**Programming Assignment 2 :** **PROGOL**

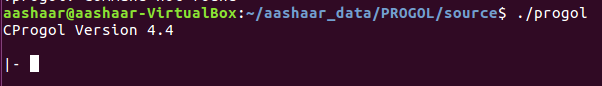
**CS6364**

Submitted by :

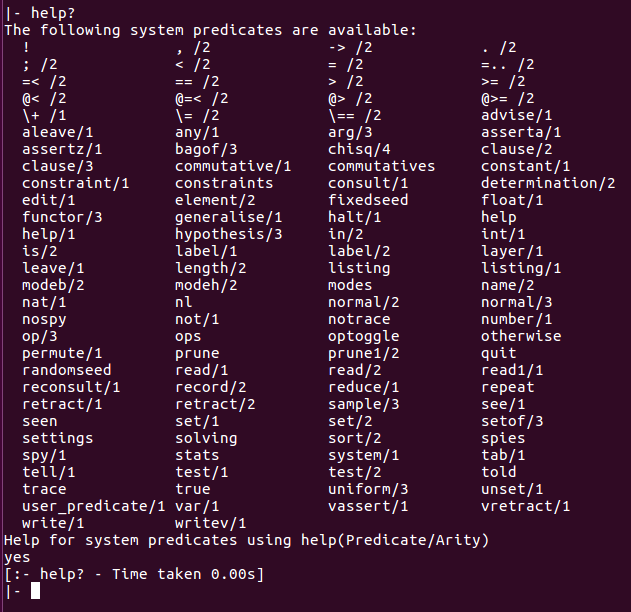
Aashaar Panchalan – adp170630

Manish Biyani

**Ex. 1:**



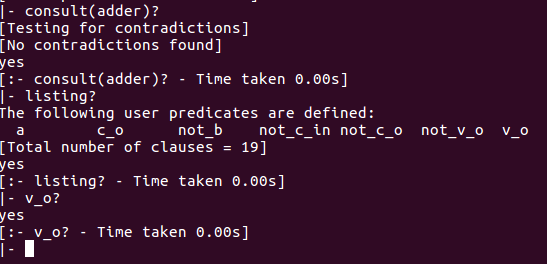
**Ex. 2:**



**Ex. 3:**

Please refer the file ‘adder.pl’ in the codes folder, for the code.

Here’s the screenshot for input state : a, not\_b, not\_c\_in:



**Ex. 4:**

**a)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **Cin** | **Vo** | **Co** | **Progol Screenshot** |
| 0 | 0 | 0 | 0 | 0 |  |
| 0 | 0 | 1 | 1 | 0 |  |
| 0 | 1 | 0 | 1 | 0 |  |
| 0 | 1 | 1 | 0 | 1 |  |
| 1 | 0 | 0 | 1 | 0 |  |
| 1 | 0 | 1 | 0 | 1 |  |
| 1 | 1 | 0 | 0 | 1 |  |
| 1 | 1 | 1 | 1 | 1 |  |

From the truth-table, we have verified that Progol gives correct answers for all the output propositions.

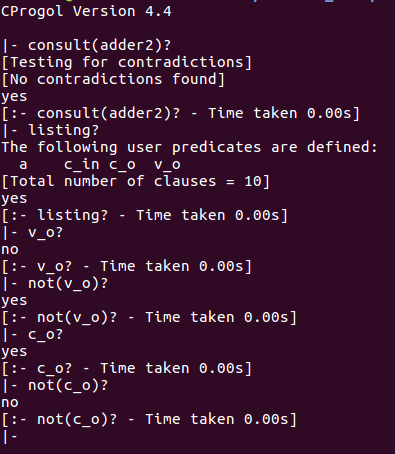
**b)**

Yes, the adder circuit can be represented by fewer clauses by eliminating the negation of the two outputs. If we have the propositions for v\_o and c\_o we don’t need to maintain not\_v\_o and not\_c\_o. if required, they can simply be obtained by negating the v\_o & c\_o respt. Thus we can eliminate 8 clauses and the same adder circuit can be formed using only 8 clauses – 4 each for v\_o and c\_o.

**Ex. 5:**

Please refer the file ‘adder2.pl’ in the codes folder, for the code.

Here’s the screenshot for input state : a, c (since we require not(b) to be true, we haven’t specified ‘b’ in the file so that by closed world assumption it becomes true.):



From the output in the above screenshot, we can see that not\_v\_o and not\_c\_o can be derived from the the v\_o and c\_o respt., simply by negating them due to the closed world assumption.

**b)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **Cin** | **Vo** | **Co** | **Progol Screenshot** |
| 0 | 0 | 0 | 0 | 0 |  |
| 0 | 0 | 1 | 1 | 0 |  |
| 0 | 1 | 0 | 1 | 0 |  |
| 0 | 1 | 1 | 0 | 1 |  |
| 1 | 0 | 0 | 1 | 0 |  |
| 1 | 0 | 1 | 0 | 1 |  |
| 1 | 1 | 0 | 0 | 1 |  |
| 1 | 1 | 1 | 1 | 1 |  |

From the truth-table, we have verified that Progol gives correct answers for all the output propositions.

**Ex. 6:**

1. in\_and(X,S1,S2):-  
    elem(X,S1).

in\_and(X,S1,S2):-  
 elem(X,S2).

1. in\_mult(X,Y,S1,S2):-  
    elem(X,S1).

in\_mult(X,Y,S1,S2):-

elem(X,S2).

1. in\_div(X,S1,S2):-  
    elem(X,S1).

in\_div(X,S1,S2):-

not(elem(X,S2)).

**Ex. 7:**

1. less\_than\_5(0,1).

less\_than\_5(0,2).

less\_than\_5(0,3).

less\_than\_5(0,4).

less\_than\_5(1,2).

less\_than\_5(1,3).

less\_than\_5(1,4).

less\_than\_5(2,3).

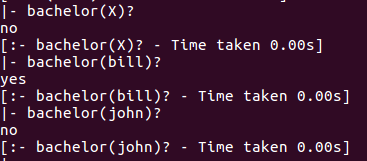
less\_than\_5(2,4).

less\_than\_5(3,4).

1. in\_mult(X,Y,A,B):-  
    elem(X,A),elem(X,B),less\_than\_5(X,Y).

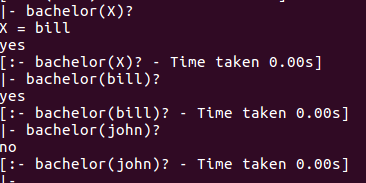
**Ex. 8 :**

1. bachelor(X) returns NO.
2. bachelor(bill) returns YES and bachelor(john) returns NO.  
   Please refer below screenshot for the above two answers.



1. bachelor(X) gives   
    X=Bill  
    Yes.

bachelor(bill) returns YES and bachelor(john) returns NO.  
Please refer below screenshot for the above two answers.



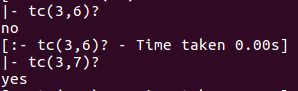
**Ex. 9:**

We added all the predicates to a file named tc.pl (available in codes folder) and changed the definition for each question.

1. **Definition 1:**

tc(3,6)- No

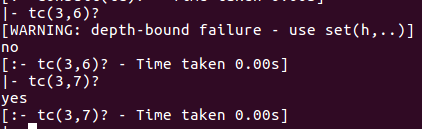
tc(3,7) – Yes



**Definition 2:**

tc(3,6)- No

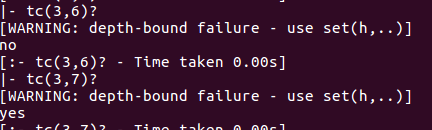
tc(3,7) – Yes



**Definition 3:**

tc(3,6)- No

tc(3,7) – Yes

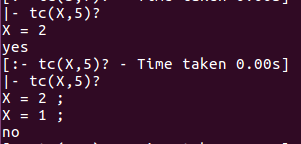


1. **Definition 1:**

tc(X,5)

X = 2 ;

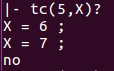
X = 1 ;



tc(5,X)

X = 6 ;

X = 7 ;



tc(X,Y)

X = 1

Y = 2 ;

X = 1

Y = 3 ;

X = 2

Y = 4 ;

X = 2

Y = 5 ;

X = 3

Y = 4 ;

X = 4

Y = 7 ;

X = 5

Y = 6 ;

X = 6

Y = 7 ;

X = 1

Y = 4 ;

X = 1

Y = 5 ;

X = 1

Y = 7 ;

X = 1

Y = 6 ;

X = 1

Y = 7 ;

X = 1

Y = 4 ;

X = 1

Y = 7 ;

X = 2

Y = 7 ;

X = 2

Y = 6 ;

X = 2

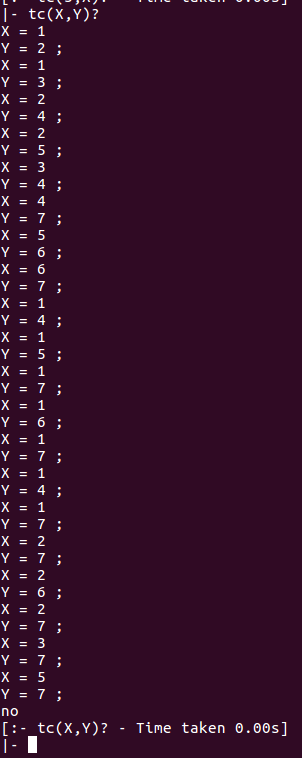
Y = 7 ;

X = 3

Y = 7 ;

X = 5

Y = 7 ;



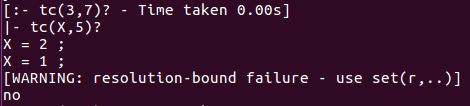
**(Ex.5b conted …)**

**Definition 2:**

tc(X,5)

X = 2;

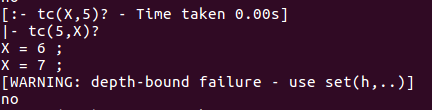
X = 1;



tc(5,X)

X = 6;

X =7;



tc(X,Y)

X = 1

Y = 2 ;

X = 1

Y = 3 ;

X = 2

Y = 4 ;

X = 2

Y = 5 ;

X = 3

Y = 4 ;

X = 4

Y = 7 ;

X = 5

Y = 6 ;

X = 6

Y = 7 ;

X = 1

Y = 4 ;

X = 1

Y = 5 ;

X = 1

Y = 4 ;

X = 2

Y = 7 ;

X = 2

Y = 6 ;

X = 3

Y = 7 ;

X = 5

Y = 7 ;

X = 1

Y = 7 ;

X = 1

Y = 6 ;

X = 1

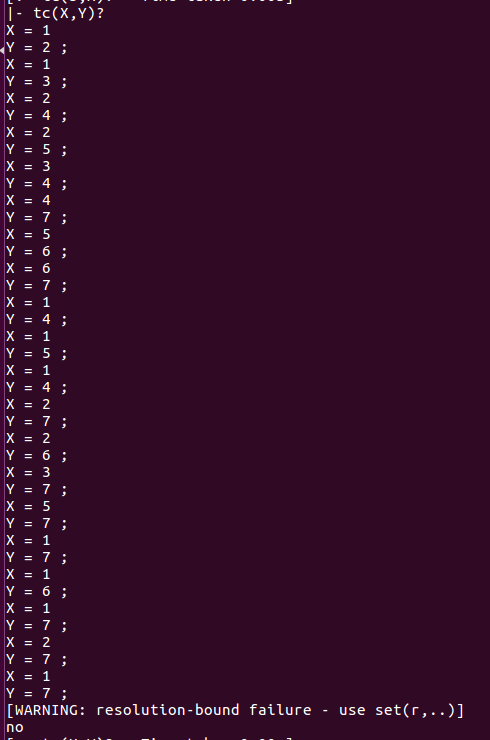
Y = 7 ;

X = 2

Y = 7 ;

X = 1

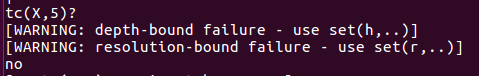
Y = 7 ;



**(Ex.5b conted …)**

**Definition 3:**

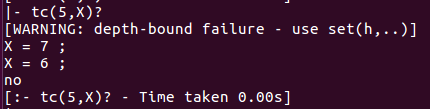
tc(X,5) – None



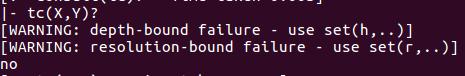
tc(5,X)

X = 6;

X =7;



tc(X,Y) – None



**c)**

X = 2 ;

X = 3 ;

X = 3 ;

X = 4 ;

X = 1 ;

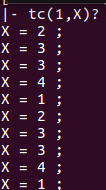
X = 2 ;

X = 3 ;

X = 3 ;

X = 4 ;

X = 1 ;



**Ex. 10:**

**a)** X= 2+3? - Yes

**b)** X is 2+3?

X = 5?

Yes

