Stochastic linear indexed grammars for simple repeat recognition in biological data

M.L. Souza
University of California Berkeley
Biophysics

June 13, 2011

This article explores the use and limitations of stochastic linear indexed grammars for recognition of mobile element-like sequences.

1 Motivation

Context-free grammars (CFGs) are well suited to modeling of arbitrarily complex nestings of dyad symmetry in sequences. However, CFGs are incapable of generating languages of direct repeats; and thus biological languages are frequently beyond context-free. Here we consider the application of extensions to CFGs capable of limited generation of repeats, while preserving polynomial parsing complexity.

In the following, we consider a few motivating examples of repetition in transposon sequences. Let T^* be the Kleene closure of a set of terminal symbols, e.g. $T = \{A, C, G, T\}$, and let $\overline{w}^R$ denote the "inverted repeat" of w. (E.g. for $w = CTGGA$, $\overline{w}^R = TCCAG$.)

1.1 Repeats Flanking Inverted Repeats

Class II DNA transposons have sequence structure consisting of a variable region surrounded by inverted repeats. Such inverted repeats are precisely the dyad structure that CFGs are suited for.
However, flanking the inverted repeats there exist short direct repeats from the process of transposition. The class of languages resembles the following:

$$L_{SRIR} = \left\{ y(wx\overline{w}^R)y \mid w, \overline{w}^R, x, y \in T^* \right\}$$

1.2 LTR Retrotransposons

Class I retrotransposons have a structure as shown in figure 1. Their distinctive structural feature is precisely what makes them unsuited for recognition by CFGs; flanking terminal repeats.

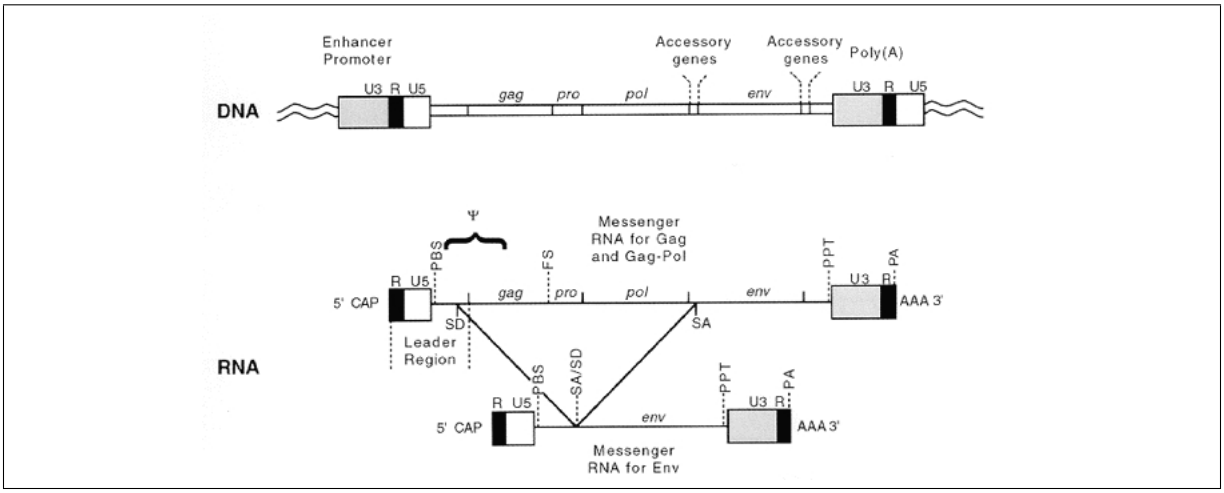


Figure 1: (Figure stolen from: [1] Fix me.)

This class of languages is:

$$Ł_{LTR} = \{wxw \mid w, x \in T^*\}$$

1.3 Insertion Sequences

The form of an insertion sequence has both dyad symmetry and a direct repeat structure, separated by sequences coding for transposition and structural machinery.

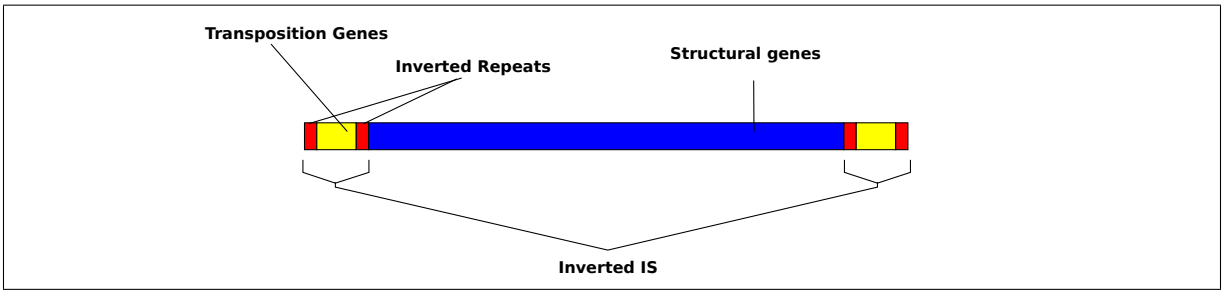


Figure 2: (Figure stolen and modified from wikipedia transposable element page. Fix me.)

The class of languages describing insertion sequences is roughly:

$$Ł_{IS} = \left\{(wx\overline{w}^R)y(wx\overline{w}^R) \mid w, \overline{w}^R, x, y \in T^*\right\}$$

Note that this language is a more structured form of $\mathbb{L}_{LTR}$.

2 Extended Context-Free Grammars

In this section we introduce indexed grammars.

2.1 Review of CFGs

Context-free grammars are defined as a 4-tuple: $G = (N, T, P, S)$ where N is a set of non-terminals; T is a set of terminals; P is a set of production (rewrite) rules; and S a designated

start symbol.
Each production rule maps a single non-terminal symbol to any combination of terminal and non-terminal that is substituted in the LHS non-terminal's place in a derivation.
An example CFG generating perfect inverted repeats in DNA has production rules:

$$S \to aSt \mid tSa \mid cSg \mid gSc \mid \epsilon$$

An example derivation using the above grammar:

$$S \Rightarrow cSg \Rightarrow ctSag \Rightarrow ctgScag \Rightarrow ctggSccag \Rightarrow ctggaStccag \Rightarrow ctggatccag$$

The above example intuitively demonstrates how CFGs are suited to generate nestings of correlated symbols. The process of recognition by a CFG's analogous pushdown automaton provides another interpretation. The symbols of the pushdown stack can only be emitted in a last-in-first-out way; the "memory" of the already-emitted sequence only plays in reverse like a ball rolling back down a hill, producing dyad symmetry. Thus it's the memory accession method for CFGs that results in their being unsuited to direct repetition.

The inability of a CFG to produce a language of repeats can also be shown formally using the Pumping Lemma, which we demonstrate later.

2.2 Indexed Grammars

The Indexed Grammars are context-free grammars extended with stacks of indices associated with each non-terminal.

2.2.1 Definition

Let $G = (N, T, I, S, P)$ where N is a set of non-terminals; T a set of terminals, I a set of index symbols, $S \in N$ a start symbol, and P the set of production rules. Let $A, B, C \in N$ be non-terminal symbols, $\alpha \in (T \cup N)^*$ be a string of terminals/nonterminals, and $i_1, i_2, i_3 \in I$ be index symbols, and $\sigma, \sigma_1, \sigma_2 \in I^*$ be arbitrary (possibly empty) strings of index symbols.

Each non-terminal in a production rule in a string such as α has an independent stack of symbols which can optionally inherit and modify the stack passed on the LHS non-terminal.
We demonstrate stack passage and modification in IG rewrite rules by example:

1. $A[\sigma] \to A[\sigma]A[\sigma\ i_1\ i_2]B$
   Has stack $[\sigma]$ is passed to the left-most A, $[\sigma]$ passed and extended with symbols i_1, i_2 to the middle A, and the stack of B unmodified.

2. $A[\sigma\ i_1\ i_2\ i_3] \to B[\sigma]$
   Has i_1, i_2, i_3 removed from A's stack, and the remainder passed to B.

2.2.2 Example: L_{IS}

Consider the grammar $G_{IS} = (N, T, I, S, P)$ defined as follows: $T = \{a, c, g, t\}$, $N = \{C,\ X,\ Y,\ R_1\ R_2\ R_3\}$, $I = \{C_a,\ C_g,\ C_c,\ C_t,\ X_a,\ X_g,\ X_c,\ X_t\}$, $S = C$.

Define P with the following rules:

1. $C[\sigma] \rightarrow aC[\sigma\ C_a] \mid gC[\sigma\ C_g] \mid cC[\sigma\ C_c] \mid t\ C[\sigma\ C_t] \mid X[\sigma]$
2. $X[\sigma] \rightarrow aX[\sigma\ X_a] \mid gX[\sigma\ X_g] \mid cX[\sigma\ X_c] \mid t\ X[\sigma\ i_t] \mid R_1[\sigma]\ Y[]\ R_2[\sigma]\ R_1[\sigma]$
3. $Y[\sigma] \rightarrow aY[\sigma] \mid gY[\sigma] \mid cY[\sigma] \mid tY[\sigma] \mid \epsilon$
4. $R_1[\sigma\ C_a] \rightarrow tR_1[\sigma],\ R_1[\sigma\ C_g] \rightarrow cR_1[\sigma],\ R_1[\sigma\ C_c] \rightarrow gR_1[\sigma],\ R_1[\sigma\ C_t] \rightarrow aR_1[\sigma]$
5. $R_1[\sigma\ X_a] \rightarrow R_1[\sigma],\ R_1[\sigma\ X_g] \rightarrow R_1[\sigma],\ R_1[\sigma\ X_c] \rightarrow R_1[\sigma],\ R_1[\sigma\ X_t] \rightarrow R_1[\sigma]$
6. $R_1[] \rightarrow \epsilon,\ R_2[] \rightarrow \epsilon$
7. $R_2[\sigma\ C_a] \rightarrow R_2[\sigma]a,\ R_2[\sigma\ C_g] \rightarrow R_2[\sigma]g,\ R_2[\sigma\ C_c] \rightarrow R_2[\sigma]c,\ R_2[\sigma\ C_t] \rightarrow R_2[\sigma]t$
8. $R_2[\sigma\ X_a] \rightarrow R_2[\sigma]a,\ R_2[\sigma\ X_g] \rightarrow R_2[\sigma]g,\ R_2[\sigma\ X_c] \rightarrow R_2[\sigma]c,\ R_2[\sigma\ X_t] \rightarrow R_2[\sigma]t$

This grammar generates $\mathbb{L}_{IS}$.
For example, the following is a derivation of the sequence ACG T CGT A ACG T CGT:

$$
\begin{aligned}
C &\overset{1.1}{\Rightarrow} aC[C_a] \overset{1.3}{\Rightarrow} acC[C_a, C_c] \overset{1.2}{\Rightarrow} acgC[C_a, C_c, C_g] \\
&\overset{1.5}{\Rightarrow} acg\ X[C_a, C_c, C_g] \overset{2.4}{\Rightarrow} acgtX[C_a, C_c, C_g, X_t] \\
&\overset{2.5}{\Rightarrow} acgt\ R_1[C_a, C_c, C_g, X_t]\ Y[]\ R_2[C_a, C_c, C_g, X_t]\ R_1[C_a, C_c, C_g, X_t] \\
&\overset{3.1}{\Rightarrow} acgt\ R_1[C_a, C_c, C_g, X_t]\ aY[]\ R_2[C_a, C_c, C_g, X_t]\ R_1[C_a, C_c, C_g, X_t] \\
&\overset{3.5}{\Rightarrow} acgt\ R_1[C_a, C_c, C_g, X_t]\ a\ R_2[C_a, C_c, C_g, X_t]\ R_1[C_a, C_c, C_g, X_t] \\
&\overset{5.4\ x2}{\Rightarrow} acgt\ R_1[C_a, C_c, C_g]\ a\ R_2[C_a, C_c, C_g, X_t]\ R_1[C_a, C_c, C_g] \\
&\overset{4.2\ x2}{\Rightarrow} acgt\ c\ R_1[C_a, C_c]\ a\ R_2[C_a, C_c, C_g, X_t]\ cR_1[C_a, C_c] \\
&\overset{4.3\ x2}{\Rightarrow} acgt\ cg\ R_1[C_a]\ a\ R_2[C_a, C_c, C_g, X_t]\ cgR_1[C_a] \\
&\overset{4.1\ x2}{\Rightarrow} acgt\ cgt\ R_1[]\ a\ R_2[C_a, C_c, C_g, X_t]\ cgtR_1[] \\
&\overset{6.1\ x2}{\Rightarrow} acgtcgta\ R_2[C_a, C_c, C_g, X_t]\ cgt \\
&\overset{8.4}{\Rightarrow} acgtcgta\ R_2[C_a, C_c, C_g]\ t\ cgt \\
&\overset{7.2}{\Rightarrow} acgtcgta\ R_2[C_a, C_c]\ g\ tcgt \\
&\overset{7.3}{\Rightarrow} acgtcgta\ R_2[C_a]\ cg\ tcgt \\
&\overset{7.1}{\Rightarrow} acgtcgta\ R_2[]\ acg\ tcgt \\
&\overset{6.2}{\Rightarrow} acgtcgtaacgtcgt
\end{aligned}
$$

The indexed languages are a proper subset of context-sensitive languages, but the general recognition problem for the indexed grammars is NP-complete. Several forms of indexed grammars exist which maintain parsing efficiency by restricting the way in which the stack can be inherited.

2.3 Linear Indexed Grammars

2.3.1 Definition

In **linear indexed grammars** (LIGs) [(defined in 1988 by Gazdar; can't find a copy of the PDF. Fix me.), only one non-terminal can inherit the stack in a rewrite rule.

Let $\{w_1, w_2, ..., w_{n+1}\} \in T^*$, $\{A_1, \cdots, A_n, A\} \in N$, and $\{\alpha_1, \alpha_2, \cdots, \alpha_n\} \in I^*$

The general form for LIG rewrite rules is:

$$A[\sigma] \rightarrow w_1 A_1[\alpha_1] w_2 A_1[\alpha_2] \cdots w_i A_i[\cdot\, \alpha_i] \cdots w_n A_n[\alpha_n] w_{n+1}$$

The non-terminal $A_i[\cdot\cdot\alpha_i]$ in the above rule is the one that inherits the stack, and is known as the **distinguished child of** $A[\cdot\, \alpha]$. In this document when the distinguished child non-terminal is apparent we omit the "$\cdot\cdot$" in denoting the stack $[\cdot\, \sigma]$.

2.3.2 Examples

Note that in the above example of an indexed grammar for L_{IS}, rule 2.5 violates the LIG requirement since the stack is inherited by three non-terminals. We later consider whether LIGs can produce this language.

Consider the language L_{LTR}.
This language is considerably less structured than L_{IS}, and can be generated with the following grammar $G_{LTR} = (N, T, I, S, P)$ defined as follows:

$T = \{a, c, g, t\}$, $N = \{C,\ X,\ C_R\}$, $I = \{a,\ g,\ c,\ t\}$, $S = C$.
Define P with the following rules:

1. $C[\sigma] \rightarrow aC[\sigma\ C_a] \mid gC[\sigma\ C_g] \mid cC[\sigma\ C_c] \mid t\ C[\sigma\ C_t] \mid X[\sigma]$
2. $X[\sigma] \rightarrow aX[\sigma] \mid gX[\sigma] \mid cX[\sigma] \mid t\ X[\sigma] | C_R[\sigma]$
3. $C_R[\sigma\ a] \rightarrow C_R[\sigma]a,\ C_R[\sigma\ g] \rightarrow C_R[\sigma]g,\ C_R[\sigma\ c] \rightarrow C_R[\sigma]c,\ C_R[\sigma\ t] \rightarrow C_R[\sigma]t$
4. $C_R[] \rightarrow \epsilon$

The sequence $ACGT\ TC\ ACGT$ can be derived as follows:
$C \overset{1.1}{\Rightarrow} aC[a] \overset{1.3}{\Rightarrow} acC[a, c] \overset{1.2}{\Rightarrow} acgC[a, c, g] \overset{1.2}{\Rightarrow} acgtC[a, c, g, t]$
$\overset{1.5}{\Rightarrow} acgt\ X[a, c, g, t]$
$\overset{2.3}{\Rightarrow} acgt\ \overset{1.4}{\Rightarrow} acgt\ tX[a, c, g, t]\ \overset{1.3}{\Rightarrow} acgt\ tcX[a, c, g, t]\ \overset{1.5}{\Rightarrow} acgt\ tcC_R[a, c, g, t]$
$\overset{3.4}{\Rightarrow} acgttc\ C_R[a, c, g]\ t\ \overset{3.2}{\Rightarrow} acgttc\ C_R[a, c]\ gt\ \overset{3.3}{\Rightarrow} acgttc\ C_R[a]\ cgt \overset{3.1}{\Rightarrow} acgttc\ C_R[]\ acgt$
$\overset{4.1}{\Rightarrow} acgttcacgt$

Next, we consider the language L_{SRIR}.
Consider the grammar $G_{SRIR} = (N, T, I, S, P)$ defined as follows:

$T = \{a, c, g, t\}$, $N = \{Y,\ X, W,\ Y_R\}$, $I = \{a,\ g,\ c,\ t\}$, $S = Y$.
Define P with the following rules:

1. $Y[\sigma] \rightarrow aY[\sigma\ a] \mid gY[\sigma\ g] \mid cY[\sigma\ c] \mid t\ Y[\sigma\ t] \mid C\ Y_R[\sigma]$
2. $C[] \rightarrow aC[\sigma]t \mid tC[\sigma]a \mid cC[\sigma]g \mid gC[\sigma]c \mid X[\sigma]$
3. $X[] \rightarrow aX[\sigma] \mid gX[\sigma] \mid cX[\sigma] \mid t\ X[\sigma] \mid \epsilon$
4. $Y_R[\sigma\ a] \rightarrow Y_R[\sigma]a,\ Y_R[\sigma\ g] \rightarrow Y_R[\sigma]g,\ Y_R[\sigma\ c] \rightarrow Y_R[\sigma]c,\ Y_R[\sigma\ t] \rightarrow Y_R[\sigma]t$

5. $Y_R[] \rightarrow \epsilon$

and consider the derivation of $AA\ CGAC\ T\ GTCG\ AA$: (Fix me.)

2.4 Going Probabilistic

Strict duplication as we generate in G_{IS}, and G_{LTR} is insufficient to describe real data due to the imperfection of repetition seen in nature. Such "imperfection" sometimes isn't just tolerated error, but is essential for biological activity. [2] [3]

To address this, we construct grammars with patterns accommodating the exceptions but we adjust for indiscriminate false-positives by associating probabilities with emissions.

We assign probabilities (weights) to each production rule of the grammar such that the sum of the weights of all productions from any single non-terminal is 1,
i.e. $\sum_{\alpha} P(X \rightarrow \alpha) = 1$ for all rules $(X \rightarrow \alpha) \in P$.

The grammar G produces a probability distribution over its emitted strings s: $\sum_{s} P(s \mid G) = 1$.

We're interested in ways to address the following problems:

1. **Scoring**: Calculate $P(s \in T^* \mid G)$
   Given a stochastic grammar, calculate the probability of it having emitted a sentence.
2. **Training**: W s.t. $E\left[P(t_1, t_2, \cdots, t_m \mid G)\right]$ is maximal.
   (Exposition here about how probabilities are defined/refined) Given a set of positive-matching examples, we want to re-estimate the production weight distribution such that the probabilities of emitting the example sequence are maximized.

Below we consider how LIGs are extended to become probabilistic. Later in the section on parsing, we consider methods to address the above two problems.

2.4.1 Stochastic LIGs

We define a stochastic LIG (SLIG) by a 6-tuple: $G = (N, T, I, S, P, W)$ where N is a set of non-terminals; T a set of terminals, I a set of index symbols, $S \in N$ a start symbol, P the set of production rules, and $W : P \rightarrow \mathbb{R}$ is a rule weight function.

In defining W over the set of productions, we incorporate the stack into the probability definition with: $\sum_{\alpha} P(X[\cdot\, i] \rightarrow \alpha) = 1$ for all rules $(X[\cdot\, i] \rightarrow \alpha) \in P$.

As an example, we modify the grammar G_{LTR} above to generate imperfect repeats:
Consider the grammar $SG_{LTR} = (N, T, I, S, P, W)$ defined as follows:

$T = \{a, c, g, t\}$, $N = \{C,\ X,\ C_R\}$, $I = \{a,\ g,\ c,\ t\}$, $S = C$.
Define P, W with the following rules: (W defined by listing rule probabilities to the right)

1. (a) $C[\sigma] \rightarrow aC[\sigma\ C_a]$ **0.2075**

(b) $C[\sigma] \rightarrow aC[\sigma\ C_g]$ **0.01**
(c) $C[\sigma] \rightarrow aC[\sigma\ C_c]$ **0.01**
(d) $C[\sigma] \rightarrow aC[\sigma\ C_t]$ **0.01**
(e) $C[\sigma] \rightarrow gC[\sigma\ C_a]$ **0.01**
(f) $C[\sigma] \rightarrow gC[\sigma\ C_g]$ **0.2075**
(g) $C[\sigma] \rightarrow gC[\sigma\ C_c]$ **0.01**
(h) $C[\sigma] \rightarrow gC[\sigma\ C_t]$ **0.01**
(i) $C[\sigma] \rightarrow cC[\sigma\ C_a]$ **0.01**
(j) $C[\sigma] \rightarrow cC[\sigma\ C_g]$ **0.01**
(k) $C[\sigma] \rightarrow cC[\sigma\ C_c]$ **0.2075**
(l) $C[\sigma] \rightarrow cC[\sigma\ C_t]$ **0.01**
(m) $C[\sigma] \rightarrow tC[\sigma\ C_a]$ **0.01**
(n) $C[\sigma] \rightarrow tC[\sigma\ C_g]$ **0.01**
(o) $C[\sigma] \rightarrow tC[\sigma\ C_c]$ **0.01**
(p) $C[\sigma] \rightarrow tC[\sigma\ C_t]$ **0.2075**
(q) $C[\sigma] \rightarrow X[\sigma]$ **0.05**

2. (a) $X[\sigma] \rightarrow aX[\sigma]$ **0.2375**
(b) $X[\sigma] \rightarrow gX[\sigma]$ **0.2375**
(c) $X[\sigma] \rightarrow cX[\sigma]$ **0.2375**
(d) $X[\sigma] \rightarrow tX[\sigma]$ **0.2375**
(e) $X[\sigma] \rightarrow C_R[\sigma]$ **0.05**

3. (a) $C_R[\sigma\ a] \rightarrow C_R[\sigma]a$ **1.0**
(b) $C_R[\sigma\ g] \rightarrow C_R[\sigma]g$ **1.0**
(c) $C_R[\sigma\ c] \rightarrow C_R[\sigma]c$ **1.0**
(d) $C_R[\sigma\ t] \rightarrow C_R[\sigma]t$ **1.0**
(e) $C_R[] \rightarrow \epsilon$ **1.0**

In rule 1.a to 1.l above, the majority of weight is given to canonical base-pairing, and low (1%) probability to non-canonical pairings. Also note in rule 3.a to 3.e how each emission has unit mass due to the association of the matching index with the non-terminal; in a SCFG, the mass would be distributed amongst each of the 5 rules having non-terminal C_R on the LHS.

We discuss the methods for calculating probabilities of interest below in the parsing section. (PUT A BETTER INTERNAL REFERENCE HERE. Fix me.)

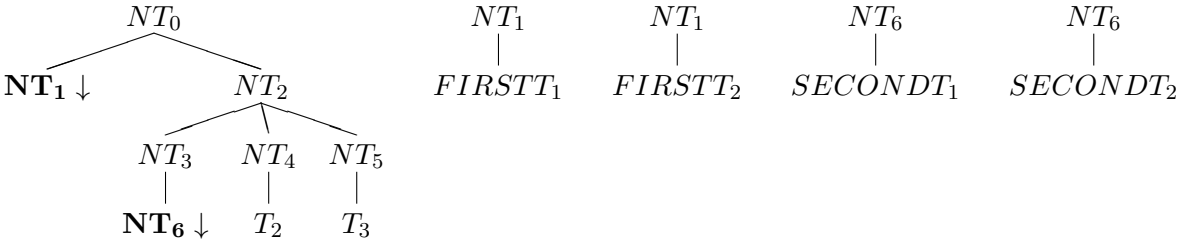
2.5 Equivalent Formalisms

2.5.1 Tree Substitution Grammars

We first consider the formalism of **Tree Substitution Grammars** (TSGs).

Tree substitution grammars allow for the explicit statement of the structure of parse trees which context-free grammars ultimately produce. But unlike the rule rewriting of a CFG, the elementary operation is the substitution of one elementary sub-tree of a parse into another at designated locations.
In this way, explicit relationships can be modeled without having to resort to specialized constraints within rule rewriting.

For instance in the following TSG:



Tree substitution grammars and CFGs have the same weak generative capacity, thus they're not "extensions" of CFGs. However in their weak equivalence to CFGs they serve as a stepping stone to understanding how the next class of grammars is an extension of generative capacity.

2.5.2 Tree Adjoining Grammars

If we begin with a TSG, and add the ability to embed one elementary tree within another (in a process known as "adjoining") then we arrive at the formalism of **Tree Adjoining Grammars**.
Write about weak/strong generative capacity equivalence to LIGs.
Write about going from TAG ¡-¿ LIG.
Fix me.

2.5.3 The Other Formalisms

Fix me.

3 The Limitations of Grammars

We now consider the inherent limitations the formalism we adopt imposes on the complexity of the sequences we can match.

3.1 Pumping Lemmas

In section 2.1, the inability of CFGs to generate a language of copies was intuitively discussed. The **pumping lemmas** provide tools with which we can prove this and similar claims more rigorously. They are a collection of theorems that give necessary (but not sufficient) conditions that languages must satisfy to exist within a given class. In practice they can provide a way to show when a grammar **can't** generate a language.

3.1.1 Pumping Lemma for Regular Grammars

As the complexity of the formalism increases so does the pumping argument machinery, and so for the sake of pedagogy, we begin with regular grammars and their inability to produce a characteristically context-free language: palindromes.

3.1.2 Pumping Lemma for CFGs

The Pumping Lemma for Context-Free Languages is as follows:

Bar-Hillel Theorem 1.
if $(\mathbb{L}$ *is Context Free*$)$ ***then***
Pumping property must hold :
$\exists\ p \in \mathbb{Z}$ *s.t.*
$\forall s \in \mathbb{L}\,(with\ \ |s| \geq\ p,\ s = uvxyz,\ |vxy| \leq\ p,\ \&\ \ |vy| \geq 1)$:
$\forall n \in \mathbb{N} \cup \{0\}\,, u(v^n)x(y^n)z \in \mathbb{L}$
end if

We can use this theorem to show that no CFG can produce language L_{LTR}:

Proof. Suppose thm. 1 holds for L_{LTR}, and let $p \in \mathbb{Z}$ satisfy the theorem.
Denote the substring of string w from position j to k (indexed at 0) as $w_{j,k}$.

Let $s \in \mathbb{L}_1$, $s = wxw$ where $w, x \in N^*$, $|x| < p - 1$, and let $j = \left\lfloor \frac{p-(|x|+1)}{2} \right\rfloor$,

Define substrings $u, v, y, z \in N^*$ as follows: $u = w_{0,|w|-j-1}$, $v = w_{|w|-j,|w|}$, $y = w_{0,j+1}$, and $z = w_{j+2,|w|}$, giving $s = uvxyz$ with $uv = w = yz$.

To see that the requirements are satisfied with this partitioning, note that $|vxy| = |x|+2j+1 = |x| + 2\left\lfloor \frac{p-(|x|+1)}{2} \right\rfloor \leq |x| + 2\left(\frac{p-(|x|+1)}{2}\right) = p$, and $|vy| = 2j + 1 = 2\left\lfloor \frac{p-(|x|+1)}{2} \right\rfloor + 1 \geq 1$.

If any terminal symbol of w at position i such that $0 \leq i < |w| - j$ is different from the terminal at position $j + i$, then for $n = 0$, we have $u(v^n)x(y^n)z = uxz$, but $u \neq z$ by construction and so $u(v^n)x(y^n)z \notin L_{LTR}$.

□

3.1.3 Pumping Lemma beyond CFGs

Summarize Weir's hierarchy beyond CFGs here [4]. Fix me.
(...)LIGs are level-2 controls grammars.
The generalized pumping lemma for a level-k control grammar is as follows:

Generalized Pumping Theorem 1. *Fill me in. Fix me.*

We'd like to see if the language L_{IS} can't be generated by a LIG. We showed that it was indexed language by construction; but could we have done it by passing a single stack?

Proof. Fill me in. Fix me. □

4 Parsing Algorithms for LIGs

We now consider parsing with linear indexed grammars. In this section we draw heavily from the presentation found in chapter 3 of [5], and also from [6], [7] and [8].

Parsing of an input string consists of determining a derivation which generates the input from the initial symbol of the grammar. To specify a parsing technique, we have the option of either describing the algorithm in pseudo-code, or by providing deduction rules for a parsing schema.

A **parsing schema** consists of:

1. **Representation of Items**: A format in which intermediate results (items) are represented.
2. **Deduction Rules**: The specification of rules for how new results progress from existing ones. To specify deduction rules, we adopt the following notation:

   $$\textbf{Rule Name:}\ \frac{\text{Antecedent}}{\text{Consequent}}\ \text{Side-Conditions}$$

   Where "Antecedent" is a listing of the result items that must already exist, "Side-Conditions" must be satisfied in addition to the antecedent item existence, and "Consequent" are the items which should be produced when the requirements of antecedent and side-conditions are met.
   Rules having empty antecedents are known as **axioms**.
3. **Goal**: The end result of a parse.

For a non-stochastic grammar, finding a parse is also known as **recognition**; the input string is regarded as "grammatical" with respect to that grammar when we can find a parse that generates it. For stochastic grammars, in which we explicitly allow for exceptional emissions, we are interested in determining all possible parses (**the parse forest**) such that we can build up the total probability of emitting the input string. Since different parses can often share sub-parses, a dynamic programming approach for reuse of already-computed results can significantly boost computational efficiency.

Chart parsing is the linguistic term for such a dynamical programming approach. The general process of chart parsing is to first apply all rules not having antecedents (axioms) in order to produce a set of initial items, and to then apply the rest of the deduction rules until we exhaust all possibilities. As rules are applied and items produced, items are stored within the chart in addition to pointers to each item's previous forms (other more primitive items). This allows one to back-trace through the entire parse-forest upon the algorithm's completion.

4.1 Review of CFG Parsing

On the road to a CYK-like algorithm for LIG parsing, we present the significantly simpler CYK algorithm for CFGs which it extends.

Durbin disambiguation: Note that in [9], the name "CYK" is used in the context of SCFG

parsing for the best-path variant of the "Inside algorithm"; where as in the presentation below we present a non-stochastic form first which traverses all possible parses, and stores back-references which can be used to calculate the best-parse (as well as any other) probability. Elsewhere the exhaustive chart parsing is called "CYK-like". Need to adopt some consistent naming here. Fix me.

Suppose we have a CFG having non-terminals $A, B, C \in N$, each production rule for the grammar must be in"Chomsky-normal form"; meaning each production rule either emits a single terminal ($A \rightarrow t$, for $t \in T \cup \{\epsilon\}$), or has the form $A \rightarrow B\ C$.

Regarding Chomsky normal form: As is discussed in chapter 10 (p. 273 Fix me.) of [9], the normal form adopted in an implementation need not be restricted to Chomsky normal form; parsing algorithms can be adapted and made much more efficient by adopting specialized normal forms for productions that are more appropriate for the grammar at hand.
For instance, (Put some example refs here. Fix me.)
We discuss a customized normal form appropriate to LIG parsing of transposons in (PUT INTERNAL REFERENCE HERE. Fix me.).

The CYK algorithm for CFGs is a non-directional, bottom-up parsing approach.
Suppose we have input string $a = a_1 a_2 \cdots a_n$ we wish to derive from Chomsky-normal CFG G. The algorithm has two phases, an initialization phase, and an inductive phase. In the initialization phase, the individual characters of the input string are mapped to non-terminals which emit single terminals. In the inductive phase, by considering every span length $i \in \{2, \cdots, n\}$, every starting index $j \in \{1, 2, \cdots, n\}$, and every way to split a span into two parts (into partitions of size k and $k - i$, $k \in \{1, \cdots, i - 1\}$) productions of the form $A \rightarrow B\ C$ are sought such that B generates the left part, and C the right part. For any matching B and C, one knows that A generates the span $a_i a_{i+1} \cdots a_{i+k-1} + a_{i+k} a_{i+k+1} \cdots a_{i+j}$ previously generated by B and C, and thus in subsequent iterations longer spans of the input string are built-up using A on the right-hand side of productions such as in $D \rightarrow A\ B$.

More formally the CYK parsing schema is:

Representation:
Let $P[i, j, A]$ be a boolean, initially false, that is true when the substring $a_i a_{i+1} \cdots a_{i+j-1}$ can be generated by non-terminal $A \in N$.

Deduction Rules:

1. **Initialization:** $\frac{(\text{Axiom})}{[i,1,A]}\ \ (A\ \rightarrow\ a_i) \in P,\ i \in \{1, 2, \cdots, n\}$

2. **Induction:** $\frac{[j,k,B],\ [j+k,i-k,C]}{[i,k,A]}\ \ (A\ \rightarrow\ B\ C) \in P$

Goal: $[0,\ n,\ S] = \text{True}$

4.2 Vijay-Shanker and Weir's CYK-like Algorithm for LIGs

The CYK strategy as described in the previous section has been extended by Vijay-Shanker and Weir to LIG parsing. [8].

In [8], the following normal form for LIGs is adopted:

1. $A[\alpha] \rightarrow c$ where $c \in T \cup \{\epsilon\}$
2. $A[\cdot\cdot\gamma_1\gamma_2\cdots\gamma_m] \rightarrow A_p[\cdot\cdot\gamma_p]$
3. $A[\cdot\cdot\gamma_1\gamma_2\cdots\gamma_m] \rightarrow A_p[\cdot\cdot\gamma_p]\ A_s[\alpha_s]$ where $m \geq 0$.
4. $A[\cdot\cdot\gamma_1\gamma_2\cdots\gamma_m] \rightarrow A_s[\alpha_s]\ A_p[\cdot\cdot\gamma_p]$ where $m \geq 0$.

We call the distinguished non-terminal A_p the **primary constituent**, and the other non-terminal, A_s, the **secondary constituent**.
This is similar to Chomsky normal form, except in a LIG it matters whether the non-terminal A_p is on the left or right in the RHS of a production rule.

The basic intuition behind chart parsing for a LIG in this form is very similar to what was done for CFGs above. We begin by applying axioms in which single-terminal-emitting non-terminals are mapped to the input string to create an initial set of parse items, and we then attempt to chain together additional non-terminals inductively. However, having growing stacks of symbols associated with the non-terminals that derive substrings of the input makes for additional complexity.
Notably, the cardinality of the set of stack-symbol sequences which derive a substring from a non-terminal can grow exponentially with the size of the substring; i.e. we have a worst-case scenario of exponential memory complexity if all of those derivations are simply stored along with the non-terminal symbol. To get around this, the redundant nature of last-in first-out stack system is exploited; rather than storing all of the symbols we store pointers to existing stack sequences, and note when the stack is updated somehow. This pointer generation and shuffling to avoid a blow-up in memory complexity obfuscates the presentation of the algorithm, but is necessary. I.e. we must worry about back-references of two varieties: to previous items in order to efficiently describe the parse forest, and to previous stack-symbol sequences for tractable chart memory consumption.

4.2.1 Stochastic Parsing in LIGs

Fix me.

5 Application to Identification of Mobile Genetic Elements

5.1 Specialized LIG Parsing

As was discussed earlier, the normal form for production rules in a grammar (...) Fix me.

References

[1] Editors. Coffin JM, Hughes SH, Varmus HE. Genetic Organization, 1997.

[2] Eva M Strawbridge, Gary Benson, Yevgeniy Gelfand, and Craig J Benham. The distribution of inverted repeat sequences in the Saccharomyces cerevisiae genome. *Current genetics*, pages 321–340, May 2010.

[3] E Costello, R Sahli, B Hirt, and P Beard. The mismatched nucleotides in the 5'-terminal hairpin of minute virus of mice are required for efficient viral DNA replication. *Journal of virology*, 69(12):7489–96, December 1995.

[4] D Weir. A geometric hierarchy beyond context-free languages. *Theoretical Computer Science*, 104(2):235–261, October 1992.

[5] Laura Kallmeyer. *Parsing beyond context-free grammars*. Cognitive Technologies. Springer Verlag, Berlin, Heidelberg, 2010.

[6] Klaas Sikkel. How to compare the structure of parsing algorithms. In *In: Pighizzini G., San Pietro P.(Eds.), Proc. ASMICS Workshop on Parsing Theory*, pages 1–19. Citeseer, 1994.

[7] K Sikkel. Parsing schemata and correctness of parsing algorithms. *Theoretical Computer Science*, 199(1-2):87–103, June 1998.

[8] K. Vijay-Shanker and D.J. Weir. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636, 1993.

[9] R. Durbin. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*, volume 64. Cambridge Univ Pr, April 1998.