Alternative Grammatical Formalisms

Darcey Riley

CS 465: Natural Language Processing

December 7, 2012
Why use other grammatical formalisms?

- CFGs are useful for describing natural language.
- Unlike regular expressions, they are expressive enough to capture many dependencies in natural language.
- Unlike more expressive formalisms, they can be parsed efficiently in $O(n^3)$ time.
  - For instance, TAG takes $O(n^6)$ time to parse.
  - There is probably no polynomial time algorithm to parse context sensitive grammars.
Limitations of CFGs

- CFGs have trouble capturing non-local dependencies, such as verb subcategorization frames.
Limitations of CFGs

- CFGs have trouble capturing non-local dependencies, such as verb subcategorization frames.
What to do about the limitations of CFGs?

- This is why a CFG read directly off the Penn Treebank has poor parsing performance.
- One solution: create a different CFG which captures this dependency by annotating the nonterminals with additional information capturing these dependencies.
  - Could annotate with specific features, like the head word.
  - Could automatically learn to cluster each symbol into subsymbols.

```
S
  |   |
NP  VP[put]
  |   |   |
N V[put] NP PP
  |   |   |   |
Jeff put DT N P NP
  |   |   |   |   |
the book on DT N
  |   |   |   |   |
the table
```
What to do about the limitations of CFGs?

- Another solution: use alternative grammatical formalisms.
- This lecture will present two such formalisms:
  - Tree Substitution Grammar (TSG)
  - Tree Adjoining Grammar (TAG)
- There are many other such formalisms, notably Combinatory Categorial Grammar (CCG).
What is a Tree Substitution Grammar?

A TSG is like a CFG, but instead of rewriting as a string of nonterminals and terminals, each nonterminal rewrites as a whole tree fragment.
Formal Definition

- Formally, a TSG is a 4-tuple $G = (T, N, S, R)$
  - $T$: terminals
  - $N$: nonterminals
  - $S$: distinguished start symbol
  - $R$: productions
- Each production is a *tree fragment*, also called an *elementary tree*.
  - Internal nodes are nonterminals.
  - Leaf nodes can be terminals or nonterminals.
  - Nonterminals at leaves are called *frontier nodes* or *substitution sites*, marked with $\downarrow$.
  - These are the nonterminals which rewrite as elementary trees.

```
             VP
               ↓
              VP
              ↓   ↓
             NP  PP
             ↓  ↓
            put
```

```
            NP
           ↓
          DT  N
          ↓  ↓
         the  Jeff
```

```
         NP
        ↓
       N
      ↓
      N
```
The process of rewriting is called *derivation*.
- It continues until there are no frontier nodes left to replace.
- The result is called the *derived tree*.

As usual, once the derivation process is finished, we can read the sentence off the leaves of the tree.
Equivalence of CFGs and TSGs

- TSGs and CFGs are formally equivalent.
- Every CFG is just a TSG whose rules all have height 2.
- Every TSG can be converted to a CFG by flattening the rules.
- So TSG is no more expressive than CFG.
- But in practice it’s useful for learning from the Penn Treebank.
- Lets us capture long-distance dependencies, such as verb subcategorization frames.
- And because it’s formally equivalent to CFG, we can still parse in $O(n^3)$ time.

```
VP     NP    PP  
|      ↓     ↓   
V     put      
```

```
NP    PP     put  
|      ↓     ↓   
NP    NP      
```
What is a Tree Adjoining Grammar?

- TAG is like TSG, except that in addition to substitution, we also have an operation called adjunction.
- Substitution occurs at frontier nodes and adds things to the bottom of the tree.
- On the other hand, adjunction occurs at internal nodes, and involves a different kind of tree fragment, called auxiliary trees.
- An auxiliary tree has a special foot node, marked with a *, that is the same nonterminal as its root node.
Adjunction in Action

```
S
  NP
    N Justin
    V fired
    NP DT a
    N laser

VP
  VP*
    ADV quickly

VP
```
Adjunction in Action

S
  └── VP
      └── VP*
          └── ADV
              └── quickly

N
  └── NP
      └── VP
          └── V
              └── NP
                  └── DT
                          └── N
                              └── a
                                  └── laser

Justin

fired
Adjunction in Action

```
S
  NP  VP
    N   VP
      Justin V NP
        fired DT N
            a N
                laser
    ADV
  quickly
```
Formally, a TAG is a 5-tuple \( G = (T, N, S, E, A) \)
- \( T \): terminals
- \( N \): nonterminals
- \( S \): distinguished start symbol
- \( E \): elementary trees
- \( A \): auxiliary trees

Some formulations of TAG allow substitution as an operation.
- Others only allow adjunction.
- These turn out to be equivalent.
- If there is no substitution, the elementary trees \( E \) will only have terminals on their frontiers.
Not formally equivalent to CFG!

TAG can generate languages like $a^n b^n c^n$, $a, b, c \in T$ and $ww$, $w \in T^*$, which are not context free.

- $ww$ corresponds to *cross-serial dependencies*, as in the sentence “Jason gave a cookie and some chocolate to Sonja and Jonas, respectively.”

It turns out that TAG is also strictly less powerful than context sensitive languages.

- TAG generates what are called *mildly context sensitive languages*. 
Parsing with TAGs

- TAGs can be parsed in $O(n^6)$ time, using a dynamic programming algorithm similar to CKY.
- As with CKY, we restrict the grammar to be binary branching.
- No nonterminal in any tree can have more than two children.
- We also assume (for the sake of simplicity) that the only operation is adjunction.
- Lastly, we follow the restriction that adjunction can only happen once at each node.
Some TAG-Parsing Intuitions (Part 1)

- We will have a CKY-like chart where we keep track of what constituents we’ve built so far.
- What should these constituents be?
- They can’t be entire tree fragments: we can’t just parse a whole tree fragment at once, because what if something got adjoined inside it?
- This leads us to the first intuition: we should **split up the tree fragments into their component CFG rules**.
- Using these component rules, we can then parse in a bottom-up fashion.
Now that we’ve decomposed our rules in manner, what’s to stop us from applying the CKY algorithm in the usual fashion?

Let’s see what happens if we try it.

We parse for a bit, start building the bottom of one of our elementary trees $\alpha$, and then we get to a nonterminal where there could be adjunction of the elementary tree $\beta_1$.

So we start building the auxiliary tree there. But we need to keep track of what elementary tree we were building, so we can finish it later.

No problem, we say. We’ll just pass that information up the tree.
But what happens when we’re in the middle of parsing auxiliary tree $\beta_1$, and we get to a nonterminal that could be the foot of auxiliary tree $\beta_2$?

Well, then we start parsing $\beta_2$, but again we need to keep track of the fact that we were in the middle of parsing $\beta_1$, while in the middle of parsing $\alpha$.

In order to keep track of all this, we’d need a stack.

Intuitively, this is why we can’t just use CKY to parse a TAG.
How do we get around the problems described above?

By parsing the auxiliary trees first!

We start by parsing the auxiliary trees that were adjoined most recently.

Once these are complete, we can adjoin them in directly while chart parsing, without having to maintain a stack.
Chart Cells

- We will parse auxiliary trees in isolation, before they are adjoined to anything.
- In our chart cells, we need to keep track of the “hole” in the middle where the foot node is.
- Chart cells take the following form: \([N^\gamma, i, j|p, q|adj]\)
  - \(N\) is the nonterminal.
  - \(\gamma\) records which tree fragment we are parsing.
  - \(i\) and \(j\) are the start and end positions of the chart cell.
  - \(p\) and \(q\) are the start and end positions of the hole, \(0 \leq i \leq p \leq q \leq j\). (If we are parsing something without a hole, we leave \(p, q\) blank.)
  - \(adj\) is a boolean to enforce the restriction that we can only adjoin once per node.
- \([N^\gamma, i, j|p, q|adj]\) is true if:
  - \(N^\gamma \Rightarrow^* a_{i+1} \ldots a_p F^\gamma a_{q+1} \ldots a_j\) (\(\gamma\) must be an auxiliary tree)
  - \(N^\gamma \Rightarrow^* a_{i+1} \ldots a_j\) (\(\gamma\) could be either kind of tree)
Initializing the Chart

- Terminal rule: $[N^γ, i-1, i|-, -|\text{false}]$ if $N^γ \rightarrow a$
- Epsilon rule: $[N^γ, i, i|-, -|\text{false}]$ if $N^γ \rightarrow \epsilon$
- Foot rule: $[F^γ, i, j|i, j|\text{false}]$ if $F$ is the foot node of $γ$
Rules for Building the Insides of Tree Fragments

- In an auxiliary tree, the path from the root node to the foot node is called the “spine” of the tree fragment.
- Parsing a rule $N \gamma \rightarrow X \gamma Y \gamma$ that appears inside a tree fragment.
- Three possibilities: $X$ dominates the foot node (and thus contains a hole), $Y$ dominates the foot node (and thus contains a hole), or neither contains a hole.
- $[N \gamma, i, j|p, q|false]$ if $N \gamma \rightarrow X \gamma Y \gamma$ and one of the following holds:
  - $X$ has the hole: $[X \gamma, i, k|p, q|adj]$ is true, $[Y \gamma, k, j|-, -|adj]$ is true, and $X$ is on the spine of $\gamma$
  - $Y$ has the hole: $[X \gamma, i, k|-,-|adj]$ is true, $[Y \gamma, k, j|p, q|adj]$ is true, and $Y$ is on the spine of $\gamma$
  - neither has the hole: $[X \gamma, i, k|-,-|adj]$ is true, $[Y \gamma, k, j|-, -|adj]$ is true, and neither is on the spine of $\gamma$
Unary Rules and Adjunction

- **Unary rule:** \([N^\gamma, i, j|p, q|false] \) if \([X^\gamma, i, j|p, q|false]\) and \(N^\gamma \rightarrow X^\gamma\).

- **Adjunction rule:** \([N^\gamma, i, j|p, q|true] \) if \([R^\beta, i, j|i', j'|adj]\) and \([N^\gamma, i', j'|p, q|false]\) and \(R^\beta = N\)

- Chart is four-dimensional, and to parse with the adjunction rule, must examine all \(i', j'\) in range.

- Computational complexity is \(O(n^6)\)