Simplified CKY for NLP

by: Aly Shmahell

August 1, 2016

Contents

1 Pseudo-Code 1

2 Example 1

2.1 CFG 1

2.2 Input 1

2.3 Applying SCKY 1

1 Pseudo-Code

```
line[] = input;
node matrix[line_size+1][line_size];

// initialize the matrix with empty nodes;
for(i; i<line_size+1; i++)
    for(j=0; j<line_size; j++)
    {
        matrix[i][j] = new node;
    }

// populate the anti-diagonal with line[];
for(i=0; i<line_size; i++)
    matrix[msize-i][i] = line[i];

// go through the matrix connecting nodes according to cfg
    rules
for(counter = 0; counter < line_size+1; counter++)
{
    for(row = line_size-counter-1, col = 0; row >= 0 && col <
        line_size-counter; row--, col++)
    {
        if(leaves_in_binary_rule(matrix[row+1][col],matrix[row
            ][col+1])&&(((row!=0||col!=0)&&rule_head!="s")
            ||((row==0&&col==0)&&rule_head=="s")))
        {
            put_binary_rule_head_in(matrix[row][col]);
            connect_head(matrix[row][col]) to_leaves(matrix[
                row+1][col],matrix[row][col+1]);
        }
        else if(leaf_in_unary_rule(matrix[row+1][col]))
        {
            put_unary_rule_head_in(matrix[row][col]);
            connect_head(matrix[row][col]) to_leaf(matrix[row
                +1][col]);
        }
        else
        {
            move_previous(matrix[row+1][col]) to_current(
                matrix[row][col]);
            if(has_one_leaf(matrix[row][col] ) &&
                can_be_other_leaf(matrix[row][col+1]))
                connect_head(matrix[row][col]) to_leaf(matrix[
                    row][col+1]);
        }
    }
}

// check the resulting tree if correct
if(matrix[0][0]=="s")
{
    tree_leaves_size = dfs(matrix[0][0]).size_of_leaves;
    if(tree_leaves_size==line_size)
    {
        tree_head_found;
        return dfs(matrix[0][0]).tree_in_linear_format;
    }
}
```

2 Example

2.1 CFG

```
s -> np vp
s -> vp np
s -> vp
np -> pron
np -> pron noun
np -> det nominal
np -> pron np
nominal -> noun
nominal -> nominal noun
nominal -> nominal pp
vp -> verb
vp -> aux
vp -> verb np
vp -> vp pp
pp -> prep np
```

2.2 Input

```
// assume the following input:
I am a function
// which gives the next tokens line:
pron verb det noun
```

2.3 Applying SCKY