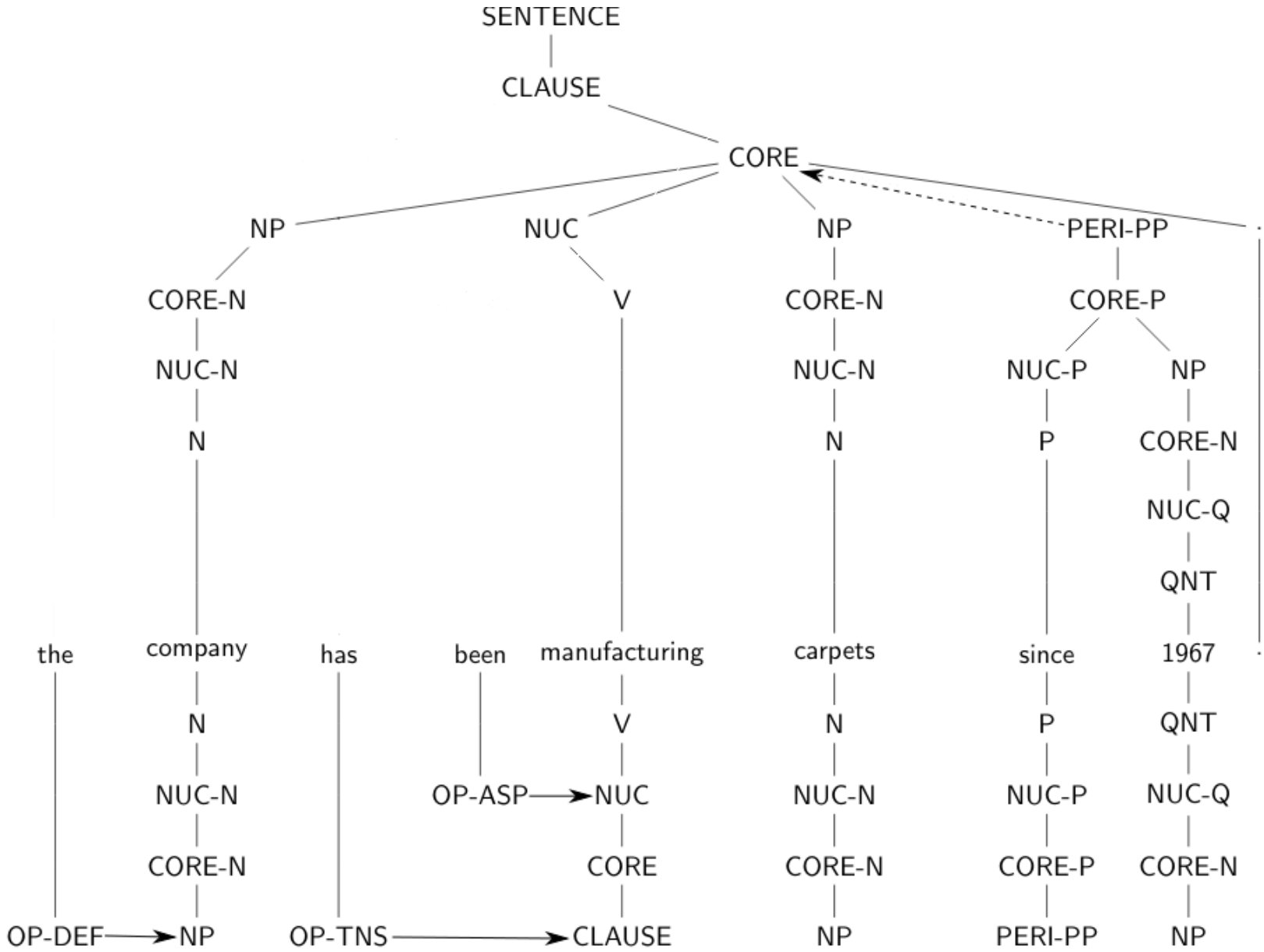
Modifying RRGbank

Davy Baardink

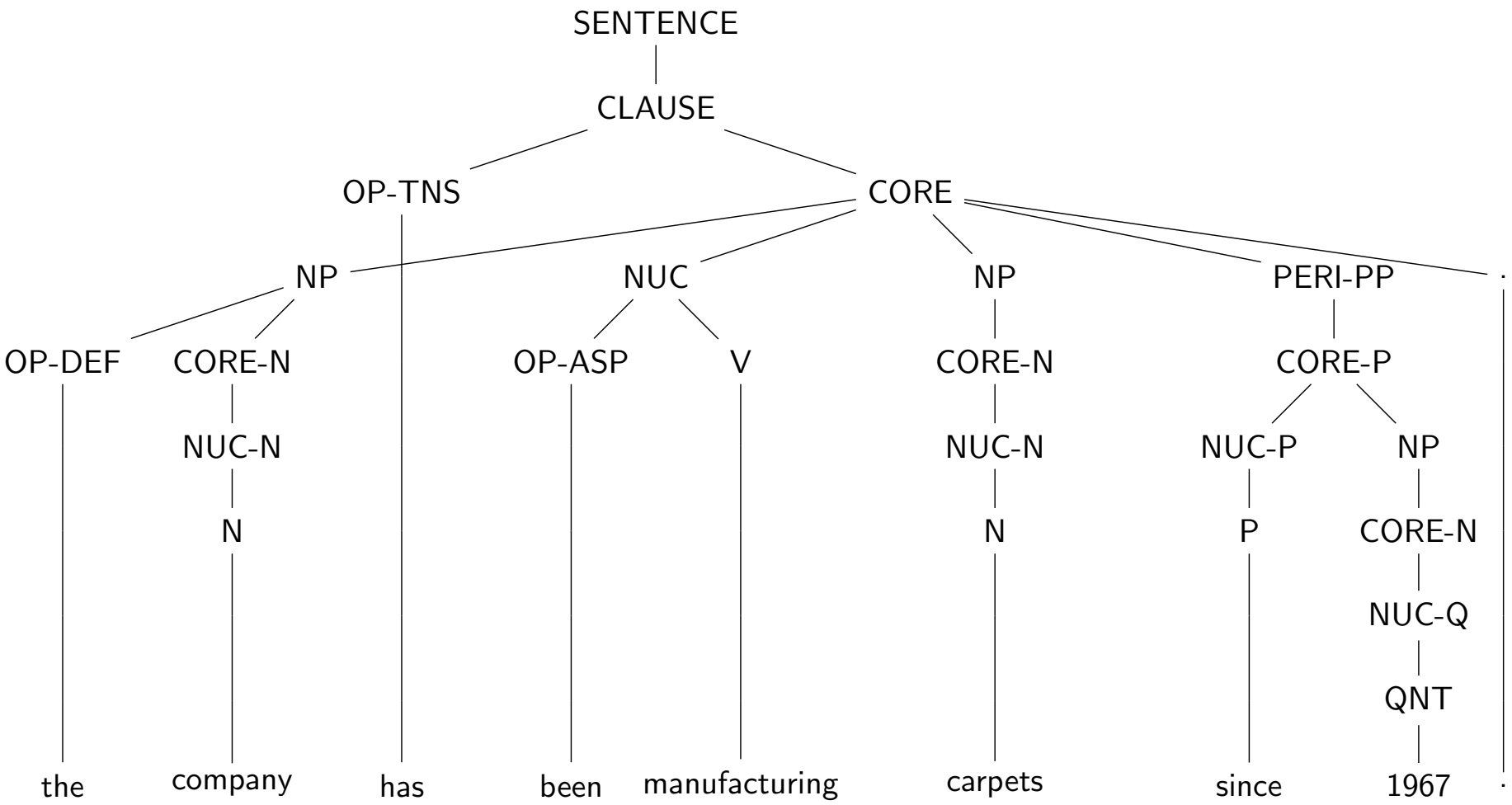
Universität Stuttgart

25-02-2019

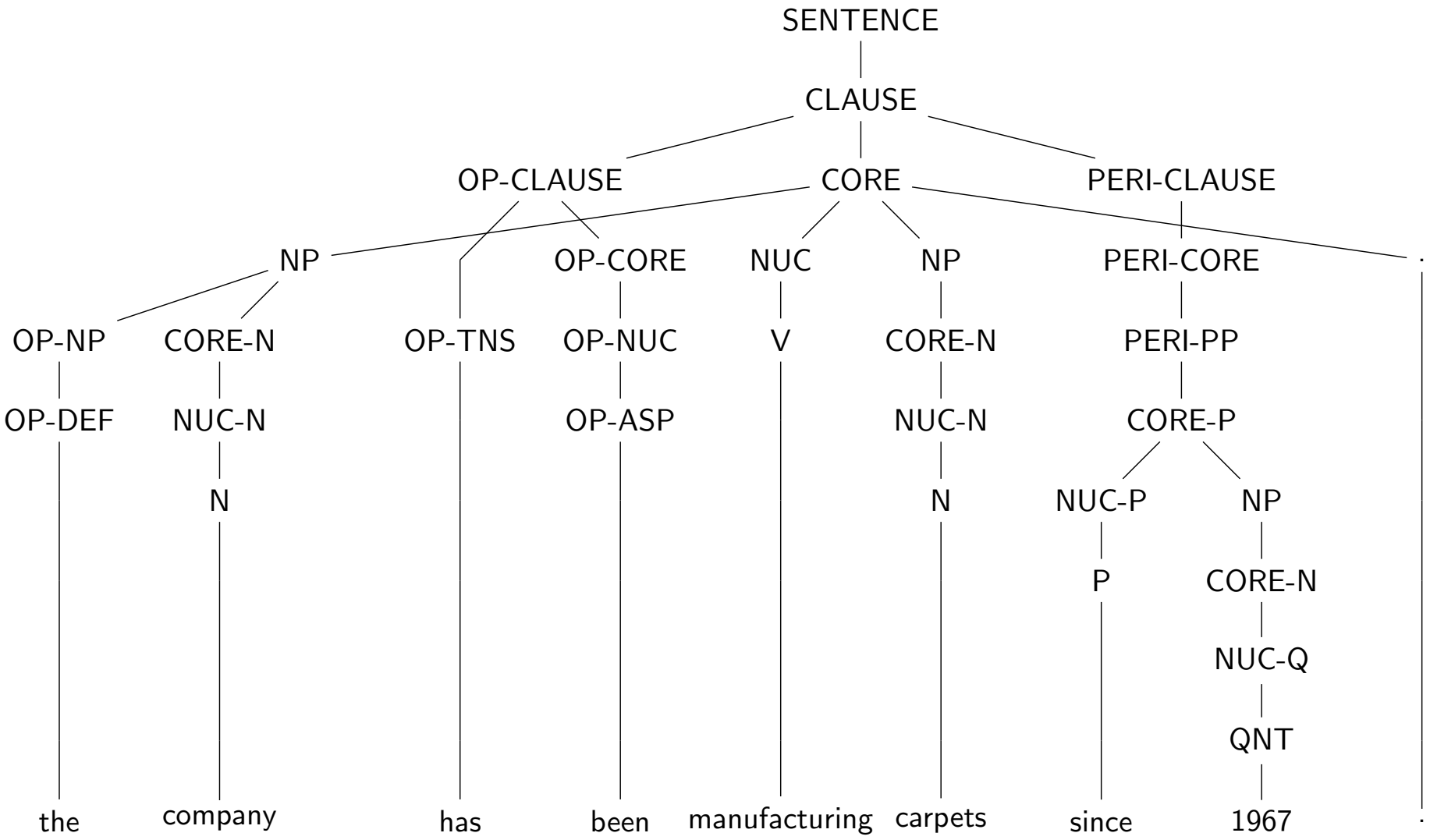
RRG notation (with operator projection)



RRGbank notation



Notation after modification



Motivation

Notation captures the notion that the linear order of operators and peripheral elements reflects their scope

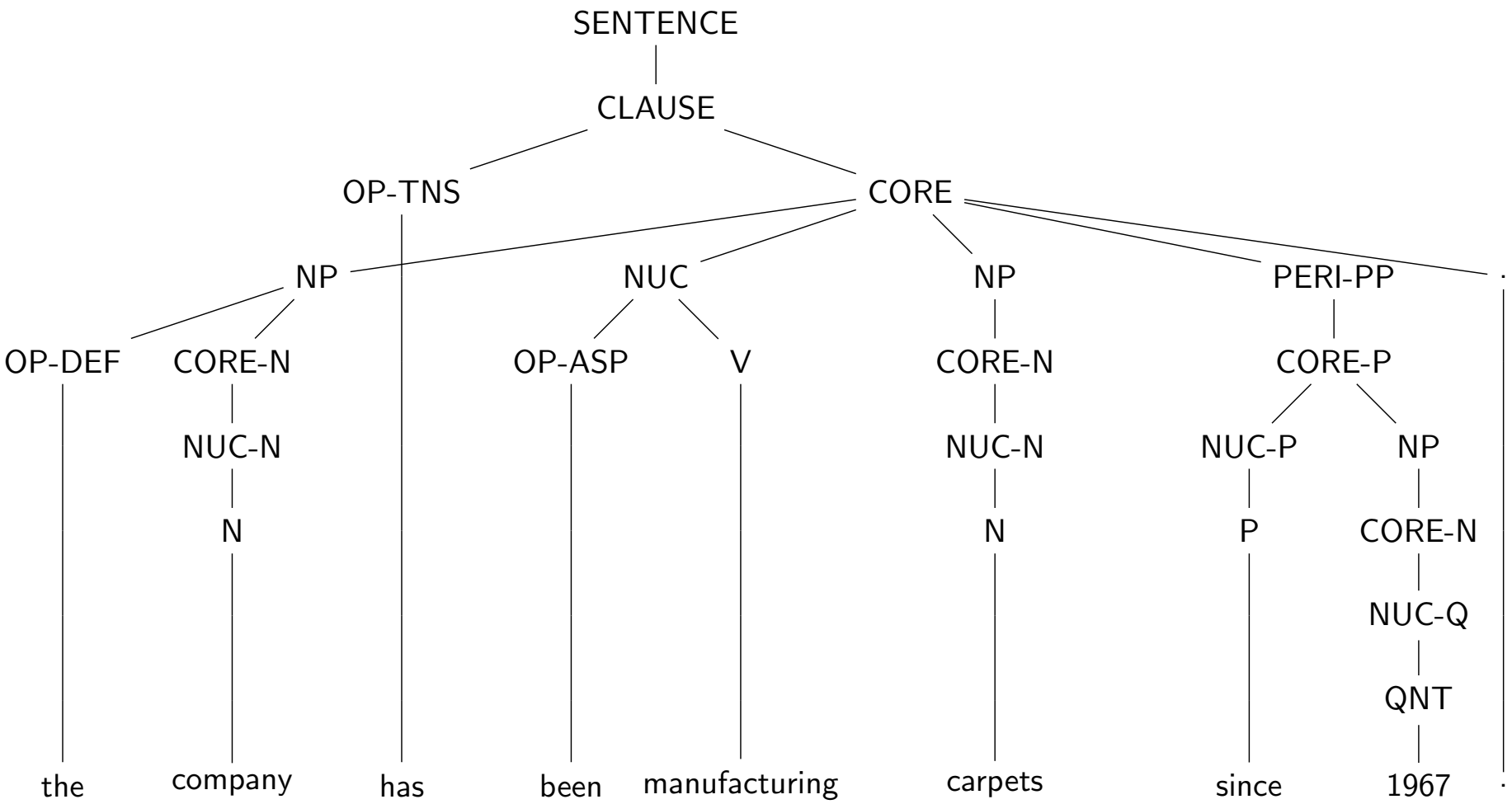
Expected that grammars learned from data in this format will also capture this notion better

Formal Description

Top layer nodes are defined as nodes labeled "CLAUSE", "NP", "PP", "AP" or "ADVP"

The spine of a nuclear node is defined as the subtree consisting of the nuclear node, its closest top-layered ancestor and all nodes and edges in between.

Spines and Local Projections



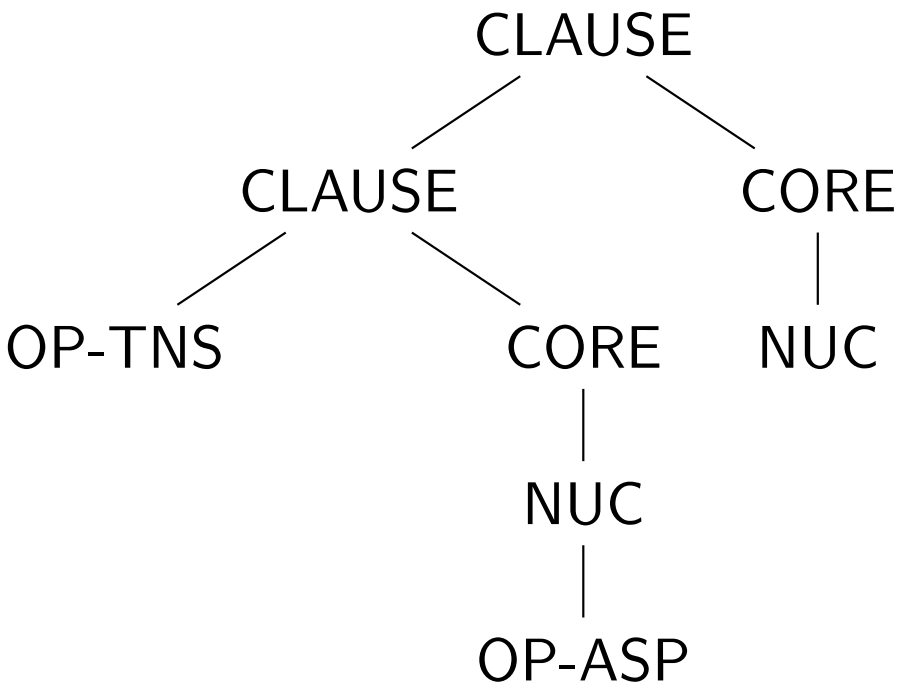
Formal Description

The local operator projection of a nuclear node is defined as the subtree consisting of all nodes on the spine that have an operator daughter, all of their ancestors on the spine and all nodes in between.

Formal Description

We then create a copy of each local operator projection with the same labels and dominance relation and add this copy as a new daughter of the root of the spine. Also for each operator node we exchange the edge from its parent to itself with an edge from the copy of its parent to itself.

Example



Formal Description

Finally, each node on the copy is relabeled.

The same is done for peripheral elements.

Implementation

In the implementation rather than keeping track of which nodes have operator/periphery daughters, the projections are made regardless, and any nodes without lexical descendants are removed as a final step.

This requires less bookkeeping but is less elegant.

Implementation

```
80      def makeProj(self,node,toAdd):
81          if type(node.id)== int:
82              node.id+=toAdd
83          if node.isTop():
84              toAdd-=6
85          for daughter in node.daughters:
86              self.makeProj(daughter,toAdd)
87          if node.isTop():
88              newnodes = []
89              newnodes.append(Node([node.id-1,"","OP_"+node.label,"","",node]))
90              node.daughters.append(newnodes[0])
91              newnodes.append(Node([node.id-2,"","OP_CORE","","",newnodes[0]]))
92              newnodes[0].daughters.append(newnodes[1])
93              newnodes.append(Node([node.id-3,"","OP_NUC","","",newnodes[1]]))
94              newnodes[1].daughters.append(newnodes[2])
95              newnodes.append(Node([node.id-4,"","PERI_"+node.label,"","",node]))
96              node.daughters.append(newnodes[3])
97              newnodes.append(Node([node.id-5,"","PERI_CORE","","",newnodes[3]]))
98              newnodes[3].daughters.append(newnodes[4])
99              newnodes.append(Node([node.id-6,"","PERI_NUC","","",newnodes[4]]))
100             newnodes[4].daughters.append(newnodes[5])
101             self.nodes = self.nodes + newnodes
102
```

Implementation

```
51          for op in reversed(self.opNodes):
52              op.mother.daughters.remove(op)
53              if len(op.mother.label)<3:
54                  target = "OP_"+op.mother.label
55              else:
56                  target = "OP_"+op.mother.label[:3]
57              newMother = self.findNewMother(op,target,"up")
58              op.mother = newMother
59              newMother.daughters.append(op)
```

Implementation

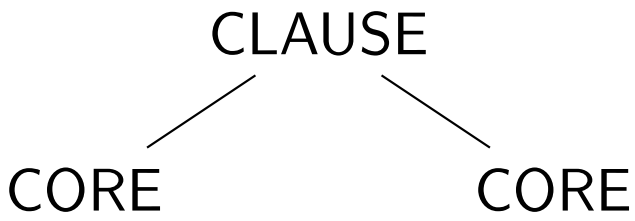
```
102     def findNewMother(self,node,target,direction):
103         if direction == "up":
104             if node.mother.isTop():
105                 return(self.findNewMother(node.mother,target,"down"))
106             else:
107                 return(self.findNewMother(node.mother,target,"up"))
108         elif direction == "down":
109             for daughter in reversed(node.daughters):
110                 if target[0]=="O" and daughter.label[:6] == target:
111                     return daughter
112                 elif target[0]=="P" and daughter.label[:8] == target:
113                     return daughter
114                 elif daughter.label[:3] == target[:3]:
115                     return(self.findNewMother(daughter,target,"down"))
116         print("No mother found")
117         return
```

Implementation

```
69          self.nodes = sorted(self.nodes, key=lambda node : node.id)
70          newnodes = []
71          for node in self.nodes:
72              if node.daughters == []:
73                  node.mother.daughters.remove(node)
74              else:
75                  newnodes.append(node)
76          self.nodes = newnodes
```

Challenges

Nexus relations.



Thanks for listening!