Conversions between *MCFG* and *D*

Logical Characterizations of the Mildly Context-Sensitive Languages

Gijs Wijnholds

Cool Logic, 21th of February 2014

Introduction

- Natural language exhibits patterns that are provably beyond the context-free boundary,
- Research into formal grammar resulted in the definition of the so called Mildly Context Sensitive Languages,
- Different extensions of Context Free formalisms have been proposed,
- We show that three of these systems are 'equivalent'.

Outline

Formal Grammar

Definition

A Formal Grammar is a quadruple (N, Σ, R, S) where:

- N is a finite set of non-terminal symbols,
- Σ is a finite set of terminal symbols,
- R is a set of rewrite rules of the form $(N \cup \Sigma)^* N (N \cup \Sigma)^* \to (N \cup \Sigma)^*$,
- $S \in N$ is a distinguished start symbol.

Definition

Let $G = (N, \Sigma, R, S)$ be a formal grammar. The string language of G, denoted $\mathcal{L}(G)$, is defined as follows:

$$\mathcal{L}(G) := \{w \in \Sigma^* | S \rightarrow^* w\}$$

Definition

Let G and G' be Formal Grammars. G and G' are said to be (weakly) equivalent iff $\mathcal{L}(G) = \mathcal{L}(G')$.

The Chomsky Hierarchy

Putting different restrictions on the rules results in different language classes, with accompanying complexity results:

The Chomsky Hierarchy

Putting different restrictions on the rules results in different language classes, with accompanying complexity results:

Language class	Restriction	Automaton
Regular	$A \to a; A \to aB$	FSA
Context Free	$A \to \gamma$	PDA
Context Sensitive	$\alpha A\beta \to \alpha\gamma\beta, \gamma \neq \epsilon$	LBA
Recursively Enumerable	$\alpha \to \beta$	TM

The Chomsky Hierarchy

Putting different restrictions on the rules results in different language classes, with accompanying complexity results:

Language class	Restriction	Automaton
Regular	$A \to a; A \to aB$	FSA
Context Free	$A \to \gamma$	PDA
Context Sensitive	$\alpha A \beta \to \alpha\gamma\beta, \gamma \neq \epsilon$	LBA
Recursively Enumerable	$\alpha \to \beta$	TM

$$RL \subset CFL \subset CSL \subset REL$$

Example of a Context Free Grammar for palindromes over three symbols:

$$
\begin{aligned}
S &\rightarrow aSa \\
S &\rightarrow bSb \\
S &\rightarrow cSc \\
S &\rightarrow \epsilon
\end{aligned}
$$

Example of a Context Free Grammar for palindromes over three symbols:

$$S \rightarrow aSa$$
$$S \rightarrow bSb$$
$$S \rightarrow cSc$$
$$S \rightarrow \epsilon$$

Example derivation:

$$S \rightarrow aSa \rightarrow acSca \rightarrow acbSbca \rightarrow acbbca$$

Next to generative grammar, another type of grammar formalism was developed: Categorial Grammar.

- A categorial grammar consists of a lexicon and a proof system,
- The lexicon assigns types to elements of the alphabet,
- The proof system governs grammaticality.
- Prototypical example: the Lambek Calculus (Logic of Concatenation)

Definition

Let T be a set of atomic types. Then the set T^* of categorial types is defined as follows:

- If $A \in T$, then $A \in T^*$,
- If $A, B \in T^*$, then $A \bullet B, B/A, A\backslash B \in T^*$.

Definition

A Lambek grammar is a triple (Σ, δ, S) where:

- Σ is a set of words,
- $\delta \subseteq \Sigma \times T^*$ is a type assignment relation,
- $S \in T^*$ is a distinguished start symbol.

Proof Theory of L

$$\frac{\delta(\alpha) = A}{\alpha : A}\ Lex. \qquad \frac{}{0 : I}\ Ax.I \qquad \frac{}{1 : J}\ Ax.J$$

$$\frac{\alpha : A \quad \beta : B}{\alpha + \beta : A \bullet B}\ I\bullet \qquad \frac{\gamma : A \bullet B \quad \begin{matrix}\alpha : A \quad \beta : B\\ \vdots \\ \Delta\langle\alpha + \beta\rangle : C\end{matrix}}{\Delta\langle\gamma\rangle : C}\ E\bullet$$

$$\frac{\begin{matrix}\alpha : A\\ \vdots \\ \alpha + \gamma : B\end{matrix}}{\gamma : A\backslash B}\ I\backslash \qquad \frac{\alpha : A \quad \gamma : A\backslash B}{\alpha + \gamma : B}\ E\backslash \qquad \frac{\begin{matrix}\alpha : A\\ \vdots \\ \gamma + \alpha : B\end{matrix}}{\gamma : B/A}\ I/ \qquad \frac{\gamma : B/A \quad \alpha : A}{\gamma + \alpha : B}\ E/$$

A Lambek grammar for (non-empty) palindromes:

$a : A$	$b : B$	$c : C$
$a : S/A$	$b : S/B$	$c : S/C$
$a : (S/A)/S$	$b : (S/B)/S$	$c : (S/C)/S$

A Lambek grammar for (non-empty) palindromes:

$a : A$	$b : B$	$c : C$
$a : S/A$	$b : S/B$	$c : S/C$
$a : (S/A)/S$	$b : (S/B)/S$	$c : (S/C)/S$

Example derivation:

$$\dfrac{\dfrac{a : (S/A)/S \quad \dfrac{b : S/B \quad b : B}{bb : S}}{abb : S/A} \quad a : A}{abba : S}$$

- Context Free Grammar and Lambek Calculus are weakly equivalent (Pentus)
- If you consider only first-order types, the conversions are not too complicated...
- ... but Pentus' proof is quite tedious!

Context Free Grammar is provably inadequate for natural language:

- ... dat Jan Marie Henk zag leren lopen.
- Can be translated into $\{a^n b^m c^n d^m | n, m \geq 1\}$ or $\{w^2 | w \in \Sigma^*\}$ (Shieber)
- These languages are not Context Free! Can be shown by the pumping lemma.
- So we want to move beyond Context Free.
- However, Context Sensitive is too general...

Mild Context Sensitivity

Introduced by Joshi in 1985, a class of languages $\mathcal{L}$ is Mildly Context Sensitive iff:

- $\mathcal{L}$ contains the class of Context Free languages,
- $\mathcal{L}$ recognizes a bounded number of cross-serial dependencies, i.e. there exists $n \geq 2$ such that $\{w^k | w \in \Sigma^*\} \in \mathcal{L}$ for all $k \leq n$,
- All languages in $\mathcal{L}$ are polynomially parsable,
- All languages in $\mathcal{L}$ have the constant growth property.

Semilinear languages have the constant growth property.

Definition

Let $\Sigma = \{a_1, ..., a_n\}$ be an alphabet with some fixed order. The Parikh image of a word $w \in \Sigma^*$ and a language $L \subseteq \Sigma^*$ are as follows:
$p(w) = \langle |w|_{a_1}, ..., |w|_{a_n} \rangle$,
$p(L) = \{p(w) \mid w \in L\}$.

Definition

Two words $w, w' \in \Sigma^*$ are letter equivalent if $p(w) = p(w')$.
Two languages $L, L' \subseteq \Sigma^*$ are letter equivalent if for every $w \in L$ there is a $w' \in L'$ such that w and w' are letter equivalent and vice versa.

A language is semilinear iff it is letter equivalent to a regular language. Parikh's theorem says that all Context Free languages are semilinear.

The extended Chomsky Hierarchy

We can place the Mildly Context-Sensitive Languages in the Chomsky Hierarchy:

The extended Chomsky Hierarchy

We can place the Mildly Context-Sensitive Languages in the Chomsky Hierarchy:

$$RL \subset CFL \subset MCSL \subset CSL \subset REL$$

The extended Chomsky Hierarchy

We can place the Mildly Context-Sensitive Languages in the Chomsky Hierarchy:

$$RL \subset CFL \subset MCSL \subset CSL \subset REL$$

However, there is (to my knowledge) no grammar formalism that characterizes precisely the class $MCSL$. Also, there is no automaton known to do this.

Some extensions of Context Free Formalisms:

- Tree Adjoining Grammar, Head Grammar, well-nested 2-Multiple Context Free Grammar (all equivalent)
- Linear Context Free Rewriting Systems, Multiple Context Free Grammar, Minimalist Grammar, simple Range Concatenation Grammar (all equivalent)
- These formalisms all describe Mildly Context Sensitive Languages, however the two groups are distinguished.

Some extensions of the Lambek Calculus:

- Combinatory Categorial Grammar (equivalent to TAG)
- Multimodal Categorial Grammar
- Displacement Calculus
- Lambek-Grishin Calculus (exceeds TAG)
- As we will show, restrictions of the Displacement Calculus generate Mildly Context Sensitive Languages.

Introduction

- Multiple Context Free Grammars are like Context Free Grammars, but they act on *tuples* of strings.
- The max. arity of tuples acted upon in such a grammar provides a measure that invokes an infinite hierarchy in the sense of generative capacity and computational complexity.

Grammar

Definition

A Multiple Context Free Grammar is a 6-tuple (N, T, F, P, S, dim) such that:

- N is a finite set of non-terminal symbols, and *dim* assigns a dimension to every non-terminal,
- T is a finite set of terminal symbols,
- F is a finite set of mcf-functions,
- P is a finite set of production rules of the form
  $A_0 \to f[A_1, \ldots, A_k]$ with $k \geq 0$
  $f : (T^*)^{dim(A_1)} \times \ldots \times (T^*)^{dim(A_k)} \to (T^*)^{dim(A_0)}$ and $f \in F$.
- $S \in N$ is a distinguished start symbol such that $dim(S) = 1$.

mcf-function

Definition

f is a *mcf*-function if:

- $f(\overrightarrow{x_1}, ..., \overrightarrow{x_k}) = \alpha_1\beta_1...\alpha_n\beta_n$ where $\alpha_i \in T^*$ and β_j a variable from some x_m.
- Each variable x_{ij} from some vector x_m occurs at most (or exactly) once in the right hand side (linearity)

Definition

The dimension of a *MCFG* G is given by the maximal dimension of the non-terminals, i.e. $max(dim(N))$. We call a *MCFG* of dimension k a k-MCFG.

Example & Notation: $\{a^n b^n c^n d^n | n \geq 1\}$

$$S \to f_1[A] \qquad A \to f_2[A] \qquad A \to f_3[]$$

$$f_1[\langle X, Y \rangle] = \langle XY \rangle \qquad f_2[\langle X, Y \rangle] = \langle aXb, cYd \rangle \qquad f_3[] = \langle ab, cd \rangle$$

Example & Notation: $\{a^n b^n c^n d^n | n \geq 1\}$

$$S \to f_1[A] \qquad A \to f_2[A] \qquad A \to f_3[]$$

$$f_1[\langle X, Y\rangle] = \langle XY\rangle \quad f_2[\langle X, Y\rangle] = \langle aXb, cYd\rangle \quad f_3[] = \langle ab, cd\rangle$$

Example run:

$$S \to f_1[A] \to f_1[f_2[A]] \to f_1[f_2[f_3[]]]$$

$$= f_1[f_2[\langle ab, cd\rangle]] = f_1[\langle aabb, ccdd\rangle] = \langle aabbccdd\rangle.$$

sRCG notation

In equivalent notation:

$$S(XY) \to A(X, Y)$$
$$A(aXb, cYd) \to A(X, Y)$$
$$A(ab, cd) \to \epsilon$$

sRCG notation

In equivalent notation:

$$S(XY) \to A(X, Y)$$
$$A(aXb, cYd) \to A(X, Y)$$
$$A(ab, cd) \to \epsilon$$

Example run:

$$S(aabbccdd) \to A(aabb, ccdd) \to A(ab, cd) \to \epsilon.$$

Well-nestedness

- Well-nested : $A(XY, ZW) \to B(X, W)C(Y, Z)$
- NOT well-nested : $A(XY, ZW) \to B(X, Z)C(Y, W)$

We denote well-nested MCFG by $MCFG_{wn}$.

String language

Definition

Let $G = (N, T, F, P, S)$ be a $MCFG_{(wn)}$.

- For every $A \in N$:
  1. For every $(A \to f[]) \in P : f[] \in yield(A)$,
  2. For every $(A \to f[A_1, ..., A_k]) \in P(k \geq 1)$ and all tuples $\tau_1 \in yield(A_1)...\tau_k \in yield(A_k) : f[\tau_1, ..., \tau_k] \in yield(A)$.
  3. Nothing else is in $yield(A)$.
- The string language of G is $L(G) = \{w | \langle w \rangle \in yield(S)\}$.

Closure Properties

Theorem

For every k, the class of k-$MCFL_{(wn)}$s is closed under:

- *substitution*
- *homomorphism and inverse homomorphism*
- *union,concatenation and Kleene closure*
- *intersection with a regular language*

So the class of k-$MCFL_{(wn)}$s forms a substitution closed full Abstract Family of Languages.

Mild Context Sensitivity

- Every $MCFL_{(wn)}$ is semilinear,
- The (fixed) recognition problem for k-$MCFG_{(wn)}$s is polynomial,
- $count_k = \{a_1^n ... a_k^n | n \geq 0\} \in (k-1)$-$MCFL$ for k odd, $(k-2)$-$MCFL$ o.w.
- $cross_k = \{a_1^n b_1^m ..., a_k^n b_k^m | l, k \geq 0\} \in k$-$MCFL$,
- $copy_k = \{w^k | w \in \Sigma^*\} \in k$-$MCFL$.

Mild Context Sensitivity

- Every $MCFL_{(wn)}$ is semilinear,
- The (fixed) recognition problem for k-$MCFG_{(wn)}$s is polynomial,
- $count_k = \{a_1^n ... a_k^n | n \geq 0\} \in (k-1)$-$MCFL$ for k odd, $(k-2)$-$MCFL$ o.w.
- $cross_k = \{a_1^n b_1^m ..., a_k^n b_k^m | l, k \geq 0\} \in k$-$MCFL$,
- $copy_k = \{w^k | w \in \Sigma^*\} \in k$-$MCFL$.

So, mild context-sensitivity?

MIX is a *MCFL*

- $MIX_k = \{w \in \{a_1, ..., a_k\} | |a_1|_w = ... = |a_k|_w\}$.
  $MIX_3 \in$ 2-*MCFL* (Salvati 2011).

MIX is a *MCFL*

- $MIX_k = \{w \in \{a_1, ..., a_k\} || a_1|_w = ... = |a_k|_w\}$.
  $MIX_3 \in 2\text{-}MCFL$ (Salvati 2011).
- It is shown in (Kanazawa,Salvati 2012) that MIX_3 is not a well-nested 2-*MCFL*.
- So, is $MCFG_{wn}$ a *better* candidate for Mild Context-Sensitivity?

Introduction

Lexicalization is important for our purposes because categorial grammar is by definition lexicalized.

- Displacement grammars are an extension of Lambek grammars
- Displacement grammars extend Lambek grammars by allowing wrapping.
- For concatenation, we have 0 as the unit, for wrapping we have 1 (separator) as unit.
- Let $|_k$ denote insertion at the k-th separator, e.g. $a1bc1d \;|_2\; ef = a1bcefd$.

Definition

Let T be a set of atomic types. Then the set T^* of *general displacement types* is defined as follows:

- If $A \in T$, then $A \in T^*$,
- If $A, B \in T^*$, then
  $A \bullet B, B/A, A\backslash B, \qquad A \odot_k B, A \uparrow_k B, B \downarrow_k A \in T^*$.

Definition

A Displacement grammar is a triple (W, δ, S) such that:

- W is a set of words,
- $\delta \subseteq W \times T^*$ is a type assignment relation,
- $S \in T^*$ is a distinguished start symbol.

Proof Theory of $D_{I,J}$

$$\frac{\delta(\alpha) = A}{\alpha : A}\ Lex. \qquad \frac{}{0 : I}\ Ax.I \qquad \frac{}{1 : J}\ Ax.J$$

$$\frac{\alpha : A \quad \beta : B}{\alpha + \beta : A \bullet B}\ I\bullet \qquad \frac{\gamma : A \bullet B \quad \begin{array}{c}\alpha : A \quad \beta : B \\ \vdots \\ \Delta\langle \alpha + \beta \rangle : C\end{array}}{\Delta\langle \gamma \rangle : C}\ E\bullet$$

$$\frac{\begin{array}{c}\alpha : A \\ \vdots \\ \alpha + \gamma : B\end{array}}{\gamma : A \backslash B}\ I\backslash \qquad \frac{\alpha : A \quad \gamma : A \backslash B}{\alpha + \gamma : B}\ E\backslash \qquad \frac{\begin{array}{c}\alpha : A \\ \vdots \\ \gamma + \alpha : B\end{array}}{\gamma : B / A}\ I/ \qquad \frac{\gamma : B / A \quad \alpha : A}{\gamma + \alpha : B}\ E/$$

$$\frac{\alpha : A \quad \beta : B}{\alpha|_k\beta : A \odot_k B}\ I\odot_k \qquad \frac{\gamma : A \odot_k B \quad \begin{array}{c}\alpha : A \quad \beta : B \\ \vdots \\ \Delta\langle \alpha|_k\beta \rangle : C\end{array}}{\Delta\langle \gamma \rangle : C}\ E\odot_k$$

$$\frac{\begin{array}{c}\alpha : A \\ \vdots \\ \alpha|_k\gamma : B\end{array}}{\gamma : A \downarrow_k B}\ I\downarrow_k \qquad \frac{\alpha : A \quad \gamma : A \downarrow_k B}{\alpha|_k\gamma : B}\ E\downarrow_k \qquad \frac{\begin{array}{c}\alpha : A \\ \vdots \\ \gamma|_k\alpha : B\end{array}}{\gamma : B \uparrow_k A}\ I\uparrow_k \qquad \frac{\gamma : B \uparrow_k A \quad \alpha : A}{\gamma|_k\alpha : B}\ E\uparrow_k$$

Proof Theory of D^1

$$\frac{\delta(\alpha) = A}{\alpha : A}\ Lex. \qquad \frac{}{0 : I}\ Ax.I \qquad \frac{}{1 : J}\ Ax.J$$

$$\frac{\alpha : A \quad \beta : B}{\alpha + \beta : A \bullet B}\ I\bullet \qquad \frac{\gamma : A \bullet B \quad \begin{matrix}\alpha : A \quad \beta : B \\ \vdots \\ \Delta\langle \alpha + \beta \rangle : C\end{matrix}}{\Delta\langle \gamma \rangle : C}\ E\bullet$$

$$\frac{\begin{matrix}\alpha : A \\ \vdots \\ \alpha + \gamma : B\end{matrix}}{\gamma : A\backslash B}\ I\backslash \qquad \frac{\alpha : A \quad \gamma : A\backslash B}{\alpha + \gamma : B}\ E\backslash \qquad \frac{\begin{matrix}\alpha : A \\ \vdots \\ \gamma + \alpha : B\end{matrix}}{\gamma : B/A}\ I/ \qquad \frac{\gamma : B/A \quad \alpha : A}{\gamma + \alpha : B}\ E/$$

$$\frac{\alpha : A \quad \beta : B}{\alpha|_k\beta : A \odot_k B}\ I\odot_k \qquad \frac{\gamma : A \odot_k B \quad \begin{matrix}\alpha : A \quad \beta : B \\ \vdots \\ \Delta\langle \alpha|_k\beta \rangle : C\end{matrix}}{\Delta\langle \gamma \rangle : C}\ E\odot_k$$

$$\frac{\begin{matrix}\alpha : A \\ \vdots \\ \alpha|_k\gamma : B\end{matrix}}{\gamma : A\downarrow_k B}\ I\downarrow_k \qquad \frac{\alpha : A \quad \gamma : A\downarrow_k B}{\alpha|_k\gamma : B}\ E\downarrow_k \qquad \frac{\begin{matrix}\alpha : A \\ \vdots \\ \gamma|_k\alpha : B\end{matrix}}{\gamma : B\uparrow_k A}\ I\uparrow_k \qquad \frac{\gamma : B\uparrow_k A \quad \alpha : A}{\gamma|_k\alpha : B}\ E\uparrow_k$$

Proof Theory of 1-D_J

$$\frac{\delta(\alpha) = A}{\alpha : A}\ Lex. \qquad \frac{}{0 : I}\ Ax.I \qquad \frac{}{1 : J}\ Ax.J$$

$$\frac{\alpha : A \quad \beta : B}{\alpha + \beta : A \bullet B}\ I\bullet \qquad \frac{\gamma : A \bullet B \quad \begin{matrix}\alpha : A \quad \beta : B \\ \vdots \\ \Delta\langle \alpha + \beta \rangle : C\end{matrix}}{\Delta\langle\gamma\rangle : C}\ E\bullet$$

$$\frac{\begin{matrix}\alpha : A \\ \vdots \\ \alpha + \gamma : B\end{matrix}}{\gamma : A \backslash B}\ I\backslash \qquad \frac{\alpha : A \quad \gamma : A \backslash B}{\alpha + \gamma : B}\ E\backslash \qquad \frac{\begin{matrix}\alpha : A \\ \vdots \\ \gamma + \alpha : B\end{matrix}}{\gamma : B / A}\ I/ \qquad \frac{\gamma : B/A \quad \alpha : A}{\gamma + \alpha : B}\ E/$$

$$\frac{\alpha : A \quad \beta : B}{\alpha|_k \beta : A \odot_k B}\ I\odot_k \qquad \frac{\gamma : A \odot_k B \quad \begin{matrix}\alpha : A \quad \beta : B \\ \vdots \\ \Delta\langle \alpha|_k \beta \rangle : C\end{matrix}}{\Delta\langle\gamma\rangle : C}\ E\odot_k$$

$$\frac{\begin{matrix}\alpha : A \\ \vdots \\ \alpha|\gamma : B\end{matrix}}{\gamma : A \downarrow_k B}\ I\downarrow_k \qquad \frac{\alpha : A \quad \gamma : A \downarrow_k B}{\alpha|_k \gamma : B}\ E\downarrow_k \qquad \frac{\begin{matrix}\alpha \neq 0 : A \\ \vdots \\ \gamma|_k \alpha : B\end{matrix}}{\gamma : B \uparrow_k A}\ I\uparrow_k \qquad \frac{\gamma : B \uparrow_k A \quad \alpha : A}{\gamma|_k \alpha : B}\ E\uparrow$$

Copy Language in D^1

$$S' = S \odot_1 I$$

$$\begin{array}{lll} a : A & a : J\backslash(A\backslash S) & a : J\backslash(S \downarrow_1 (A\backslash S)) \\ b : B & b : J\backslash(B\backslash S) & b : J\backslash(S \downarrow_1 (B\backslash S)) \end{array}$$

Copy Language in D^1

$$S' = S \odot_1 I$$

$$\begin{array}{lll} a : A & a : J\backslash(A\backslash S) & a : J\backslash(S \downarrow_1 (A\backslash S)) \\ b : B & b : J\backslash(B\backslash S) & b : J\backslash(S \downarrow_1 (B\backslash S)) \end{array}$$

Example derivation:

$$\dfrac{\dfrac{b : B \qquad \dfrac{\dfrac{a : A \qquad \dfrac{1 : J \qquad a : J\backslash(A\backslash S)}{1a : A\backslash S}}{a1a : S} \qquad \dfrac{b : J\backslash(S \downarrow_1 (B\backslash S))}{1b : S \downarrow_1 (B\backslash S)}}{a1ba : B\backslash S}}{ba1ba : S} \qquad 0 : I}{baba : S \odot_1 I}$$

Copy Language in 1-D_J

$$S = (P \uparrow X) \odot I \qquad x : X$$

$$a : A \qquad a : X \backslash (A \backslash P) \qquad a : X \backslash ((P \uparrow X) \downarrow (A \backslash P))$$

$$b : B \qquad b : X \backslash (B \backslash P) \qquad b : X \backslash ((P \uparrow X) \downarrow (B \backslash P))$$

Copy Language in 1-D_J

$$
\begin{array}{lll}
S = (P \uparrow X) \odot I & x : X & \\
\quad a : A & a : X\backslash(A\backslash P) & a : X\backslash((P \uparrow X) \downarrow (A\backslash P)) \\
\quad b : B & b : X\backslash(B\backslash P) & b : X\backslash((P \uparrow X) \downarrow (B\backslash P))
\end{array}
$$

Example derivation:

$$
\dfrac{\dfrac{\dfrac{b : B \qquad \dfrac{\dfrac{\dfrac{a : A \qquad \dfrac{x : X \quad a : X\backslash(A\backslash P)}{xa : A\backslash P}}{axa : P}}{a1a : P \uparrow X} \qquad \dfrac{x : X \quad b : X\backslash((P \uparrow X) \downarrow (B\backslash P))}{xb : (P \uparrow X) \downarrow (B\backslash P)}}{axba : B\backslash P}}{baxba : P}}{ba1ba : P \uparrow X} \qquad 0 : I}{baba : (P \uparrow X) \odot I}
$$

$L(MCFG_{wn}) \subseteq L(D^1)$ (Wijnholds, 2011)

- From left to right: Given a lexicalized rule $A(\alpha_1 a \alpha_2) \to B_1(\beta_1)...B_n(\beta_n)$, we can always (nondeterministically) find a type assignment $a : T$ such that precisely the following derivation is allowed:

$$\dfrac{\dfrac{}{\alpha : T}\,Lex. \quad \begin{array}{c}\vdots\\ \beta_1 : B\end{array} \quad \ldots \quad \begin{array}{c}\vdots\\ \beta_n : B\end{array}}{\alpha_1 a \alpha_2 : A}$$

$L(MCFG_{wn}) \subseteq L(D^1)$ (Wijnholds, 2011)

Examples:

- $A(aXY, Z) \rightarrow B(X, Z)\ C(Y)$
  $\rightsquigarrow a : A/(B \odot_1 (C \bullet J))$

- $A(Xa, YZ) \rightarrow B(X, Z)\ C(Y)$
  $\rightsquigarrow a : ((B \odot_1 (J \bullet C)) \downarrow_1 A)/J$

$L(MCFG_{wn}) \supseteq L(D^1)$ (Wijnholds, 2011)

- From right to left: a construction in stages. In the first stage, we construct the set $P_0 = \{R^A(w) \to \epsilon \mid \delta(w) = A\}$.
- In each following stage, we *decompose* the types, e.g. for any $R^{A\backslash B}(\alpha_1, ..., \alpha_n) \to \gamma$, we add a rule $R^B(Y_1, ..., Y_m X_1, ..., X_n) \to R^A(Y_1, ..., Y_k)\ R^{A\backslash B}(X_1, ..., X_n)$,

  and for any $\gamma_0 \to \gamma_1 R^{A\bullet B}(Z_1, ..., Z_k)\gamma_2$ we add a rule $R^{A\bullet B}(X_1, ..., X_n Y_1, ..., Y_m) \to R^A(X_1, ..., X_n) R^B(Y_1, ..., Y_m)$

  (respecting sorts)
- The fixed point of the staged construction plus a rule for the start symbol gives us the wanted grammar.

Example: Copy language

$S' = S \odot_1 I$

$a : A$	$a : J\backslash(A\backslash S)$	$a : J\backslash(S \downarrow_1 (A\backslash S))$
$b : B$	$b : J\backslash(B\backslash S)$	$b : J\backslash(S \downarrow_1 (B\backslash S))$

Example: Copy language

$S' = S \odot_1 I$

$a : A$	$a : J\backslash(A\backslash S)$	$a : J\backslash(S \downarrow_1 (A\backslash S))$
$b : B$	$b : J\backslash(B\backslash S)$	$b : J\backslash(S \downarrow_1 (B\backslash S))$

$S'(XY) \rightarrow S(X, Y)$

$R^A(a).$	$R^{J\backslash(A\backslash S)}(a).$	$R^{J\backslash(S\downarrow_1(A\backslash S))}(a).$
$R^B(b).$	$R^{J\backslash(B\backslash S)}(b).$	$R^{J\backslash(S\downarrow_1(B\backslash S))}(b).$

Example: Copy language

$$S' = S \odot_1 I$$

$$\begin{array}{lll} a : A & a : J\backslash(A\backslash S) & a : J\backslash(S \downarrow_1 (A\backslash S)) \\ b : B & b : J\backslash(B\backslash S) & b : J\backslash(S \downarrow_1 (B\backslash S)) \end{array}$$

$$S'(XY) \to S(X, Y)$$

$$\begin{array}{lll} R^{A}(a). & R^{J\backslash(A\backslash S)}(a). & R^{J\backslash(S\downarrow_1(A\backslash S))}(a). \\ R^{B}(b). & R^{J\backslash(B\backslash S)}(b). & R^{J\backslash(S\downarrow_1(B\backslash S))}(b). \end{array}$$

$$\begin{array}{ll} R^{A\backslash S}(\epsilon, X) \to R^{J\backslash(A\backslash S)}(X) & R^{S\downarrow_1(A\backslash S)}(\epsilon, X) \to R^{J\backslash(S\downarrow_1(A\backslash S))}(X) \\ R^{B\backslash S}(\epsilon, X) \to R^{J\backslash(B\backslash S)}(X) & R^{S\downarrow_1(B\backslash S)}(\epsilon, X) \to R^{J\backslash(S\downarrow_1(B\backslash S))}(X) \end{array}$$

Example: Copy language

$$S' = S \odot_1 I$$

$$\begin{array}{lll} a : A & a : J\backslash(A\backslash S) & a : J\backslash(S\downarrow_1 (A\backslash S)) \\ b : B & b : J\backslash(B\backslash S) & b : J\backslash(S\downarrow_1 (B\backslash S)) \end{array}$$

$$S'(XY) \to S(X, Y)$$

$$\begin{array}{lll} R^{A}(a). & R^{J\backslash(A\backslash S)}(a). & R^{J\backslash(S\downarrow_1(A\backslash S))}(a). \\ R^{B}(b). & R^{J\backslash(B\backslash S)}(b). & R^{J\backslash(S\downarrow_1(B\backslash S))}(b). \end{array}$$

$$\begin{array}{ll} R^{A\backslash S}(\epsilon, X) \to R^{J\backslash(A\backslash S)}(X) & R^{S\downarrow_1(A\backslash S)}(\epsilon, X) \to R^{J\backslash(S\downarrow_1(A\backslash S))}(X) \\ R^{B\backslash S}(\epsilon, X) \to R^{J\backslash(B\backslash S)}(X) & R^{S\downarrow_1(B\backslash S)}(\epsilon, X) \to R^{J\backslash(S\downarrow_1(B\backslash S))}(X) \end{array}$$

$$\begin{array}{ll} R^{S}(ZY, X) \to R^{A}(Z)R^{A\backslash S}(Y, X) & R^{A\backslash S}(XZ, WY) \to R^{S}(X, Y)R^{S\downarrow_1(A\backslash S)}(Z, W) \\ R^{S}(ZY, X) \to R^{B}(Z)R^{B\backslash S}(Y, X) & R^{B\backslash S}(XZ, WY) \to R^{S}(X, Y)R^{S\downarrow_1(B\backslash S)}(Z, W) \end{array}$$

Plan

- We show $L(MCFG_{wn}) \subseteq L(1\text{-}D_J) \subseteq L(D^1)$.
- By the first characterization, then, we have the second one: $L(MCFG_{wn}) = L(1\text{-}D_J)$.

$L(MCFG_{wn}) \subseteq L(1\text{-}D_J)$

- Basically the same construction as for $L(MCFG_{wn}) \subseteq L(D^1)$, but:
- For each rule labeled with A of dimension n, we add $x_i^A : X_i^A$ for $1 \leq i \leq n-1$.
- Whenever we introduce the kth separator J^k for an A tuple, we instead introduce x_k^A.
- Whenever we introduce a $A \odot_k B$ construction, we instead use $(A \uparrow X_k^A) \odot B$. Similarly for $A \downarrow_k B$.
- We have 'flattened' types such that we only have two-dimensional strings,
- We use higher-order constructions to do intercalation.

$L(MCFG_{wn}) \subseteq L(1\text{-}D_J)$

Examples:

- $A(aXY, Z) \to B(X, Z)\ C(Y)$
  $\rightsquigarrow a : A/(B \odot_1 (C \bullet J))$
  $\rightsquigarrow a : A/((B \uparrow X_1^B) \odot (C \bullet X_1^A))$

- $A(Xa, YZ) \to B(X, Z)\ C(Y)$
  $\rightsquigarrow a : ((B \odot_1 (J \bullet C)) \downarrow_1 A)/J$
  $\rightsquigarrow a : ((((B \uparrow X_1^B) \odot (X_1^B \bullet C)) \uparrow X_1^B) \downarrow A)/X_1^A$

$L(1\text{-}D_J) \subseteq L(D^1)$

- An expression of type $A \uparrow B$ is an expression of type A with an expression of type B extracted out.
- We say that a type $A \uparrow B$ is in *input position* iff it occurs as one of the following types: $(A \uparrow B)\backslash C, C/(A \uparrow B), (A \uparrow B) \bullet C, C \bullet (A \uparrow B), (A \uparrow B) \downarrow C$.
- Why? Because in these cases we need to use the $I \uparrow$ rule to get an expression of type $A \uparrow B$ and we want to eliminate exactly these derivations.
- Idea: We can replace $A \uparrow B$ in input position by A' and add type assignments such that all derivable expressions of type A' mimick the behaviour of $A \uparrow B$.

Example: Copy language (again)

$$
\begin{array}{lll}
S = (P \uparrow X) \odot I & x : X & \\
a : A & a : X \backslash (A \backslash P) & a : X \backslash ((P \uparrow X) \downarrow (A \backslash P)) \\
b : B & b : X \backslash (B \backslash P) & b : X \backslash ((P \uparrow X) \downarrow (B \backslash P))
\end{array}
$$

Example: Copy language (again)

$$S = (P \uparrow X) \odot I \qquad x : X$$
$$a : A \qquad a : X\backslash(A\backslash P) \qquad a : X\backslash((P \uparrow X) \downarrow (A\backslash P))$$
$$b : B \qquad b : X\backslash(B\backslash P) \qquad b : X\backslash((P \uparrow X) \downarrow (B\backslash P))$$

$$S = P' \odot I \qquad x : X$$
$$a : A \qquad a : X\backslash(A\backslash P) \qquad a : X\backslash(P' \downarrow_1 (A\backslash P))$$
$$b : B \qquad b : X\backslash(B\backslash P) \qquad b : X\backslash(P' \downarrow_1 (B\backslash P))$$

Example: Copy language (again)

$$S = (P \uparrow X) \odot I \qquad x : X$$
$$a : A \qquad a : X\backslash(A\backslash P) \qquad a : X\backslash((P \uparrow X) \downarrow (A\backslash P))$$
$$b : B \qquad b : X\backslash(B\backslash P) \qquad b : X\backslash((P \uparrow X) \downarrow (B\backslash P))$$

$$S = P' \odot I \qquad x : X$$
$$a : A \qquad a : X\backslash(A\backslash P) \qquad a : X\backslash(P' \downarrow_1 (A\backslash P))$$
$$b : B \qquad b : X\backslash(B\backslash P) \qquad b : X\backslash(P' \downarrow_1 (B\backslash P))$$

$$a : J\backslash(A\backslash P') \qquad a : J\backslash(P' \downarrow_1 (A\backslash P'))$$
$$b : J\backslash(B\backslash P') \qquad b : J\backslash(P' \downarrow_1 (B\backslash P'))$$

Conclusion

- We have shown two logical characterizations of the Mildly Context-Sensitive Languages
- We have a choice between a (bounded) high number of connectives but only first-order constructions or a fixed number of connectives but allowing higher-order constructions.
- Which system is favorable?
- Open problem: is there a variant of *D* that relates to *MCFG*? If so, how exactly?