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))

1. Input:

A>B|C

Output:

A>(B|C)

1. Input:

A>B>C

Output:

Not well formed formula

1. Input:

P|Q&>R

Output:

Not well formed formula

Solution:

1. Algorithm Used:
2. Algorithm Complexity:

Best Case: O(nlog(n))

Worst Case: O(n2)

1. 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)

}