Artificial Intelligence Assignment – 2

Submitted by

Debonil Ghosh (M21AIE225)

QUESTION:

Given a formula in propositional logic, write a code to put appropriate and necessary brackets in the formula following the precedence order of operators. If not possible, output “Not well formed formula”.

Note-1: We will use the following characters for different operators:
AND: &
OR : |
NOT : !
IMPLICATION : >
BICONDITIONAL : ~

Note-2: Each propositional symbol will be denoted by a capital letter (e.g., A, B, C, …, etc.).

Note-3: The input formula will never contain the truth values “True” and “False”.

Examples:

1. Input:
   P|Q&R~Q>!R
   Output:
   (P|(Q&R))~(Q>(!R))

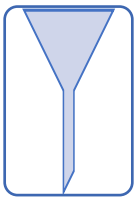
2. Input:
   A>B|C
   Output:
   A>(B|C)

3. Input:
   A>B>C
   Output:
   Not well formed formula

4. Input:
   P|Q&>R
   Output:
   Not well formed formula

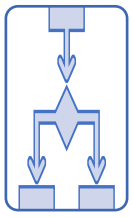
SOLUTION:

1) ALGORITHM USED:



Validate the given Formula

- Check if there exists two consecutive operator or character except the case of negetion operator
- In case of negetion operator, check if two consecutive token is of different type
- Check if there exists more than one Iplication operator
- If any condition from the above holds true return "Not well formed formula"



Form Binary Tree by parsing

- Parse character/operator as a node of tree
- If tree empty, add node as root
- If new node has higher precedence than root node, then add existing tree as left sub tree of new node. And treat new node as root.
- else apply the above step on right child of the root until empty space is available for new node

Extract Result by Tree inorder traversal

- Put root in the middle
- create string for left sub tree recusively
- Put left-sub-tree-string in the left of the root with parenthesis
- create string for right sub tree recusively
- Put right-sub-tree-string in the right of the root with parenthesis

2) ALGORITHM COMPLEXITY:

Best Case: O(nlog(n))
Worst Case: $O(n^2)$

3) Algorithm Pseudo Code:

```
comparePrecedence(a, b)
{
     opa = isOperator(a)
     opb = isOperator(b)
    if (opa != opb)
        return opa < opb
    if (opa)
    {
         preca = getPrecedence (a)
         precb = getPrecedence (b)
        return preca > precb
    }
    return 0
}

createTreeNode(val)
{
    tree = allocteMemory()
    tree->left = NULL
    tree->right = NULL
    tree->val = val
    tree->size = 1
    return tree
}

treeAdd(treeNode, val)
{
    if (treeNode == NULL)
    {
        return createTreeNode(val)
    }
    else if (comparePrecedence (treeNode->val, val))
    {
        newNode = createTreeNode(val)
        newNode->left = treeNode
        newNode->size += treeNode->size
        return newNode
    }
    else
    {
        treeNode->right = treeAdd(treeNode->right, val)
        treeNode->size++
        return treeNode
    }
```

```
}



getParenthesisStr(tree)
{
    buff = ""
    if (tree != NULL)
    {
        op = tree->val
        if (isOperator(op))
        {
            concatchar(buff, PAREN_OPEN)
            concat(buff, getParenthesisStr(tree->left))
            concatchar(buff, op)
            concat(buff, getParenthesisStr(tree->right))
            concatchar(buff, PAREN_CLOSE)
        }
        else
        {
            concatchar(buff, op)
        }
    }

    return buff
}




addParenthesis(tree)
{
    if (tree != NULL && tree->size > 0)
    {
        buff = ""
        concat(buff, getParenthesisStr(tree->left))
        concatchar(buff, tree->val)
        concat(buff, getParenthesisStr(tree->right))
        return buff
    }
    return NOT_WFF
}
```

```
validateAndAddParenthesis(str)
{
     length = strlength(str)
    if (length == 1)
    {
        if (isOperator(str[0]))
            return NOT_WFF
        return str
    }
    last = str[0]
     lstOprnd = isOperator(last)
     uniDirCnt = isUniDirOps(last)
    tree = treeAdd(NULL, last)
    for ( i = 1 i < length i++)
    {
         op = isOperator(str[i])
        if ((op == lstOprnd) != isUnary(str[i]))
            return NOT_WFF
        uniDirCnt += isUniDirOps(str[i])
        if (uniDirCnt > 1)
            return NOT_WFF
        tree = treeAdd(tree, str[i])
        last = str[i]
        lstOprnd = op
    }
    return addParenthesis(tree)
}
```