TOP-DOWN & BOTTOM-UP PARSING

SOURCE CODE:

```
import nltk
from nltk import CFG
from nltk.parse import ChartParser
from nltk.tokenize import word_tokenize
nltk.download('punkt')
# Define the grammar
grammar = CFG.fromstring("""
  S -> NP VP
  VP -> V NP
  NP -> NAME
  NP -> ART N
  NAME -> 'John'
  V -> 'ate'
  ART -> 'the'
  N -> 'cat'
""")
# Create the parser
parser = ChartParser(grammar)
# Tokenize the sentence
sentence = "John ate the cat."
tokens = word_tokenize(sentence)[:-1]  # Removing the period
# Original sentence
print("Original Sentence:")
print(sentence)
# TOP-DOWN PARSING
print("\nTOP-DOWN PARSING: ")
top_down_steps = ["S"]  # Start with the root node
np_count = 0  # Count NP expansions to differentiate them
print(*top_down_steps)
while True:
  expanded = top_down_steps[-1].split()
  for i, symbol in enumerate(expanded):
    if symbol == "S":
      expanded[i:i+1] = ["NP", "VP"]
      np_count += 1
      break
    elif symbol == "VP":
      expanded[i:i+1] = ["V", "NP"]
      np_count += 1
      break
    elif symbol == "NP":
      if np_count == 1:  # First NP → NAME
        expanded[i:i+1] = ["NAME"]
      else:  # Second NP → ART N
        expanded[i:i+1] = ["ART", "N"]
      break
    elif symbol == "NAME":
      expanded[i] = "John"
      break
```

```
        elif symbol == "V":
            expanded[i] = "ate"
            break
        elif symbol == "ART":
            expanded[i] = "the"
            break
        elif symbol == "N":
            expanded[i] = "cat"
            break
    step_sentence = ' '.join(expanded)
    if step_sentence == top_down_steps[-1]:  # Stop if no more expansion
        break
    top_down_steps.append(step_sentence)
    print(step_sentence)
# Ensure the final step matches the full sentence
if top_down_steps[-1] != ' '.join(tokens):
    top_down_steps.append(' '.join(tokens))
    print(top_down_steps[-1])
# BOTTOM-UP PARSING
print("\nBOTTOM-UP PARSING: ")
bottom_up_steps = []
current = tokens[:]
while len(current) > 1:
    if current[0] == "John":
        current[0] = "NAME"
    elif current[1] == "ate":
        current[1] = "V"
    elif current[2] == "the":
        current[2] = "ART"
    elif len(current) > 3 and current[3] == "cat":
        current[3] = "N"
    elif len(current) > 3 and current[2] == "ART" and current[3] == "N":
        current = current[:2] + ["NP"]
    elif len(current) > 2 and current[1] == "V" and current[2] == "NP":
        current = ["S"]
    bottom_up_steps.append(' '.join(current))
    print(bottom_up_steps[-1])
    if current == ["S"]:
        break  # Stop when the sentence is fully parsed
# Parse the sentence and print the parse tree
print("\nTop-Down Parsing Tree:")
for tree in parser.parse(tokens):
    print(tree)
    tree.pretty_print()
```

OUTPUT:

```
Original Sentence:
John ate the cat.

TOP-DOWN PARSING:
S
NP VP
NAME VP
```

John VP
John V NP
John ate NP
John ate ART N
John ate the N
John ate the cat

BOTTOM-UP PARSING:
NAME ate the cat
NAME V the cat
NAME V ART cat
NAME V ART N
NAME V NP
S

Top-Down Parsing Tree:
(S (NP (NAME John)) (VP (V ate) (NP (ART the) (N cat))))