MCFGs and D_k

Two infinite hierarchies

Gijs Wijnholds & Michiel de Winter

LMNLP, 2012

Outline

1. MCFGs
   - Grammar
   - Generative Capacity
   - Automaton
   - Lexicalization of *MCFG*
2. k-MCFG induces D_k
   - Derivation and String Algebras
   - Linearization and Dependency Semantics
   - Block-ordered trees

Introduction

- Multiple Context Free Grammars are like Context Free Grammars, but they act on *tuples* of strings.
- The max. number 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
  $A_0 \to f[A_1, \dots, A_k]$ with $k \geq 0$
  $f : (T^*)^{dim(A_1)} \times \dots \times (T^*)^{dim(A_k)} \to (T^*)^{dim(A_0)}$ 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:

- $f(\overrightarrow{x_1}, \ldots, \overrightarrow{x_k}) = \alpha_1\beta_1 \ldots \alpha_n\beta_n$ 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:

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

Example run:

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

String language

Definition

Let $G = (N, T, F, P, S)$ be a *MCFG*.

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

Closure Properties

Theorem

For every k, the class of k-MCFLs is closed under:

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

So the class of k-MCFLs forms a substitution closed full Abstract Family of Languages.

Mild Context Sensitivity

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

So, mild context-sensitivity?

MIX is a *MCFL*

- $MIX_k = \{w \in \{a_1, \ldots, a_k\} \| |a_1|_w = \ldots = |a_k|_w\}$. $MIX_3 \in 2\text{-}MCFL$ (Salvati 2011). General case: can show with shuffle closure that $MIX_k \in k\text{-}MCFL$.
- This is bad, we do not want completely free word order.

MIX is a *MCFL*

- $MIX_k = \{w \in \{a_1, \dots, a_k\} \mid |a_1|_w = \dots = |a_k|_w\}$. $MIX_3 \in 2\text{-}MCFL$ (Salvati 2011). General case: can show with shuffle closure that $MIX_k \in k\text{-}MCFL$.
- This is bad, we do not want completely free word order.
- (Kanazawa, 2009,2010) discusses well-nested *MCFG*, which also is capable of describing $count_k, cross_k, copy_k$. It is not known (but suspected) that *MIX* is not a well-nested *MCFL*.

Beyond *MCFL*

We saw the definition of k-pumpability for a string, but:

Beyond *MCFL*

We saw the definition of k-pumpability for a string, but:

- The pumping lemma for k-*MCFL*s is weak in the sense that it is existential:

Theorem

(Seki et al. 1991) For any infinite MCFL L, there exists a 2k-pumpable string $w \in L$.

Beyond *MCFL*

We saw the definition of k-pumpability for a string, but:

- The pumping lemma for k-*MCFL*s is weak in the sense that it is existential:

Theorem

(Seki et al. 1991) For any infinite MCFL L, there exists a 2k-pumpable string $w \in L$.

- In contrast, the pumping lemma for well-nested *MCFL* is universal:

Theorem

(Kanazawa,2010) For any $MCFL_{wn}$ L, all but finitely many strings $w \in L$ are 2k-pumpable.

MCFL=OUT(DTWT) (D.J.Weir)

The class of string languages that can be described by *MCFGs* are also characterized by Deterministic Tree Walking Transducers:

Definition

A DTWT is a 6-tuple $(Q, G, \Delta, \delta, q_0, F)$ where:

- Q is a finite set of states,
- $G = (N, T, S, R)$ is a *CFG* without ϵ-rules,
- Δ is a finite set of output symbols,
- $\delta : Q \times (N \cup T) \to Q \times \{stay, up\} \cup \{d(k)|k \geq 1\} \times \Delta^*$ is the transition function,
- q_0 is the initial state,
- $F \subseteq Q$ is the set of final states.

A DTWT for $\{a^n b^n c^n d^n | n \geq 1\}$

Consider $M = (\{q_0, q_1, q_2, q_3\}, G, \{a, b, c, d\}, \delta, q_0, \{q_3\})$ where $G = (\{S, A\}, \{e\}, S, \{S \rightarrow A, A \rightarrow A, A \rightarrow e\})$ and:

$$
\begin{array}{ll}
\delta(q_0, S) = (q_0, d(1), \epsilon) & \delta(q_2, A) = (q_2, d(1), c) \\
\delta(q_0, A) = (q_0, d(1), a) & \delta(q_2, e) = (q_3, up, \epsilon) \\
\delta(q_0, e) = (q_1, up, \epsilon) & \delta(q_3, S) = (q_3, up, \epsilon) \\
\delta(q_1, S) = (q_2, d(1), \epsilon) & \delta(q_3, A) = (q_3, up, d) \\
\delta(q_1, A) = (q_1, up, b) &
\end{array}
$$

A DTWT for $\{a^n b^n c^n d^n | n \geq 1\}$

Consider $M = (\{q_0, q_1, q_2, q_3\}, G, \{a, b, c, d\}, \delta, q_0, \{q_3\})$ where $G = (\{S, A\}, \{e\}, S, \{S \to A, A \to A, A \to e\})$ and:

$$
\begin{array}{ll}
\delta(q_0, S) = (q_0, d(1), \epsilon) & \delta(q_2, A) = (q_2, d(1), c) \\
\delta(q_0, A) = (q_0, d(1), a) & \delta(q_2, e) = (q_3, up, \epsilon) \\
\delta(q_0, e) = (q_1, up, \epsilon) & \delta(q_3, S) = (q_3, up, \epsilon) \\
\delta(q_1, S) = (q_2, d(1), \epsilon) & \delta(q_3, A) = (q_3, up, d) \\
\delta(q_1, A) = (q_1, up, b) &
\end{array}
$$

Exercise: Draw the derivation tree + traversal for *aabbccdd*.

Introduction

Lexicalization is important for our purposes because dependency structures correspond precisely to lexicalised grammars.

Substitution

Given some rule $A_0(\overrightarrow{\alpha_0}) \to A_1(\overrightarrow{\alpha_1})...A_n(\overrightarrow{\alpha_n})$, we substitute A_k by considering all rules $A_k(\overrightarrow{\beta_0}) \to \gamma$ and replacing the variables of $\overrightarrow{\alpha_k}$ in α_0 by their corresponding chunks in β_0, and replacing $A_k(\overrightarrow{\alpha_k})$ by γ:

$$\begin{gathered} A(X, YZ) \to B(X, Y)D(Z) \\ B(aX, bY) \to C(X, Y) \\ \Downarrow \\ A(aX, bYZ) \to C(X, Y)D(Z) \end{gathered}$$

This preserves string language and does not affect dimension.

Elimination of left-recursion

Given left-recursive rules $A_0(\overrightarrow{\alpha_0}) \to A_0(\overrightarrow{\delta_1})\ldots A_n(\overrightarrow{\alpha_n})$ and other rules $A_0(\overrightarrow{\beta_0}) \to \gamma$, we eliminate left-recursion by choosing a fresh non-terminal B with $dim(B) = dim(A_0)$ and for each of the left-recursive rules:

1. Add the rules $B(\overrightarrow{\alpha_0}) \to A_1(\overrightarrow{\alpha_1})\ldots A_n(\overrightarrow{\alpha_n})B(\overrightarrow{\delta_1})$ and $B_0(\overrightarrow{\alpha'_0}) \to A_1(\overrightarrow{\alpha_1})\ldots A_n(\overrightarrow{\alpha_n})$
   where $\alpha'_0 = \alpha_0/\alpha_1$.
2. Add the rule $A_0(\beta'_0) \to \gamma B(\beta_1)$ where $\beta'_0 = \beta_0 + +\beta_1$ (variables inserted).
3. Remove the left-recursive rule.

Example

$$A(XY,Z) \rightarrow A(X,Z)C(Y)$$
$$A(X,Y) \rightarrow D(X,Y)$$
$$A(X,YZ) \rightarrow E(X,Y,Z)$$
$$\Downarrow$$
$$B'(X,\epsilon) \rightarrow C(X)$$
$$B'(XY,Z) \rightarrow C(Y)B'(X,Z)$$
$$A(XT_1,T_2Y) \rightarrow D(X,Y)B'(T_1,T_2)$$
$$A(XT_1,T_2YZ) \rightarrow E(X,Y,Z)B'(T_1,T_2)$$
$$A(X,Y) \rightarrow D(X,Y)$$
$$A(X,YZ) \rightarrow E(X,Y,Z)$$

Preserves string language and dimension.

Algorithm

On a *MCFG* G with $\epsilon \notin L(G)$, we lexicalize it by the following algorithm:

1. Order the clauses, say $\{A_1, ..., A_n\}$,
2. Ensure (with substitution) that if $A_j(\alpha_1, ..., \alpha_m) \to A_k(X_1, ..., X_n)\gamma$, $j \leq k$,
3. Eliminate left-recursive clauses $A_k \to A_k\gamma$, thereby introducing new clauses B_k,
4. Lexicalize the clauses, starting with A_{n-1} and ending with A_1,
5. Lexicalize the B_k clauses,
6. Add a new start clause $S'(X) \to S(X)$.

The construction only uses substitution and elimination of left-recursive rules, and hence preserves both string language and dimension.

Example

$$
\begin{aligned}
&S(XYZ) \rightarrow A(X, Z)B(Y)\\
&A(X, YZ) \rightarrow A(X, Y)C(Z)\\
&A(X, Y) \rightarrow D(X)E(Y)\\
&B(b) \rightarrow \epsilon\\
&C(c) \rightarrow \epsilon\\
&D(d) \rightarrow \epsilon\\
&E(e) \rightarrow \epsilon
\end{aligned}
$$

Result

$$
\begin{aligned}
&S'(X) \to S(X)\\
&S(dYeU) \to C(U)B(Y)\\
&S(dXYeZ) \to B'(X,Z)B(Y)\\
&A(dX,eY) \to B'(X,Y)\\
&A(d,e) \to \epsilon\\
&B'(X,Yc) \to B'(X,Y)\\
&B'(\epsilon,c) \to \epsilon\\
&B(b) \to \epsilon\\
&C(c) \to \epsilon\\
&D(d) \to \epsilon\\
&E(e) \to \epsilon
\end{aligned}
$$

Some questions

1. A *DTWT* takes a *CFG* G and produces a *MCFL* L using the derivation trees of G. Is there a relation with dependency structures? Not trivial, because *DTWT* just produce string languages, unknown whether the derivation trees of *MCFG*s have a connection with configurations of a *DTWT*.
2. MIX_k is a shuffle language (see Bergland et al. 2011,Salvati 2011). Suspicion: $SL \subset MCFL$, $SL \nsubseteq MCFL$. $SL \wedge RL$ (shuffle languages intersected with regular languages) contain $count_k, cross_k, MIX_k$ but not $copy_k$. Is there a relation with $MCFL - MCFL_{wn}$ and **LG**?

Introduction

- *MCFG*s induce exactly the dependency structures of bounded degree. More specifically, the dimension k of some *MCFG* G corresponds to the maximal block degree of the induced dependency structure.
- In the case of *CFG*, the induction was quite simple: given a derivation tree t, the induced dependency structure is simply an ordering of the nodes v in t with respect to the string position of the anchor produced by v.
- Here, we will show how to construct a derivation algebra $T_{\Sigma(G)}$ for a *MCFG* G, and define linearization and dependency semantics.

Derivation Algebra

Definition

Let $G = (N, T, F, P, S, dim)$ be an $MCFG$. Define $\Sigma(G)$ to be the N-sorted set given by P, where

$$Type_{\Sigma(G)}(A \to f[A_1, \dots, A_m]) = A_1 \times \dots \times A_m \to A.$$

The derivation algebra of G is defined as the term algebra $T_{\Sigma(G)}$ over $\Sigma(G)$.

Example

Consider the grammar

$$
\begin{array}{ll}
S \to f_1[A] & f_1[\langle X\rangle] = \langle Xb\rangle \\
S \to f_2[A, B] & f_2[\langle X\rangle, \langle YZ\rangle] = \langle XYbZ\rangle \\
A \to f_3[] & f_3[] = \langle a\rangle \\
B \to f_4[A, B] & f_4[\langle X\rangle, \langle YZ\rangle] = \langle XY, bZ\rangle \\
B \to f_5[A] & f_5[\langle X\rangle] = \langle X, b\rangle .
\end{array}
$$

Example

Then $\Sigma(G) = \{$

$$(S \to f_1[A]) : A \to S$$
$$(S \to f_2[A, B]) : A \times B \to S$$
$$(A \to f_3[]) : A$$
$$(B \to f_4[A, B]) : A \times B \to B$$
$$(B \to f_5[A]) : A \to B$$

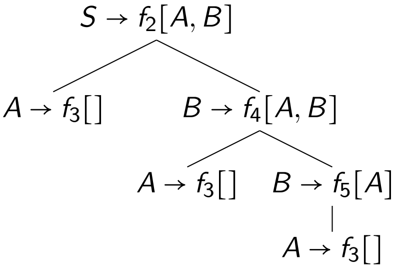
$\}$.

Example

Then $\Sigma(G) = \{$

$(S \to f_1[A]) : A \to S$
$(S \to f_2[A, B]) : A \times B \to S$
$(A \to f_3[]) : A$
$(B \to f_4[A, B]) : A \times B \to B$
$(B \to f_5[A]) : A \to B$
$\}$.

And



is the tree representation of some term in $T_{\Sigma(G)}$.

String Algebra

Kuhlmann's way of defining the string language of an *MCFG*:

Definition

Let $G = (N, T, F, P, S, dim)$ be an *MCFG*. The string algebra for G is the $\Sigma(G)$-algebra Θ with:

- $dom(\Theta)_A = (T^*)^{dim(A)}$ for all $A \in N$.
- For each production rule $p = A \rightarrow f[A_1, \dots, A_m]$ with $f :: k_1 \times \dots \times k_m \rightarrow k$ and body $\overrightarrow{\gamma}$: $f_p(\overrightarrow{\alpha_1}, \dots, \overrightarrow{\alpha_m}) = \overrightarrow{\gamma}[x_{ij}/\alpha_{ij}]$ for x_{ij} the corresponding variable for α_{ij} in $\overrightarrow{\alpha_i}$.
- The string language of G is $L(G) := \{\overrightarrow{a} | \exists t \in T_{\Sigma(G),S} : \langle \overrightarrow{a} \rangle \in [\![t]\!]_\Theta\}$

Reading off a string

Consider the running example:

- $S \to f_2[A, B]$
  - $A \to f_3[]$
  - $B \to f_4[A, B]$
    - $A \to f_3[]$
    - $B \to f_5[A]$
      - $A \to f_3[]$

$$f_2[f_3[], f_4[f_3[], f_5[f_3[]]]]$$

Reading off a string

Consider the running example:

$$S \to f_2[A, B]$$

$$A \to f_3[] \qquad B \to f_4[A, B]$$

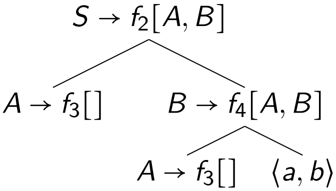
$$A \to f_3[] \qquad B \to f_5[A]$$

$$\langle a \rangle$$

$$f_2[f_3[], f_4[f_3[], f_5[\langle a \rangle]]]$$

Reading off a string

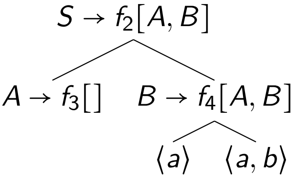
Consider the running example:



$$f_2[f_3[], f_4[f_3[], \langle a, b \rangle]]$$

Reading off a string

Consider the running example:



$$f_2[f_3[\,], f_4[\langle a \rangle, \langle a, b \rangle]]$$

Reading off a string

Consider the running example:

$$S \to f_2[A, B]$$

$$A \to f_3[] \qquad \langle aa, bb \rangle$$

$$f_2[f_3[], \langle aa, bb \rangle]$$

Reading off a string

Consider the running example:

$$S \to f_2[A, B]$$

$$\langle a \rangle \qquad \langle aa, bb \rangle$$

$$f_2[\langle a \rangle, \langle aa, bb \rangle]$$

Reading off a string

Consider the running example:

$$\langle aaabbb \rangle$$

$$\langle aaabbb \rangle$$

Linearization Semantics

- Kuhlmann first gets rid so-called non-essential concatenation functions, i.e.:
  - ϵ-rules. Example: $A(b, \epsilon) \to \epsilon$.
  - Ill-ordered rules. Example: $A(XY) \to B(Y)C(X)$.
- Kuhlmann does this by relabelling $\Rightarrow$ there are also normal form algorithms for this (Kallmeyer, §7.2)

Lemma

(Kuhlmann, lemma 6.2.1) For each lexicalized MCFG G, there is an equivalent lexicalized MCFG G' such that the derivation trees of G and G' are isomorphic modulo relabelling, the string semantics are equal, and G' does not contain useless, ill-ordered, or ϵ-rules.

Definition

Let $G = (N, T, F, P, S, dim)$ be an $MCFG$. The linearization algebra for G is the $\Sigma(G)$-algebra Θ with:

- $dom(\Theta)_A = ((\mathbb{N}^*)^+)^{dim(A)}$ for all $A \in N$.
- For each production rule $p = A \to f[A_1, ..., A_m]$ with $f :: k_1 \times ... \times k_m \to k$,anchor a and body $\overrightarrow{\gamma}$: $f_p(\overrightarrow{\alpha_1}, ..., \overrightarrow{\alpha_m}) = \overrightarrow{\gamma}[a/\epsilon][x_{ij}/pfx_i(\alpha_{ij})]$ for X_{ij} the corresponding variable for α_{ij} in $\overrightarrow{\alpha_i}$ and pfx_i is the string homomorphism defined by $pfx_i(u) = i \circ u$.
- The linearization language of G is $\Lambda(G) := \{\overrightarrow{u} | \exists t \in T_{\Sigma(G),S} : \langle \overrightarrow{u} \rangle \in [\![t]\!]_\Theta\}$.

Linearizing a tree

Consider the running example:

$$S \rightarrow f_2[A, B]$$

$$A \rightarrow f_3[] \qquad B \rightarrow f_4[A, B]$$

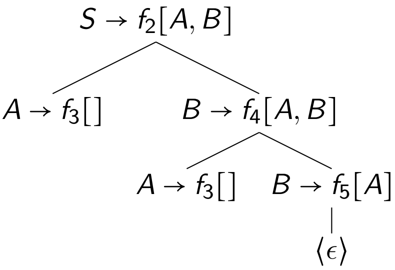
$$A \rightarrow f_3[] \qquad B \rightarrow f_5[A]$$

$$A \rightarrow f_3[]$$

$$f_2[f_3[], f_4[f_3[], f_5[f_3[]]]]$$

Linearizing a tree

Consider the running example:



$$f_2[f_3[], f_4[f_3[], f_5[\langle \epsilon \rangle]]]$$

Linearizing a tree

Consider the running example:

$$S \to f_2[A, B]$$

$$A \to f_3[] \qquad B \to f_4[A, B]$$

$$A \to f_3[] \qquad \langle 1, \epsilon \rangle$$

$$f_2[f_3[], f_4[f_3[], \langle 1, \epsilon \rangle]]$$

Linearizing a tree

Consider the running example:

$$S \to f_2[A, B]$$

$$A \to f_3[\,] \qquad B \to f_4[A, B]$$

$$\langle \epsilon \rangle \qquad \langle 1, \epsilon \rangle$$

$$f_2[f_3[\,], f_4[\langle \epsilon \rangle, \langle 1, \epsilon \rangle]]$$

Linearizing a tree

Consider the running example:

$$
\begin{array}{c}
S \to f_2[A, B] \\
\diagup \qquad \diagdown \\
A \to f_3[] \qquad \langle 1 \cdot 21, \epsilon \cdot 2 \rangle
\end{array}
$$

$$f_2[f_3[], \langle 1 \cdot 21, \epsilon \cdot 2 \rangle]$$

Linearizing a tree

Consider the running example:

$$S \to f_2[A, B]$$

$$\langle \epsilon \rangle \qquad \langle 1 \cdot 21, \epsilon \cdot 2 \rangle$$

$$f_2[\langle \epsilon \rangle, \langle 1 \cdot 21, \epsilon \cdot 2 \rangle]$$

Linearizing a tree

Consider the running example:

$$\langle 1 \cdot 21 \cdot 221 \cdot \epsilon \cdot 2 \cdot 22 \rangle$$

$$\langle 1 \cdot 21 \cdot 221 \cdot \epsilon \cdot 2 \cdot 22 \rangle$$

Dependency Semantics

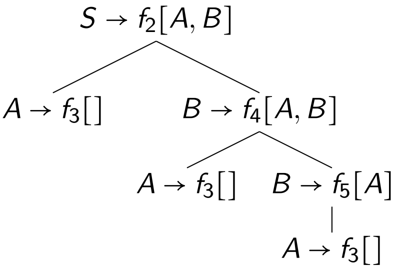
Definition

Induced dependency structures Let G be an *MCFG* and let $t \in T_{\Sigma(G)}$. The dependency structure induced by t is the segmented structure $D := (nod(t), \trianglelefteq, \preceq, \equiv)$ with:

- $u \trianglelefteq v$ iff u dominates v in t,
- $u \preceq v$ iff u precedes v in $[\![t]\!]_{\mathbf{L}}$,
- $u \equiv v$ iff u and v appear in the same component of $[\![t]\!]_{\mathbf{L}}$.

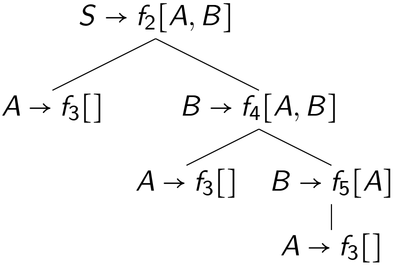
Computing a dependency structure

Running example:

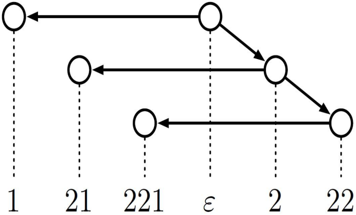


Computing a dependency structure

Running example:



Dependency structure:



Block-order collect (again)

```
BLOCK-ORDER-COLLECT(u)
L ← NIL; calls[u] ← calls[u] + 1
foreach v in order[u][calls[u]]
        do if v = u
                then L ← L · [u]
                else L ← L · BLOCK-ORDER-COLLECT(v)
return L
```

Relation to block-ordered trees

- In the case of block-ordered trees, we go over a node *i* times, considering every *j*-th element of the *i*-th tuple. This is similar to the semantics for the *j*-th variable of the *i*-th component in an *mcf*-function. Hence, we obtain the following result:

Theorem

Kuhlmann, 6.2.1 $\forall k \in \mathbb{N} : D(k\text{-}MCFG) = D_k$.

Exercises

- Give the derivation tree + corresponding dependency structure for *aabbccdd* using the (lexicalized version of the) grammar given for $\{a^n b^n c^n d^n\}$.
- Draw all possible dependency structures in D_3 with 4 nodes.