1 Restricting the weak GC (and keeping the strong)

- Acquisition is made possible by restricting the space of possible human languages, PHL.

- One hypothesis:
  PHL can be characterized, completely or to a very important degree, as a class of string-sets with a very general mathematical definition—e.g. in automata-theoretic terms
  - Chomsky Hierarchy
    1. Type 3 language
       Regular Grammar: \( A \rightarrow a B \)
       Finite State Automaton
       \( \cdots \)
    2. Type 2 language
       Context Free Grammar: \( A \rightarrow \beta \)
       Pushdown Automaton
       \( \cdots \)
    3. Type 1 language
       Context Sensitive Grammar: \( \alpha A \delta \rightarrow \beta \)
       Linear Bounded Automaton
       \( \cdots \)
    4. Type 0 language
       Unrestricted Grammar: \( \alpha \rightarrow \beta \)
       Turing Machine

- The data require a class of languages somewhere between Types 0 and 2 (noninclusive).

  1. To know that the string set of PHL is a Type 0 language is to know nothing.

  2. Regular Grammars:
     (a) Fail to provide any appropriate descriptions of constituency
     (b) Cannot describe arbitrarily many unbounded dependencies

  3. Context Free Grammars
     (a) Do provide appropriate descriptions of constituency in many if not all cases—namely, in the structure of their derivations

<table>
<thead>
<tr>
<th>Grammar 1 ((G_1))</th>
<th>Derivation of a string in (G_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow NP \ VP )</td>
<td>( S )</td>
</tr>
<tr>
<td>( NP \rightarrow D \ N )</td>
<td>( NP \ VP )</td>
</tr>
<tr>
<td>( VP \rightarrow V \ NP )</td>
<td>the boy pounded NP</td>
</tr>
<tr>
<td>( D \rightarrow ) the</td>
<td>the boy pounded the cutlet</td>
</tr>
<tr>
<td>( N \rightarrow ) boy, cutlet</td>
<td></td>
</tr>
<tr>
<td>( V \rightarrow ) pounded</td>
<td></td>
</tr>
</tbody>
</table>
(b) Can describe arbitrarily many unbounded dependencies—but only if they nest
(and even then you can worry about getting the right structures using a simple string-rewriting grammar; e.g. \(a^n b^k\))

(c) Cannot describe unbounded **crossing** dependencies
Assume that matching letters are dependent: \(abab, ababc, abcdabcd, \ldots\)

(1) Pretend you’re smoking hams in the Alps (Schieber 1985):
   a. Hans Maria the hound saw help
      ‘Hans saw Maria help the hound’
   b. Hans Maria the hound her pups saw help suckle
      ‘Hans saw Maria help the hound suckle her pups’
   c. Hans saw the hound the ewe her pups saw help convince feed
      ‘Hans saw Maria help the hound convince the ewe to suckle her pups’

• Fact: unbounded crossing dependencies do not require the full power of Types 0 or 1.
  They require only “mild” Context Sensitivity, as defined in Joshi 1985.

1. Generates all context-free languages
2. Generates certain dependencies (nesting, crossing) but not all. For example, cannot generate the language with the same number of \(a\)’s as \(b\)’s, etc., in any order.
3. The constraint growth property: If all strings are ordered by length, the difference between any two consecutive lengths is bounded by some constant (thus e.g. the strings do not grow exponentially)
4. Parsable in polynomial time (TAG can be parsed in \(O(n^6)\), worst case.

• All of these formalisms (grammatical metalanguages) are MCSGs, but not proper CSGs:
  1. Pure TAG, and several variants
  2. Combinatory Categorial Grammar (Steedman)
  3. Minimalist Grammar (Stabler)
  4. Linear Indexed Grammars (REF)
  5. Head Grammars (Pollard)

In contrast, grammatical formalisms that incorporate upward transmission of unbounded feature matrices—such as HPSG—have greater GC. Under most instantiations, HPSG is turing equivalent.¹

• The big question: Is mild context sensitivity enough?
  Can MCSGs provide adequate structural descriptions for every construction, in every NL?

• If the answer is Yes, then adopting a MCSG formalism allows us to capture this fact **without stipulations internal to the substantive linguistic analysis.** It just falls out from the choice of metalanguage.

But is the answer Yes??

¹For discussion of the GC of GPSG, HPSG, and LFG see (e.g.) Johnson 1988 (Attribute Value Logic and the Theory of Grammar), Ristad 1990 (L&P13), Carpenter 1991 (CompLing17). [I’ll link the latter two].
2 What is a TAG?

• TAG is a formalism that builds big trees by assembling smaller trees, using the operation(s) of Adjoining and Substitution.

\[
\tau_1 = \begin{array}{c}
\text{NP} \\
\text{VP} \\
\text{laughed}
\end{array}
+ \tau_2 = \begin{array}{c}
\text{VP} \\
\text{ADV} \\
\text{loudly}
\end{array}
+ \tau_3 = \begin{array}{c}
\text{NP} \\
\text{Bob}
\end{array}
= \tau_4 = \begin{array}{c}
\text{NP} \\
\text{Bob} \\
\text{VP} \\
\text{ADV} \\
\text{laughed}
\end{array}
\]

There are two types of elementary trees in a TAG

1. Initial trees
   The frontier of an initial tree comprises terminal symbols, or nonterminal symbols whose label is not that of the root. The latter are substitution nodes, and are conventionally annotated with ‘/’, as a convenience.

2. Auxiliary trees
   Exactly one nonterminal node on the frontier with the same label as the root. This node, called the foot, node is conventionally annotated with ‘∗’.

• Derived object versus Derivation

  – String rewriting grammars
    * A string-rewriting grammar derives strings, by rewriting nonterminal symbols
    * The derivation of a string defines a tree
    * The structure of the derivation tree gives us subtrees that are supposed to be linguistically significant: constituents
    * In the Chomskyan tradition, various linguistic constraints and transformations are stated over the derivation tree—this being what makes the constituent subtrees linguistically significant, under this model

  – Tree adjoining grammars (think of them as ‘tree rewriting grammars’)
    * A TAG derives trees, by rewriting a node in tree α as tree β
    * It is the derived object, a tree, that includes the constituency information which a CFG provides in its derivation trees
    * As it happens, the TAG derivation of a tree itself defines a tree
      
      \[
      \tau_1
      \begin{array}{c}
      \tau_2 \\
      \tau_3
      \end{array}
      \]

      Of course this can’t be where we state things like Binding Theory. It does, however, display dependencies between parts of the derived tree. And it may therefore serve as a graph semantic interpretation.
We can translate CFG rules into trees of depth 1, and assemble them by Substitution into frontier nodes (i.e. those without children). Substitution is rewriting of a node labeled X with a tree whose root is labeled X.

**Grammar $t(G_1)$**

```
S --> NP VP
NP --> D N
VP --> V NP
```

```
D --> the
N --> boy
N --> cutlet
V --> pounded
```

The resulting grammar, $t(G)$, has as its set of derived objects, $L(t(G))$, exactly the set of derivation trees for $L(G)$. Thus each string in $L(G)$ is the yield of some tree in $L(t(G))$, and each tree in $L(t(G))$ has as its yield a string in $L(G)$.

**TAGs are different in two ways:**

1. The elementary trees can have any depth, up to some finite bound

```
S --> NP VP
NP --> V NP
```

2. Adjoining

Any node with label X can be rewritten with an auxiliary X tree.

In the output of Adjoining, the recursive foot node of the adjoined tree dominates exactly the structure dominated by the targeted node in the adjoined-into tree.

```
X
\ldots X \ldots + A \ldots X \ldots = A \ldots X \ldots \ldots X \ldots \\ B \ldots X \ldots \ldots X \ldots B
```

Thus Adjoining splits the target tree in two, and wraps the two resulting parts around some recursive piece of structure.

N.B.: Any node can be marked for which types of trees can be adjoined there: all, some, or none. These are called **Adjoining Constraints**.

**TAGs produce non-CF languages as a result of these two properties interacting.**

1. Unbounded crossing dependencies (though CF string language)

   - Grammar ABX

   $\tau_0 = S \epsilon \quad \tau_\alpha = S_{NA} \quad \tau_0 = S_{NA} a S$

   - Derivation:

   $\tau_0[2] \quad \tau_0[2.1] \quad \tau_\alpha[2]$
In prose:
Adjoin (instance 1) of \( \tau_\alpha \) into (instance 2) of \( \tau_\alpha \) at node 2 = \( \tau_3 \)
Adjoin \( \tau_4 \) into (instance 3) of \( \tau_\alpha \) at node 2 = \( \tau_5 \)
Substitute \( \tau_0 \) into \( \tau_3 \) at node 2.1 = \( \tau_6 \)

- Derived tree
\[
\begin{array}{c}
S \\
\hspace{1cm} a_3 \quad S \\
\hspace{2cm} a_2 \quad S \\
\hspace{3cm} a_1 \quad S \\
\hspace{4cm} S \quad b_1 \\
\hspace{5cm} S \quad b_2 \\
\hspace{6cm} \epsilon \quad b_3 \\
\end{array}
\]

2. String languages that are strictly Context Sensitive

- Grammar Count-4
\[
\begin{array}{c}
\tau_0 = \\
\hspace{1cm} S \\
\hspace{2cm} \epsilon \\
\end{array}
\]
\[
\begin{array}{c}
\tau_\alpha = \\
\hspace{1cm} S_{NA} \\
\hspace{2cm} a \quad S \\
\hspace{3cm} b \quad S_{NA} \\
\hspace{4cm} c \\
\end{array}
\]

- Derivation:
\[
\begin{array}{c}
\tau_0 \quad \tau_3 \\
\tau_4 \quad \tau_0[2.2] \\
\tau_1 \quad \tau_0[2] \\
\end{array}
\]

In prose:
Adjoin (instance 1) of \( \tau_\alpha \) into (instance 2) of \( \tau_\alpha \) at node 2 = \( \tau_4 \)
Adjoin \( \tau_4 \) into (instance 3) of \( \tau_\alpha \) at node 2 = \( \tau_5 \)
Substitute \( \tau_0 \) into \( \tau_3 \) at node 2.2 = \( \tau_6 \)
• Derived tree

\[ \tau_6 = \]

* Tree Substitution Grammar

A TSG allows elementary trees of depth greater than 1, but employs only Substitution.

– This permits structural descriptions that are not otherwise possible. Compare:

1. \[
S \quad S \quad S \\
\quad \quad a \quad S \quad b
\]

2. \[
S \quad S \\
\quad a \quad S
\]

– But the string languages of TSGs remain strictly context free (Type 2). Some examples:

* Grammar ABN

\[ \tau_0 = \begin{array}{c}
S \\
\mid \\
\epsilon
\end{array} \quad \tau_\alpha = \begin{array}{c}
S \\
\quad a \quad S \\
\quad \mid \\
\epsilon
\end{array} \]

* Derivation:

In prose:
Substitute (instance 1) of \( \tau_\alpha \) into (instance 2) of \( \tau_\alpha \) at node 2.1 = \( \tau_4 \)
Substitute \( \tau_4 \) into (instance 3) of \( \tau_\alpha \) at node 2.1 = \( \tau_5 \)
Substitute $\tau_0$ into $\tau_3$ at node $2.1 = \tau_6$

* Derived tree

$\tau_6 =$

```
S
  /\  \\
a_3 /  \ b_3
  / \    \\
a_2 /   \ b_2
  /    \   \\
a_1 /     \  b_1
   /       \   \\
/         \  e
```

* Grammar n-ABCD

$\tau_0 =$

```
S
 /  \
|   e
```

$\tau_\alpha =$

```
S
   /\
  a/  \\
  b S c
```

* Derivation:

```
\tau_3^3
  /  \\
\tau_2[2] \tau_0[2.2] \\
\tau_4[2.2]
```

In prose:
Substitute (instance 1) of $\tau_\alpha$ into (instance 2) of $\tau_\alpha$ at node $2.2 = \tau_4$
Substitute $\tau_4$ into (instance 3) of $\tau_\alpha$ at node $2.2 = \tau_5$
Substitute $\tau_0$ into $\tau_3$ at node $2.2 = \tau_6$
• There are string languages that cannot be generated with a pure TAG. For example, \( \{ w : w = a^n b^n c^n d^n e^n \} \).

• In order to impose a kind of monotonicity on the derivations, it is generally assumed that TAG derivations—not the string yields of the tree language, but the derivations—are “context-free,” in this sense:

Given a mother A and daughter B in a derivation tree, then trees A and B can be combined regardless of what daughters B may (or may not) have.

As we’ll see, Bob Frank makes important linguistic use of this constraint.

• In practice, it is also often—but not always—assumed that nothing can adjoin to the foot of an auxiliary tree. This cuts down on the number of possible derivations for a particular tree.
3 TAGs and lexicalization

- A lexicalized grammar is one where every elementary structure is associated with a terminal symbol.

- TAGs can be seen as ‘arising’ from an attempt to lexicalize a CFG in a linguistically meaningful way—that is, while providing the right constituent structure descriptions.

- Any CFG can be put into Greibach Normal form: \( A \rightarrow a \ B \ C \)
  But derivations in a GNF grammar won’t provide the right constituent structure.

- Provably, the class of CFGs cannot be lexicalized with preservation of the structural descriptions. Putting it differently, Tree Substitution Grammars cannot be strongly lexicalized.

  Example: this grammar cannot be lexicalized: \( S \rightarrow S \ S, \ S \rightarrow a. \)

- More interestingly, lexicalization, even when it succeeds, will often force the wrong choice for the anchor of the tree.
  
  - Nonlexicalized CFG: \( S \rightarrow NP \ VP, \ VP \rightarrow advVP, \ VP \rightarrow v, \ NP \rightarrow n \)
  
  - Two strong lexicalizations, as a TSG:

    1. \[ S \rightarrow NP \ VP \rightarrow advVP \rightarrow v \]

    2. \[ S \rightarrow NP \ VP \rightarrow v \rightarrow advVP \]

    We have to choose either adv or n as the anchor of an S tree.

- Adjoining allows an appropriate choice of anchor:

  \[ S \rightarrow NP \ VP \rightarrow advVP \rightarrow v \rightarrow NP \]

- The TAG literature generally presumes lexicalization. In practice, TAGs are LTAGs. This becomes a very important part of linguistic theories—i.e. theories of languages—that presume TAG derivations.

It is often claimed that lexicalization is linguistically desirable, inasmuch as all idiosyncratic information associated with a particular terminal can be stated within a single elementary tree, anchored by that terminal.

A less tenuous observation is that this makes online parsing more efficient, particular when the elementary trees are associated with probabilities. And lexicalization with TAGs is neater than lexicalization with feature percolation.
3.1 Theoretical perspective

- Frank calls this the “Fundamental TAG hypothesis”:

  Every syntactic dependency is expressed locally within a single elementary tree.

  To the extent that this is possible, it is made possible by the fact that the elementary trees can be deep. This permits an **Extended Domain of Locality** in comparison to most other formalisms.

- Concomitantly, the syntax proper—where big, potentially recursive structures are built—does not traffic in the transmission of dependency information:

  Recursion is separated out (factored) from the statement of dependencies.

  In frameworks like HPSG, it is the ability to transmit dependency information through structures of unbounded size that boosts the GC.

- Thus in TAG, the architecture of the formalism itself defines a domain of locality, that is the exclusive domain for stating grammatical dependencies.

  In post-Aspects theories, including GB and HPSG, such domain restrictions have to be stipulated extrinsically.

- As Bob Frank likes to point out (see Chapter 1 of Frank 2002), TAG bears a greater architectural resemblance to the grammar in LSLT:

  - Elementary trees are *kernel sentences*, the domain of singulary transformations (passive, etc.)
  - Larger trees are built by generalized transformations: Substitution and Adjoining.

4 Linguistic analysis

4.1 Overview

- To state a linguistic theory in TAG is, plain and simple, to list the elementary trees for the language.

- Given lexicalization, this means that any word will most likely be associated with a whole family of trees: e.g., the active tree, the passive tree, the topicalization tree, the question tree, etc.

- Naturally we want a substantive theory of what sorts of elementary trees are possible! But that theory is, strictly speaking, not a part of TAG.
4.2 Raising: Basics

- The TAG analysis for Raising verbs treats them as anchors of auxiliary trees that adjoin to VP—or, in more modern implementations, to $T$.

\[
\begin{align*}
\text{(2) } & \text{a. } \lambda_1 = \\
& \text{b. } \lambda_2 = \\
& \text{c. } \sigma_1 = \\
& \text{d. } \sigma_2 = \\
& \text{e. } \delta_1 = \\
& \text{f. } \delta_2 = \\
\end{align*}
\]

\[
\text{(3) }
\begin{array}{c}
\lambda_2 \\
\sigma_2[2] \quad \delta_1[1] \quad \delta_2[2,2,2] \\
\sigma_1[0]
\end{array}
\]
We want to ensure that infinitival initial trees get adjoined into. One way to do this is to stipulate an Obligatory Adjoining constraint at its T node.

A more expressive and useful way of achieving this effect is by using FTAG:

1. Each node has two sets of features: Top and Bottom. The T features are meant to represent the relations the node has to structure above it; the B features, the structures below.

2. When tree A adjoins to tree B at node n, the top features of A’s root unify with the top features of n, and the bottom features of A’s foot unify with the bottom features of n.

3. At the end of the derivation, the T and B features at each node are collapsed. If they fail to unify, the derivation is bad.

Thus you can force adjoining by giving a node incompatible features:

- Assign the T node of the ‘to like’ tree a +tense top feature and a −tense bottom feature.
- If nothing adjoins into this node, the derived tree will be illformed, because the T and B features will fail to unify.
- But things will work out if a tree adjoins in whose root has T=+tense (and B ≠ −tense), and whose foot has B = −tense (and T ≠ +tense).

This can also be used to handle agreement, etc.

Remarkably, this analysis captures many of the basic locality properties that other accounts have to stipulate independently. (Discuss, time permitting)

4.3 Wh-movement: Basics