

<p>Name: Harshit Jain Access ID: hmj5262 Recitation: 8</p>

Problem 0

Points:

Acknowledgements

- (a) I did not work in a group.
- (b) I did not consult with anyone in my group members.
- (c) I did not consult any non-class materials.

Problem 1

Points:

Algorithm 1: GREEDY-HORN

Input : set of Horn clauses
Output: either the assignment or "unsatisfiable"

```

1 Set all variables to 0;
2 while  $\exists$  an " $\implies$ " that is not satisfied do
3   | Set its RHS to 1;
4 end while
5 if all pure negative clauses are 1 then
6   | return the assignment
7 end if
8 else
9   | return "unsatisfiable"
10 end if

```

(a) According to the algorithm, first set all variables to 0 \implies

$$w = 0, x = 0, y = 0, z = 0$$

Now there are 5 clauses having " \implies " out of which 4th clause ($\implies x$) is not satisfied. So, we will set RHS of this clause to 1, that is, **x=1**.

This will lead to reconsidering the assignment of y because according to 3rd clause ($x \implies y$), if LHS is True then RHS should be set to 1, that is, **y=1**.

This will lead to reconsidering the assignment of w because according to 5th clause ($x \wedge y \implies w$), if LHS is True then RHS should be set to 1, that is, **w=1**.

This will lead to reconsidering the assignment of z because according to 1st clause ($w \wedge y \wedge z \implies x$), if RHS is True then LHS should be resolved to 1, that is, **z=1**.

Now, pure negative clauses are failed to satisfy, so there is no satisfying assignment, hence algorithm will return "**unsatisfiable**".

(b) According to the algorithm, first set all variables to 0 \implies

$$w = 0, x = 0, y = 0, z = 0$$

Now there are 4 clauses having " \implies " out of which 4th clause ($\implies z$) is not satisfied. So, we will set RHS of this clause to 1, that is, **z=1**.

This will lead to reconsidering the assignment of w because according to 2nd clause ($z \implies w$), if LHS is True then RHS should be set to 1, that is, **w=1**.

Here, x and y need not to be changed since the implications are still satisfied with having **x=0, y=0**.

Now, pure negative clauses are still 1 and hence, satisfied. So, the algorithm will return the assignment

$$\mathbf{w=1, x=0, y=0, z=1}$$

Problem 2

Points:

Problem 3**Points:**

Subproblem → Notation: $\text{LCS}(i, j)$ = length of the longest common substring for which there are indices i and j with $x_i x_{i+1} \cdots x_{i+k-1} = y_j y_{j+1} \cdots y_{j+k-1}$ where $1 \leq i \leq n, 1 \leq j \leq m$.

Recurrence → Computing $\text{LCS}(i, j)$:

$$\text{LCS}(i, j) = \begin{cases} 1 + \text{LCS}(i-1, j-1) & ; \text{ if } x[i] = y[j] \\ 0 & ; \text{ otherwise} \end{cases} \quad (1)$$

Base Case: $\text{LCS}(i, 0) = 0, \text{LCS}(0, j) = 0$

Pseudocode:

Algorithm 2: Longest Common Substring

Input : $x = x_1 x_2 \cdots x_n, y = y_1 y_2 \cdots y_m$

Output: k = length of the longest common string

```

1 Set  $k = 0$ ;
2 for  $i = 0$  to  $n$  do
3   for  $j = 0$  to  $m$  do
4     if  $i == 0$  or  $j == 0$  then
5        $\text{LCS}[i][j] = 0$ ;                                /* Base Case */
6     end if
7     else if  $x[i] == y[j]$  then
8        $\text{LCS}[i][j] = 1 + \text{LCS}[i-1][j-1]$ ;                /* Recurrence Formula */
9       if  $\text{LCS}[i][j] \geq k$  then
10         $k = \text{LCS}[i][j]$ ;                                /* Optimal Solution */
11        solutionRow =  $i$ ;                                /* The row holding Optimal Solution */
12        solutionColumn =  $j$ ;                            /* The column holding Optimal Solution */
13      end if
14    end if
15    else
16       $\text{LCS}[i][j] = 0$ 
17    end if
18  end for
19 end for
20 return  $k$ 

```

Explanation:

The approach is to find the length of the longest common substring for all substrings of both the strings x and y and store these lengths in a table LCS . Each cell (i, j) of the table LCS , that is, $\text{LCS}[i][j]$ either holds 0 if $x[i] \neq y[j]$, or holds the length of the common substring of $x[0 \cdots i]$ and $y[0 \cdots j]$ if $x[i] = y[j]$, which is made up including $x[i]$ and $y[j]$. In that way, the table keeps track of all the common substrings available and returns the largest length of common substring available in the table.

If the character happened to be the same while iterating through both the strings x (iterator i) and y (iterator j), then this algorithm checks for the similarity of until previous character of both the strings ($x[i-1] == y[j-1]$) which has already been stored in the table, and add 1 to it which signifies $x[i] = y[j]$.