=2*M.
```

Observe the program. We have used % symbol for single line comment. // symbol has been used for division operation. =:= has been used for assignment. *checkeven(N)* function checks is N is even. It first divides N by 2, keeps the result in M, and then it checks if N equals 2×M. Since , is AND operator, the *checkeven(N)* will return *true* if both the operands are true.

Following is the compilation and execution of the procedure *checkeven( )*.

```
$ swipl

?- consult('even.pl').
```

[1]Prolog Programming for Artificial Intelligence by Ivan Bratko
[2]Logic Programming with Prolog by Max Bramer

```
% even.pl compiled 0.00 sec, 2 clauses
true.

?- checkeven(14).
true.

?- checkeven(141).
false.
```

1.2 Program for Sum of Natural Numbers

We will now see the prolog program for computing sum of natural numbers $1 + 2 + 3 + \cdots + N$. Follwing is the prolog program for this.

```
% Sum of Natural numbers 1+2+3+...+N
% File name: sum.pl

sumto(1,1).
sumto(N,S):-N>1,N1 is N-1,sumto(N1,S1),S is S1+N.}
```

See that, a recursive procedure *sumto( )* has been defined. It has two arguments, first one is the value of N. Second one is the value of sum.

To run this program first compile it and then invoke the procedure *sumto( )*. Suppose we want to compute $1+2+3+\cdots+10$. So invoke the procedure *sumto(10,S)* as shown below.

```
?- consult('sum.pl').
% sum.pl compiled 0.00 sec, 3 clauses
true.

?- sumto(10,S).
S = 55
```

Press a · to return to the command prompt.

1.3 Compute Squares of 1, 2, ... , N

We will now see the prolog program for computing sum of squares of 1, 2, $\cdots$, N. See the program given below.

```
% to compute and display squares
% File namee: square.pl

writesquares(1):-write(1),nl.
writesquares(N):-N>1,N1 is N-1,writesquares(N1),Nsq is N*N,write
    (Nsq),nl.
```

See the procedure *write( )*, that has been used for printing. Also observe the use of *nl* to print a *newline*. See the definition of *writesquares( )*.

To run this program first compile it and then invoke the procedure *writesquares( )*. Suppose we want to compute squares of numbers $1, 2, 3, \cdots,$. So invoke the procedure *writesquares(8)* as shown below.

```
?- consult('square.pl').
% square.pl compiled 0.00 sec, 3 clauses
true.

?- writesquares(8).
1
4
9
16
25
36
49
64
true .
```

1.4 Program for GCD

GCD stands for greatest common divisor. It is also known as highest common factor (HCF). Suppose, we need to find GCD of m and n. The recursive definition of GCD is as follows: $GCD(m, n) = m$ if $n = 0$, otherwise, $GCD(m, n) = GCD(n, m\%n)$. Following program shows the code that computes GCD(A,B).

```
% prolog program to compute GCD
% File name: gcd.pl
```

```
hcf(A,0,A).
hcf(A,B,S):-B>0, B1 is A mod B, hcf(B,B1,S1),S is S1.
```

See the procedure *hcf( )*. First executable statement of this program defines the base case: B=0.

To run this program first compile it and then invoke the procedure *hcf( )*. Suppose we want to compute GCD of 75 and 90. So invoke the procedure *hcf(75,90,M)* as shown below.

```
?- consult('gcd.pl').
% gcd.pl compiled 0.00 sec, 1 clauses
true.


?- hcf(75,105,A).
A = 15 .

?- hcf(23,0,A).
A = 23 .

?- hcf(23,105,A).
A = 1 .
```

1.5 Program for Factorial

Factorial of a positive number n is denoted by $n!$. And the formula to compute $n!$ is defined as
$n! = 1 \times 2 \times 3 \times \cdots \times n$.

Following program computes factorial of a number.

```
% prolog program to compute Factorial
% File name: factorial.pl

fact(0,1).
fact(1,1).
fact(N,S):-N>1,N1 is N-1,fact(N1,S1),S is S1*N.
```

Closely observe the procedure *fact( )*. First executable statement of this program defines the base case: N=0.

To run this program first compile it and then invoke the procedure *fact( )*. Compilation of the program and some sample executions of this program are shown below.

```
?- consult('factorial.pl').
% factorial.pl compiled 0.00 sec, 4 clauses
true.

?- fact(5,N).
N = 120 .

?- fact(0,N).
N = 1 .

?- fact(1,N).
N = 1 .
```

1.6 Program for Fibonacci

Fibonacci Series is an important and widely studied sequence of mathematics that has many beautiful applications. N^{th} number of the Fibonacci sequence is defined by the recurrence relation: $F_n = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise} \end{cases}$

Following program computes N^{th} number of Fibonacci series.

```
% prolog program to compute Nth Fibonacci number
% File name: fibonacci.pl

fibo(0,0).
fibo(1,1).
fibo(N,S):-N>1,N1 is N-1,N2 is N-2,fibo(N1,S1),fibo(N2,S2),S is
    S1+S2.
```

Closely look at the procedure *fibo( )*. First executable statement of this program defines the base case: N=0. Second executable statement defines the case for N=1.

To run this program, first compile it and then invoke procedure *fibo( )*. Compilation of the program and some sample executions are shown below.

```
?- consult('fibonacci.pl').
% fibonacci.pl compiled 0.00 sec, 4 clauses
true.

?- fibo(5,N).
N = 5 .

?- fibo(8,N).
N = 21 .

?- fibo(0,N).
N = 0 .

?- fibo(11,N).
N = 89 .
```

1.7 Program for ^nC_r

^nC_r, or *n choose r* is a term hugely used in *permutation and combination* of mathematics. It is defined as,
$^nC_r = \frac{n}{(n-r)!\times r!}$. It can be defined recursively and recursive definition of ^nC_r is as follows:
$^nC_r = {}^nC_{(r-1)} + {}^{(n-1)}C_{(r-1)}$, with base conditions $^nC_0 = 1$, and $^nC_n = 1$.

Following program computes ^nC_r.

```
% prolog program to compute nCr, n Choose r
% File name: ncr.pl

nCr(N,0,1).
nCr(N,N,1).
nCr(N,1,N).
nCr(N,R,S):-R>1,
            N1 is N-1,
            R1 is R-1,
            nCr(N1,R1,S1),
            nCr(N1,R,S2),
            S is S1+S2.
```

Oserve the procedure *nCr( )* very carefully. To run this program first compile it and then invoke the procedure *nCr( )*. Compilation of the program and some sample executions of this program are shown below.

```
?- consult('ncr.pl').
Warning: /home/anupam/RNLKWC/PG/acad/4th sem/prolog/ncr.pl:1:
   Singleton variables: [N]
% ncr.pl compiled 0.00 sec, 5 clauses
true.

?- nCr(5,1,N).
N = 5  .

?- nCr(5,2,N).
N = 10  .

?- nCr(4,2,N).
N = 6  .

?- nCr(4,4,N).
N = 1  .

?- nCr(4,0,N).
N = 1  .
```

Although, there is an warning message after compilation, you can ignore the warning message and go ahead with execution.