In [2]:
```
!pip install nltk matplotlib -q
```

In [3]:
```
import nltk
import matplotlib.pyplot as plt

# NLTK doesn't need extra downloads for Tree, but this ensures core data is pres
# (safe to run; it will do nothing if already downloaded)
nltk.download("punkt", quiet=True)

from nltk import Tree
```

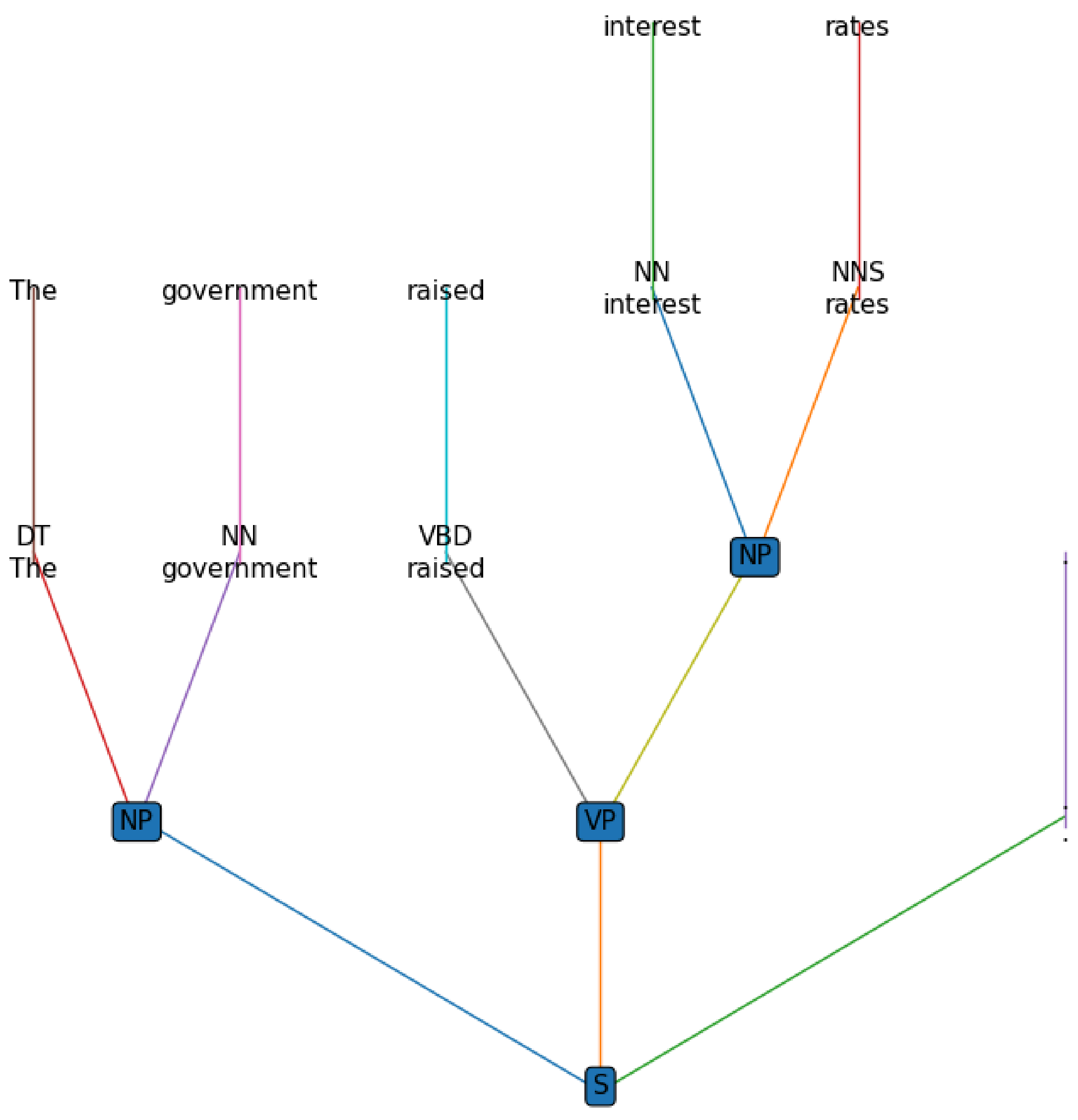
```
Matplotlib is building the font cache; this may take a moment.
```

In [4]:
```
t1 = Tree.fromstring(
    "(S (NP (DT The) (NN government)) "
    "(VP (VBD raised) (NP (NN interest) (NNS rates))) (. .))"
)

t2 = Tree.fromstring(
    "(S (NP (DT The) (NN internet)) "
    "(VP (VBZ gives) (NP (NN everyone)) (NP (DT a) (NN voice))) (. .))"
)

# Ambiguous sentence: two attachments

# 3a) Instrument reading (PP → VP)
t3a = Tree.fromstring(
    "(S (NP (DT The) (NN man)) "
    "(VP (VBD saw) (NP (DT the) (NN dog)) "
    "(PP (IN with) (NP (DT the) (NN telescope)))) (. .))"
)

# 3b) Modifier-of-dog reading (PP → NP)
t3b = Tree.fromstring(
    "(S (NP (DT The) (NN man)) "
    "(VP (VBD saw) (NP (NP (DT the) (NN dog)) "
    "(PP (IN with) (NP (DT the) (NN telescope))))) (. .))"
)
```

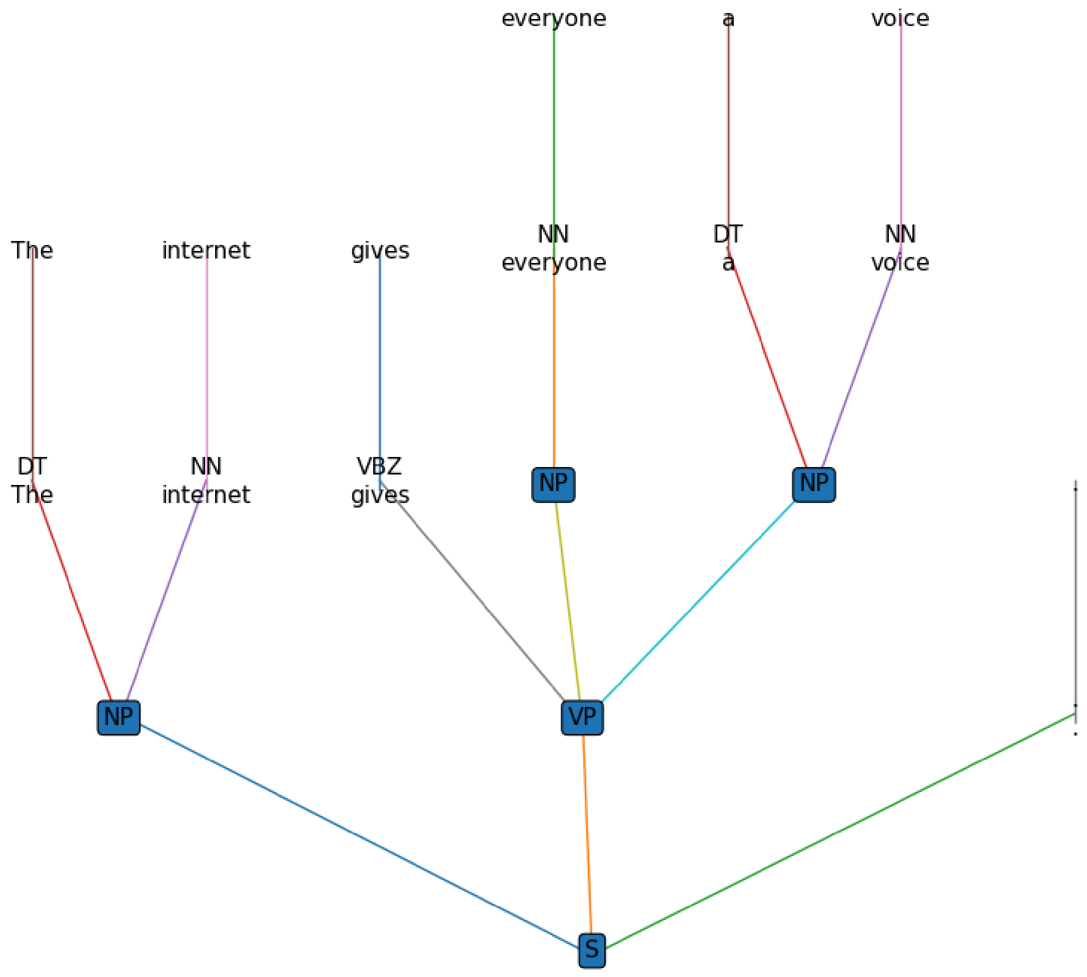
In [5]:
```
from statistics import mean

def draw_constituency_tree(t: Tree, title=None, save_path=None,
                           node_fs=11, word_fs=11, vgap=1.3, margin=0.5):
    """
    Draw an NLTK Tree inline in Jupyter using matplotlib, and optionally save to
    - t: nltk.Tree
    - title: str title above the figure
    - save_path: e.g., "tree1.png" to save a PNG
    """

    # --- 1) Compute positions using treepositions (robust for repeated words) -
    # positions are tuples like (), (0,), (0,1), etc.
    all_pos = list(t.treepositions())
    leaf_pos = list(t.treepositions('leaves'))

    # x positions: leaves get 0..N-1; internal nodes get mean of children
    xpos = {}
    for i, lp in enumerate(leaf_pos):
        xpos[lp] = i

    # fill internal nodes from bottom-up by depth
    for pos in sorted([p for p in all_pos if p not in leaf_pos], key=len, revers
        node = t[pos]
        if isinstance(node, Tree) and len(node) > 0:
            child_positions = [pos + (i,) for i in range(len(node))]
            child_xs = [xpos.get(cp) for cp in child_positions if cp in xpos]
            if child_xs:
                xpos[pos] = mean(child_xs)

    # y positions by depth (root at top)
    ypos = {pos: -len(pos) for pos in all_pos}

    # --- 2) Normalize to figure coordinates ---
    xs = list(xpos.values())
    ys = list(ypos.values())
    minx, maxx = min(xs), max(xs)
    miny, maxy = min(ys), max(ys)

    def nx(x, width):
        if maxx == minx: return margin + (width - 2*margin)/2
        return margin + (x - minx) / (maxx - minx) * (width - 2*margin)
    def ny(y, height):
        if maxy == miny: return height/2
        return height - (margin + (y - miny) / (maxy - miny) * (height - 2*margi

    # figure size from leaf count and depth
    n_leaves = len(leaf_pos)
    depth = max(len(p) for p in all_pos) + 1
    width = max(n_leaves * 1.0 + 2*margin, 4)
    height = max(depth * vgap + 2*margin, 3)

    fig, ax = plt.subplots(figsize=(width, height))
    ax.set_axis_off()

    # --- 3) Draw edges (parent→child) ---
    for pos in all_pos:
        node = t[pos]
        if isinstance(node, Tree):
            for i in range(len(node)):
                child_pos = pos + (i,)
                if child_pos in xpos:
                    ax.plot([nx(xpos[pos], width), nx(xpos[child_pos], width)],
                            [ny(ypos[pos], height)-0.03, ny(ypos[child_pos], hei
                            linewidth=1)

    # --- 4) Draw labels ---
    for pos in all_pos:
        node = t[pos]
        x = nx(xpos[pos], width)
        y = ny(ypos[pos], height)

        if isinstance(node, Tree):
            # preterminal if it has exactly one string child
            is_preterminal = (len(node) == 1) and isinstance(node[0], str)
            if is_preterminal:
                # POS tag at node; word at leaf a bit lower
                ax.text(x, y + 0.12, node.label(), ha="center", va="center", fon
                ax.text(x, y - 0.08, node[0], ha="center", va="center", fontsize
                else:
                    ax.text(x, y, node.label(), ha="center", va="center", fontsize=n
                            bbox=dict(boxstyle="round,pad=0.25", linewidth=0.8))
            else:
                # should not happen (leaves handled via preterminals), but safe-guar
                ax.text(x, y, str(node), ha="center", va="center", fontsize=word_fs)

    if title:
        ax.set_title(title, pad=10)

    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=200, bbox_inches="tight")
    plt.show()
```

In [6]:
```
draw_constituency_tree(t1, title="The government raised interest rates.")
draw_constituency_tree(t2, title="The internet gives everyone a voice.")
draw_constituency_tree(t3a, title="The man saw the dog with the telescope. (VP-a
draw_constituency_tree(t3b, title="The man saw the dog with the telescope. (NP-a
```

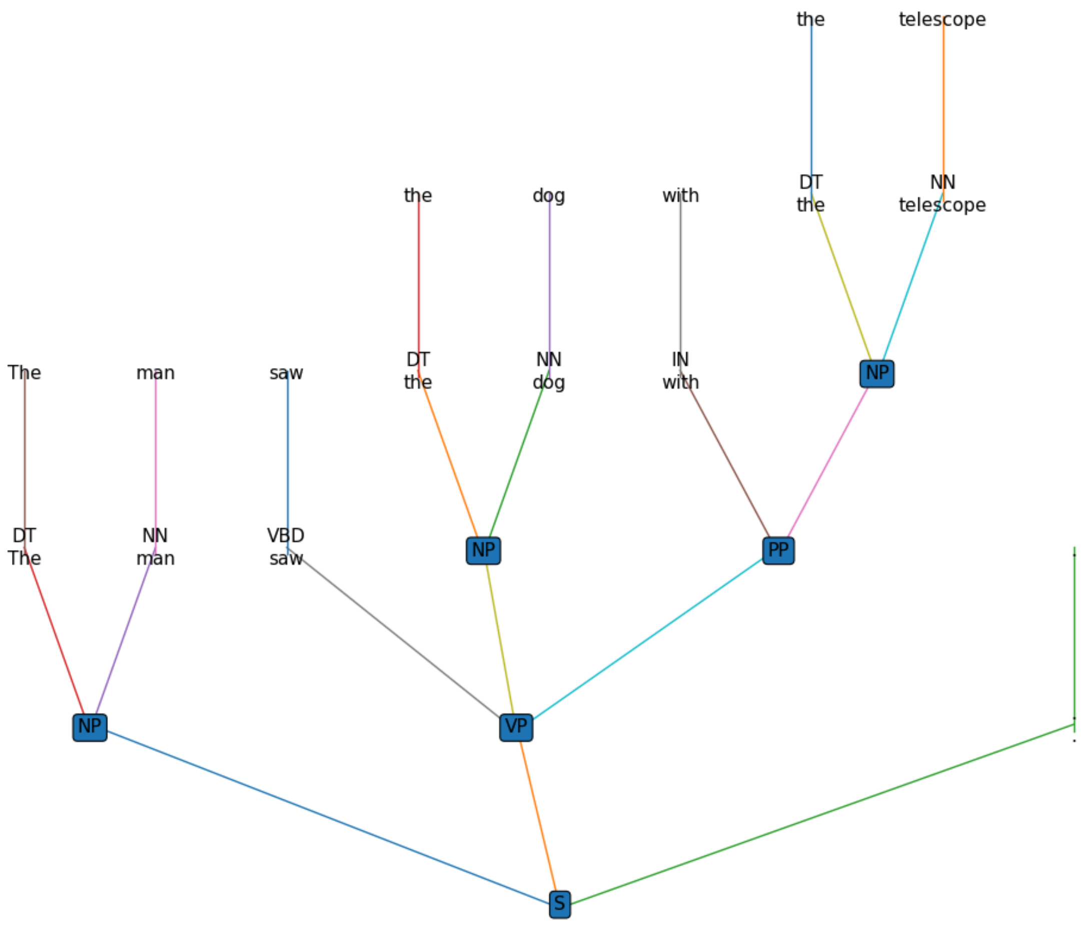
The government raised interest rates.



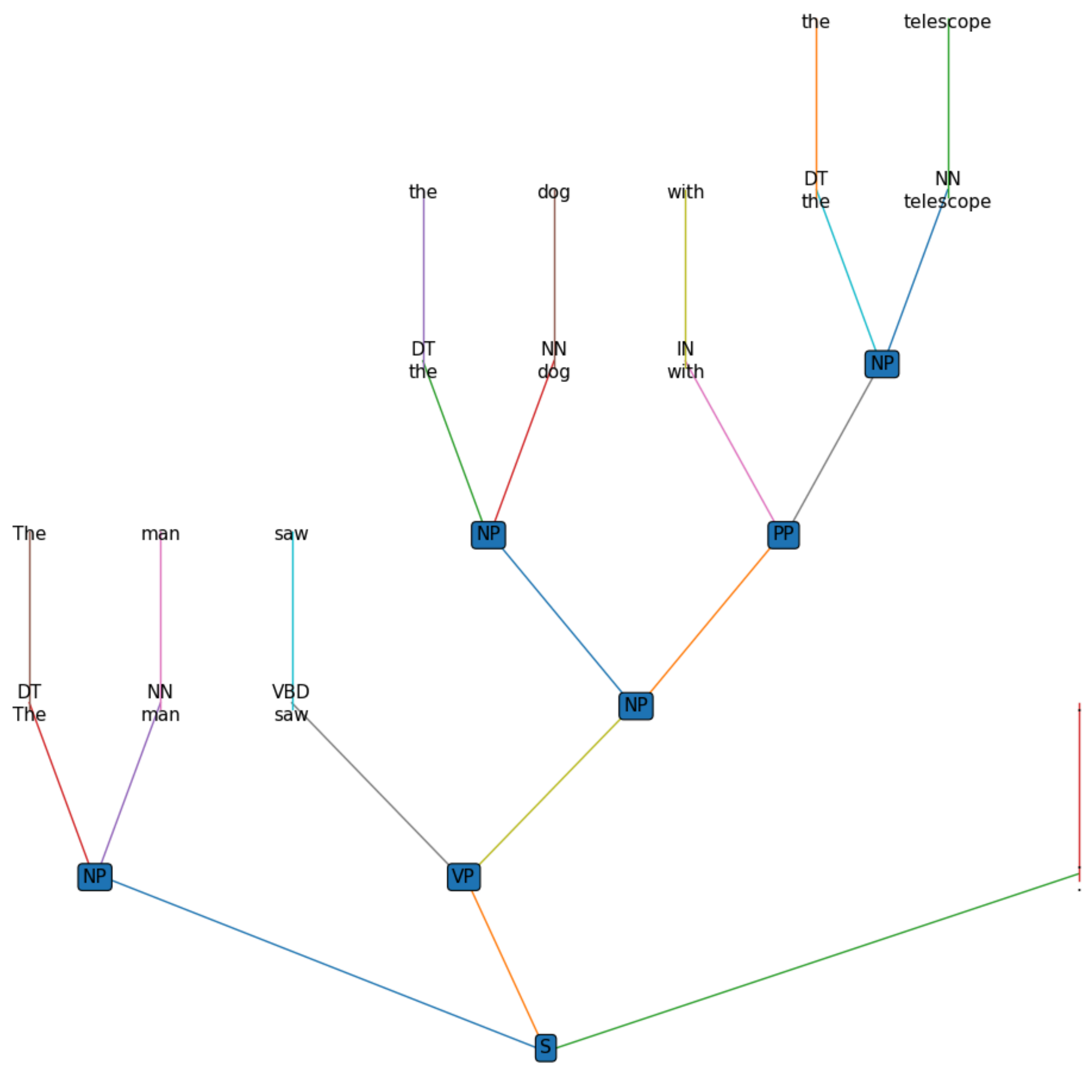
The internet gives everyone a voice.



The man saw the dog with the telescope. (VP-attach / instrument)

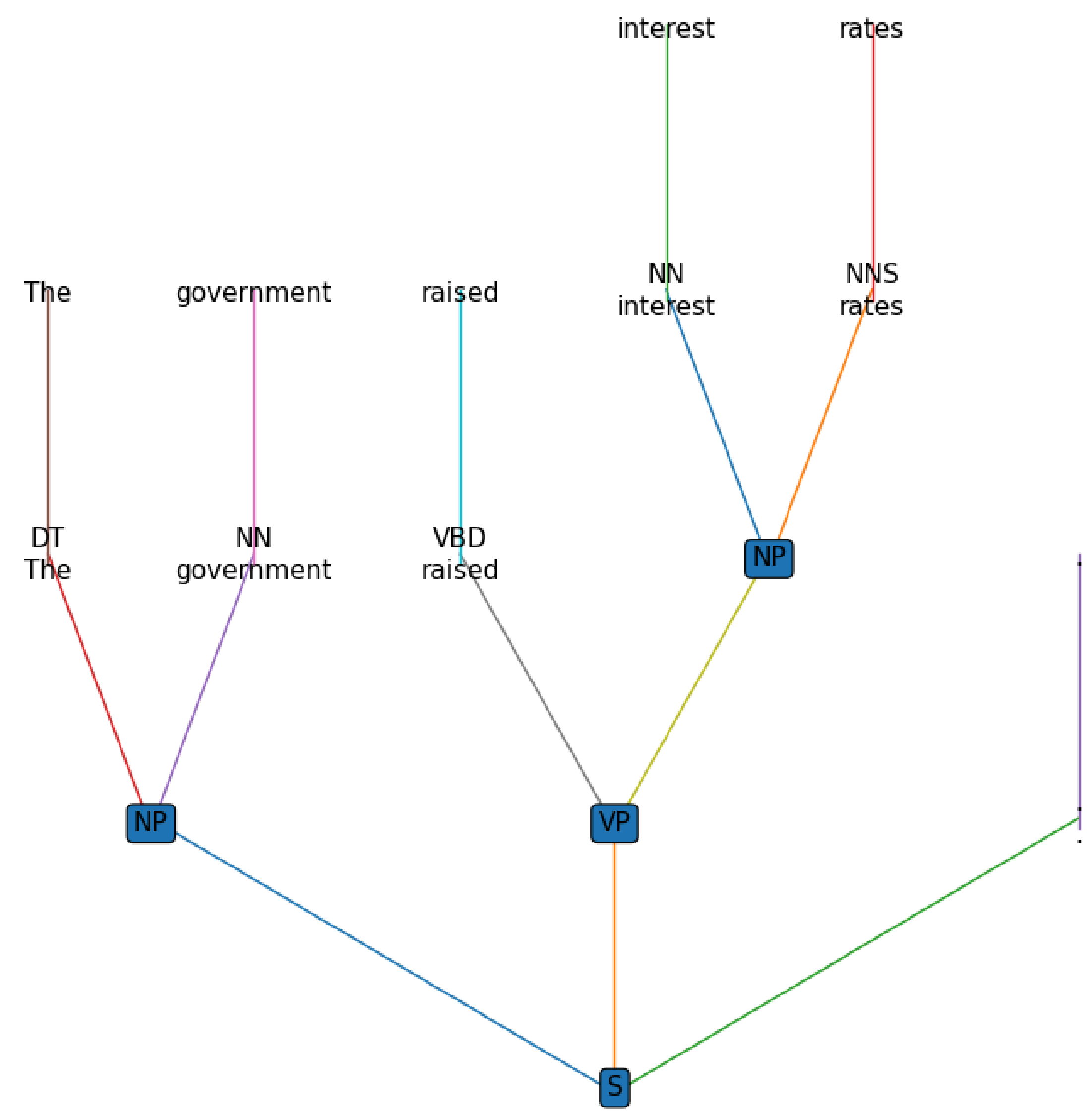


The man saw the dog with the telescope. (NP-attach / modifier)

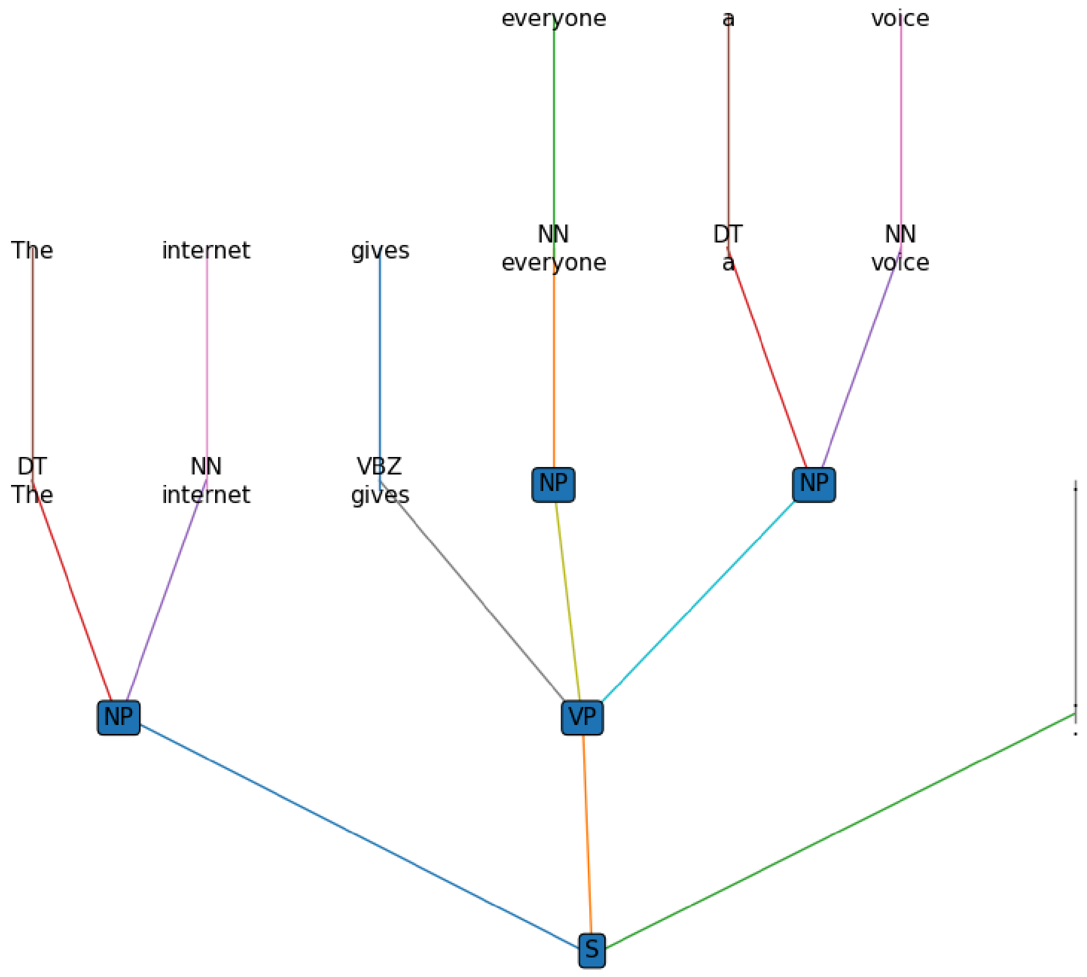


```
In [8]: draw_constituency_tree(t1, title="The government raised interest rates.", save_p
        draw_constituency_tree(t2, title="The internet gives everyone a voice.", save_pa
        draw_constituency_tree(t3a, title="The man saw the dog with the telescope. (VP-a
        draw_constituency_tree(t3b, title="The man saw the dog with the telescope. (NP-a

        print("\n--- Markdown snippet  ---\n")
        print("### Constituency Parse Trees")
        print("![Tree 1](tree1.png)")
        print("![Tree 2](tree2.png)")
        print("![Tree 3a](tree3a.png)")
        print("![Tree 3b](tree3b.png)")
```

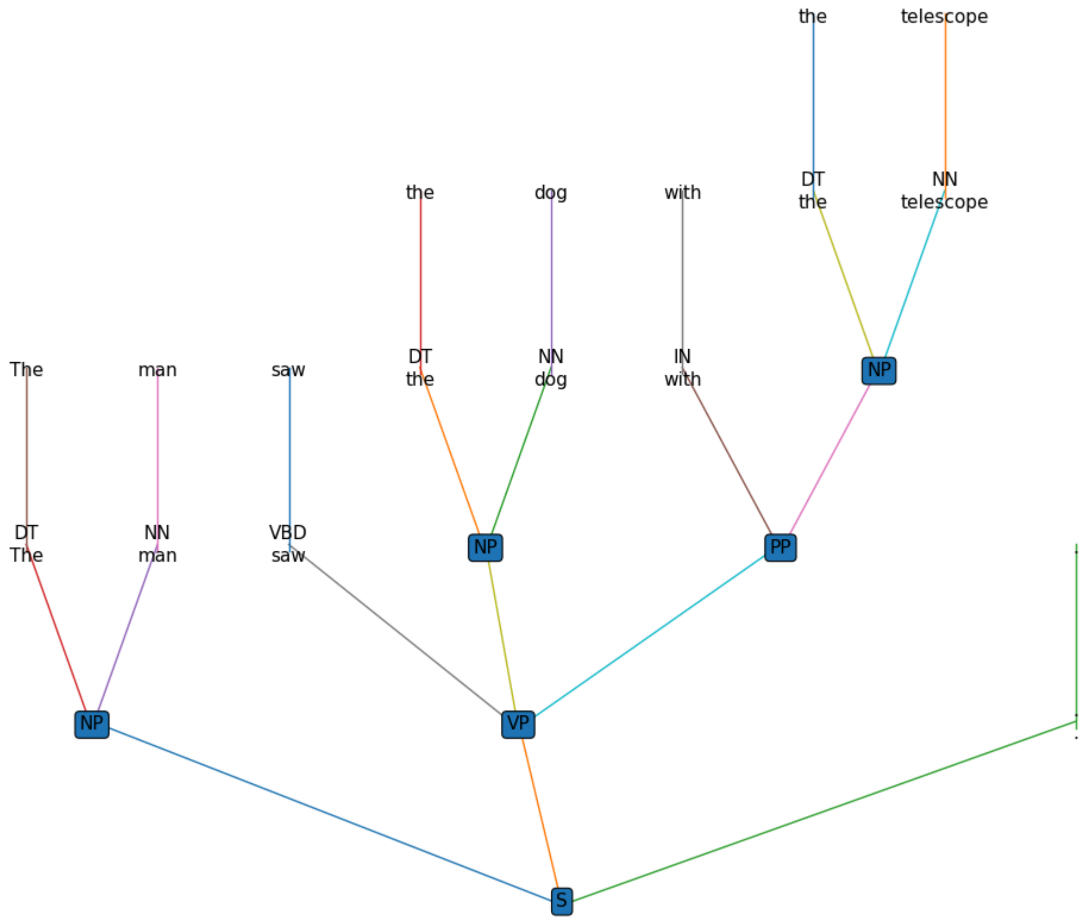
The government raised interest rates.



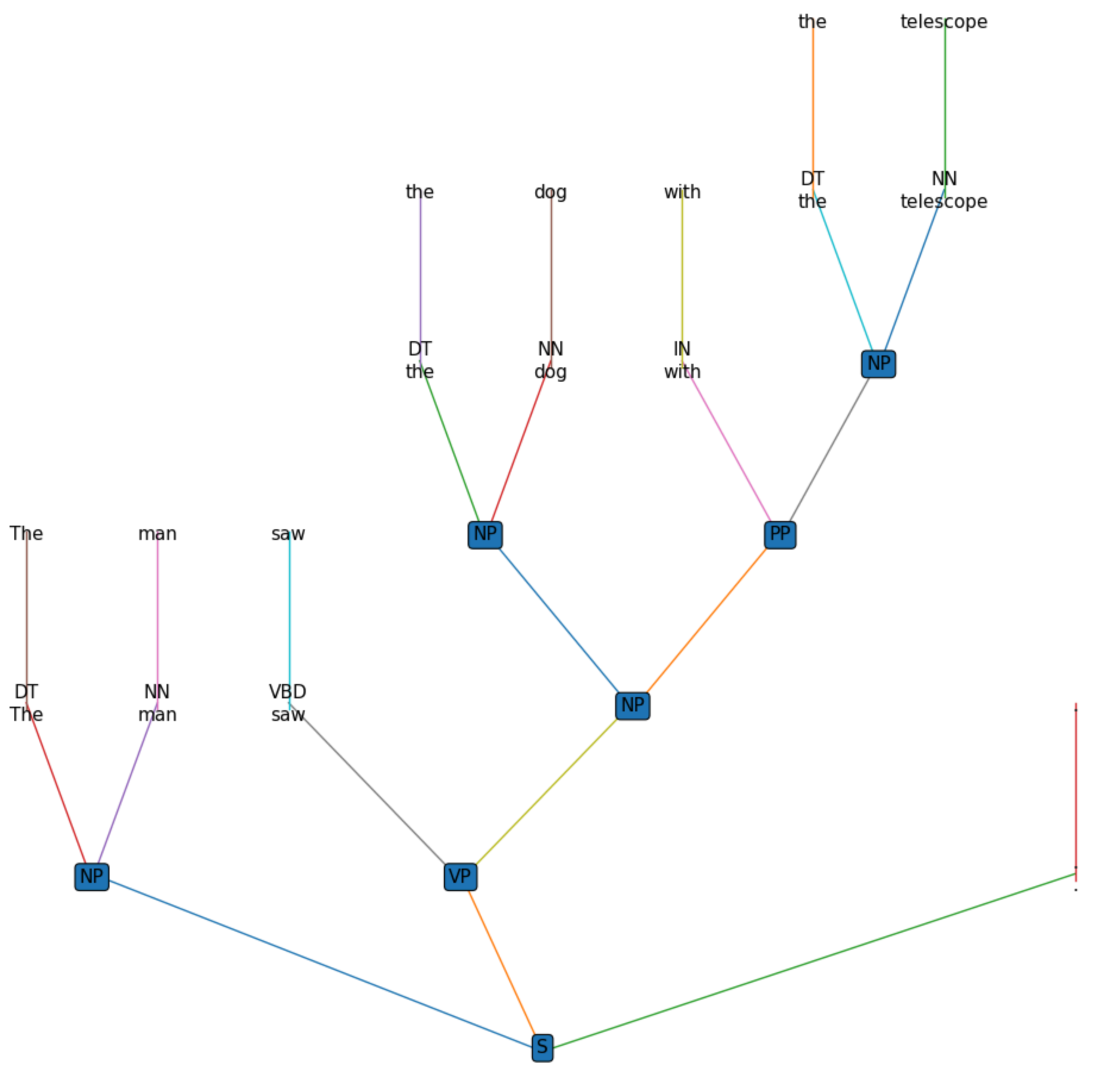
The internet gives everyone a voice.



The man saw the dog with the telescope. (VP-attach / instrument)



The man saw the dog with the telescope. (NP-attach / modifier)



```
--- Markdown snippet  ---

### Constituency Parse Trees
![Tree 1](tree1.png)
![Tree 2](tree2.png)
![Tree 3a](tree3a.png)
![Tree 3b](tree3b.png)
```

In [9]:
```
for i, t in enumerate([t1, t2, t3a, t3b], 1):
    print(f"Tree {i}:", t.pformat(margin=100))
```

```
Tree 1: (S (NP (DT The) (NN government)) (VP (VBD raised) (NP (NN interest) (NNS
rates))) (. .))
Tree 2: (S (NP (DT The) (NN internet)) (VP (VBZ gives) (NP (NN everyone)) (NP (DT
a) (NN voice))) (. .))
Tree 3: (S
  (NP (DT The) (NN man))
  (VP (VBD saw) (NP (DT the) (NN dog)) (PP (IN with) (NP (DT the) (NN telescop
e))))
  (. .))
Tree 4: (S
  (NP (DT The) (NN man))
  (VP (VBD saw) (NP (NP (DT the) (NN dog)) (PP (IN with) (NP (DT the) (NN telesco
pe)))))
  (. .))
```

```
In [ ]:
```