Tree Adjoining Grammars

Logical Methods in Natural Language Processing
Lecturer: M. Moortgat

Cecilia Chávez Aguilera

May 3, 2012

Joshi’s Program

- Joshi, Aravind K., Leon S. Levy, and Masako Takahashi. 1975. ‘Tree Adjunct Grammars.’*Journal of Computer and System Sciences*, 10(2)

Joshi's Program

- Joshi, Aravind K., Leon S. Levy, and Masako Takahashi. 1975. 'Tree Adjunct Grammars.'*Journal of Computer and System Sciences*, 10(2)
- TAG's have been a cornerstone in the road of Mildy-Context Sensitive Languages

Joshi's Program

- Joshi, Aravind K., Leon S. Levy, and Masako Takahashi. 1975. 'Tree Adjunct Grammars.'*Journal of Computer and System Sciences*, 10(2)
- TAG's have been a cornerstone in the road of Mildy-Context Sensitive Languages
- How much context sensitivity do we need to model natural languages?

Joshi's Program

- Joshi, Aravind K., Leon S. Levy, and Masako Takahashi. 1975. 'Tree Adjunct Grammars.'*Journal of Computer and System Sciences*, 10(2)
- TAG's have been a cornerstone in the road of Mildy-Context Sensitive Languages
- How much context sensitivity do we need to model natural languages?
- Polinomial Parsable

Formalism

- A TAG is a tuple $G = \langle N, T, S, I, A, \rangle$ where

Formalism

- A TAG is a tuple $G = \langle N, T, S, I, A, \rangle$ where
  - N a set of non-terminal symbols.

Formalism

- A TAG is a tuple $G = \langle N, T, S, I, A, \rangle$ where
  - N a set of non-terminal symbols.
  - T an alphabet of terminal symbols $N \cup T = \emptyset$

Formalism

- A TAG is a tuple $G = \langle N, T, S, I, A, \rangle$ where
  - N a set of non-terminal symbols.
  - T an alphabet of terminal symbols $N \cup T = \emptyset$
  - $S \in N$.

Formalism

- A TAG is a tuple $G = \langle N, T, S, I, A, \rangle$ where
  - N a set of non-terminal symbols.
  - T an alphabet of terminal symbols $N \cup T = \emptyset$
  - $S \in N$.
  - I a finite set of trees called initial trees (usually denoted by α) and

Formalism

- A TAG is a tuple $G = \langle N, T, S, I, A, \rangle$ where
  - N a set of non-terminal symbols.
  - T an alphabet of terminal symbols $N \cup T = \emptyset$
  - $S \in N$.
  - I a finite set of trees called initial trees (usually denoted by α) and
  - A a finite set of trees called auxiliary trees (usually denoted by β).

- The set $A \cup I$ is set of *elementary trees* of G from which we will construct our derived trees.

- The set $A \cup I$ is set of *elementary trees* of G from which we will construct our derived trees.
- A tree is *Initial* if its root node is labeled with S. Inner and frontier nodes can be either terminal or non-terminal.

- The set $A \cup I$ is set of *elementary trees* of G from which we will construct our derived trees.
- A tree is *Initial* if its root node is labeled with S. Inner and frontier nodes can be either terminal or non-terminal.
- A tree is *Auxiliar* if it has a special node called its *food node*, marked with * which has the same label than its root node.

- The set $A \cup I$ is set of *elementary trees* of G from which we will construct our derived trees.
- A tree is *Initial* if its root node is labeled with S. Inner and frontier nodes can be either terminal or non-terminal.
- A tree is *Auxiliar* if it has a special node called its *food node*, marked with * which has the same label than its root node.
- The string language generated by G is defined to be the set

$$L(G) = \{y(t)|t \in T(G)\}$$

where $T(G)$ is the set of all derived trees of G, and $y(t)$ is the unique string associated with t, (the yield of t) obtained by concatenating all terminal symbols labeling the frontier of t from left to right.

- The combination operations for our grammar are *Substitution* and *Adjunction*

- The combination operations for our grammar are *Substitution* and *Adjunction*
- Substitution Let $\gamma = \langle V, E, r \rangle$ be a syntactic tree, $\alpha = \langle V', E', r' \rangle$ an initial tree, and $v \in V$. Denote by $\alpha\,[v, \gamma]$, the result of substituting γ into α at node v, defined as follows:

- The combination operations for our grammar are *Substitution* and *Adjunction*
- Substitution Let $\gamma = \langle V, E, r \rangle$ be a syntactic tree, $\alpha = \langle V', E', r' \rangle$ an initial tree, and $v \in V$. Denote by $\alpha\,[v, \gamma]$, the result of substituting γ into α at node v, defined as follows:
  - If v is no leaf or $l(v) \neq l(r)$, then $\alpha\,[v, \gamma]$ is undefined.

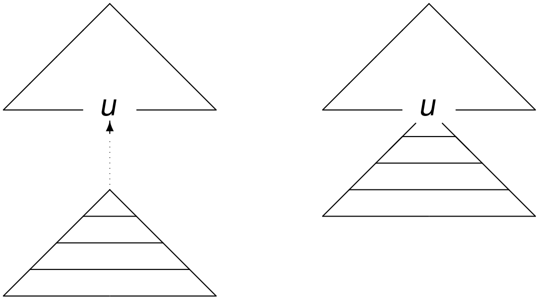
- The combination operations for our grammar are *Substitution* and *Adjunction*
- Substitution Let $\gamma = \langle V, E, r \rangle$ be a syntactic tree, $\alpha = \langle V', E', r' \rangle$ an initial tree, and $v \in V$. Denote by $\alpha\,[v, \gamma]$, the result of substituting γ into α at node v, defined as follows:
  - If v is no leaf or $l(v) \neq l(r)$, then $\alpha\,[v, \gamma]$ is undefined.
  - Otherwise, $\alpha\,[v, \gamma] := \langle V'', E'', r'' \rangle$ with $V'' = V \cup V' \backslash \{v\}$, $E'' = \{E \backslash \{\langle v_1, v_2 \rangle | v_2 = v\}\} \cup E' \cup \{\langle v_1, r \rangle | \langle v_1, v \rangle \in E\}$

- The combination operations for our grammar are *Substitution* and *Adjunction*
- Substitution Let $\gamma = \langle V, E, r \rangle$ be a syntactic tree, $\alpha = \langle V', E', r' \rangle$ an initial tree, and $v \in V$. Denote by $\alpha[v, \gamma]$, the result of substituting γ into α at node v, defined as follows:
  - If v is no leaf or $l(v) \neq l(r)$, then $\alpha[v, \gamma]$ is undefined.
  - Otherwise, $\alpha[v, \gamma] := \langle V'', E'', r'' \rangle$ with $V'' = V \cup V' \backslash \{v\}$, $E'' = \{E \backslash \{\langle v_1, v_2 \rangle | v_2 = v\}\} \cup E' \cup \{\langle v_1, r \rangle | \langle v_1, v \rangle \in E\}$
- A leaf that has a non-terminal label is called a substitution node.

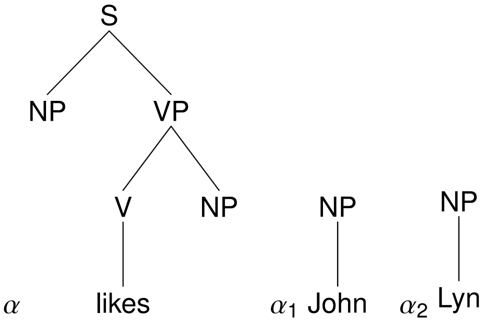
- The combination operations for our grammar are *Substitution* and *Adjunction*
- Substitution Let $\gamma = \langle V, E, r \rangle$ be a syntactic tree, $\alpha = \langle V', E', r' \rangle$ an initial tree, and $v \in V$. Denote by $\alpha[v, \gamma]$, the result of substituting γ into α at node v, defined as follows:
  - If v is no leaf or $l(v) \neq l(r)$, then $\alpha[v, \gamma]$ is undefined.
  - Otherwise, $\alpha[v, \gamma] := \langle V'', E'', r'' \rangle$ with $V'' = V \cup V' \backslash \{v\}$, $E'' = \{E \backslash \{\langle v_1, v_2 \rangle | v_2 = v\}\} \cup E' \cup \{\langle v_1, r \rangle | \langle v_1, v \rangle \in E\}$
- A leaf that has a non-terminal label is called a substitution node.
- This operation can be iterated as long as there is a substition node.

Substitution

- In a picture



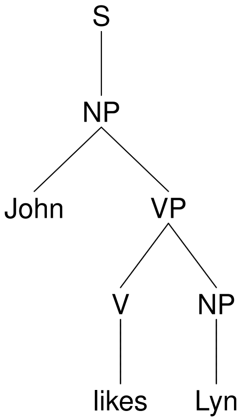
Example



Substituting

Derived tree

$$\alpha\,[NP, \alpha_1];\, \alpha\,[NP, \alpha_2]$$



Adjunction

- Let $\alpha = \langle V, E, r \rangle$ be an initial tree, $\beta = \langle V', E', r', * \rangle$, an auxiliary tree, $v \in V$ a node in α such that $l(v) = N_0$ some non-terminal symbol.

Adjunction

- Let $\alpha = \langle V, E, r \rangle$ be an initial tree, $\beta = \langle V', E', r', * \rangle$, an auxiliary tree, $v \in V$ a node in α such that $l(v) = N_0$ some non-terminal symbol.
  - Denote by α / v the subtree of α rooted at v,

Adjunction

- Let $\alpha = \langle V, E, r \rangle$ be an initial tree, $\beta = \langle V', E', r', * \rangle$, an auxiliary tree, $v \in V$ a node in α such that $l(v) = N_0$ some non-terminal symbol.
  - Denote by α / v the subtree of α rooted at v,
  - Denote by $\alpha \upharpoonright v$ the tree $\langle V^{\upharpoonright}, E^{\upharpoonright}, r^{\upharpoonright} \rangle$ where:

Adjunction

- Let $\alpha = \langle V, E, r\rangle$ be an initial tree, $\beta = \langle V', E', r', *\rangle$, an auxiliary tree, $v \in V$ a node in α such that $l(v) = N_0$ some non-terminal symbol.
  - Denote by α/v the subtree of α rooted at v,
  - Denote by $\alpha \upharpoonright v$ the tree $\langle V^{\upharpoonright}, E^{\upharpoonright}, r^{\upharpoonright}\rangle$ where:
    - ⋆ $V^{\upharpoonright} = V \backslash \{v \in V | v \in \lfloor v \rfloor\}$

Adjunction

- Let $\alpha = \langle V, E, r \rangle$ be an initial tree, $\beta = \langle V', E', r', * \rangle$, an auxiliary tree, $v \in V$ a node in α such that $l(v) = N_0$ some non-terminal symbol.
  - Denote by α / v the subtree of α rooted at v,
  - Denote by $\alpha \upharpoonright v$ the tree $\langle V^{\upharpoonright}, E^{\upharpoonright}, r^{\upharpoonright} \rangle$ where:
    - ★ $V^{\upharpoonright} = V \backslash \{v \in V | v \in \lfloor v \rfloor\}$
    - ★ $E^{\upharpoonright} = E \backslash \{\langle v_1, v_2 \rangle | v_1 = v\}$

Adjunction

- Let $\alpha = \langle V, E, r \rangle$ be an initial tree, $\beta = \langle V', E', r', * \rangle$, an auxiliary tree, $v \in V$ a node in α such that $l(v) = N_0$ some non-terminal symbol.
  - Denote by α / v the subtree of α rooted at v,
  - Denote by $\alpha \upharpoonright v$ the tree $\langle V^{\upharpoonright}, E^{\upharpoonright}, r^{\upharpoonright} \rangle$ where:
    - $V^{\upharpoonright} = V \backslash \{v \in V | v \in \lfloor v \rfloor\}$
    - $E^{\upharpoonright} = E \backslash \{\langle v_1, v_2 \rangle | v_1 = v\}$
    - $r^{\upharpoonright} = r$

Adjunction

- Let $\alpha = \langle V, E, r \rangle$ be an initial tree, $\beta = \langle V', E', r', * \rangle$, an auxiliary tree, $v \in V$ a node in α such that $l(v) = N_0$ some non-terminal symbol.
  - Denote by α / v the subtree of α rooted at v,
  - Denote by $\alpha \upharpoonright v$ the tree $\langle V^{\upharpoonright}, E^{\upharpoonright}, r^{\upharpoonright} \rangle$ where:
    - ⋆ $V^{\upharpoonright} = V \backslash \{v \in V | v \in \lfloor v \rfloor\}$
    - ⋆ $E^{\upharpoonright} = E \backslash \{\langle v_1, v_2 \rangle | v_1 = v\}$
    - ⋆ $r^{\upharpoonright} = r$
- Then, denote by $\beta(\alpha)_v$ the adjunction of β into α at node v defined in the following way:

Adjunction

- Let $\alpha = \langle V, E, r \rangle$ be an initial tree, $\beta = \langle V', E', r', * \rangle$, an auxiliary tree, $v \in V$ a node in α such that $l(v) = N_0$ some non-terminal symbol.
  - Denote by α / v the subtree of α rooted at v,
  - Denote by $\alpha \upharpoonright v$ the tree $\langle V^{\upharpoonright}, E^{\upharpoonright}, r^{\upharpoonright} \rangle$ where:
    - ★ $V^{\upharpoonright} = V \backslash \{v \in V | v \in \lfloor v \rfloor\}$
    - ★ $E^{\upharpoonright} = E \backslash \{\langle v_1, v_2 \rangle | v_1 = v\}$
    - ★ $r^{\upharpoonright} = r$
- Then, denote by $\beta(\alpha)_v$ the adjunction of β into α at node v defined in the following way:
  - If $l(v) \neq l(r')$, then this operation is undefined

Adjunction

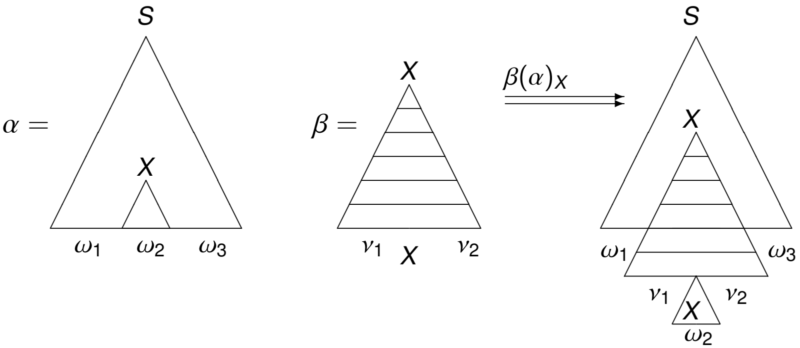
- Let $\alpha = \langle V, E, r \rangle$ be an initial tree, $\beta = \langle V', E', r', * \rangle$, an auxiliary tree, $v \in V$ a node in α such that $l(v) = N_0$ some non-terminal symbol.
  - Denote by α/v the subtree of α rooted at v,
  - Denote by $\alpha \upharpoonright v$ the tree $\langle V^{\upharpoonright}, E^{\upharpoonright}, r^{\upharpoonright} \rangle$ where:
    - ★ $V^{\upharpoonright} = V \backslash \{v \in V | v \in \lfloor v \rfloor\}$
    - ★ $E^{\upharpoonright} = E \backslash \{\langle v_1, v_2 \rangle | v_1 = v\}$
    - ★ $r^{\upharpoonright} = r$
- Then, denote by $\beta(\alpha)_v$ the adjunction of β into α at node v defined in the following way:
  - If $l(v) \neq l(r')$, then this operation is undefined
  - Otherwise, do $\alpha \upharpoonright v\,[v, \beta]$, and call γ the resulting tree, then, do $\gamma\,[*, \alpha/v]$

Adjunction

- Let $\alpha = \langle V, E, r\rangle$ be an initial tree, $\beta = \langle V', E', r', *\rangle$, an auxiliary tree, $v \in V$ a node in α such that $l(v) = N_0$ some non-terminal symbol.
  - Denote by α/v the subtree of α rooted at v,
  - Denote by $\alpha \upharpoonright v$ the tree $\langle V^{\upharpoonright}, E^{\upharpoonright}, r^{\upharpoonright}\rangle$ where:
    - $V^{\upharpoonright} = V \backslash \{v \in V | v \in \lfloor v \rfloor\}$
    - $E^{\upharpoonright} = E \backslash \{\langle v_1, v_2\rangle | v_1 = v\}$
    - $r^{\upharpoonright} = r$
- Then, denote by $\beta(\alpha)_v$ the adjunction of β into α at node v defined in the following way:
  - If $l(v) \neq l(r')$, then this operation is undefined
  - Otherwise, do $\alpha \upharpoonright v\,[v, \beta]$, and call γ the resulting tree, then, do $\gamma\,[*, \alpha/v]$
- This operation can always be iterated

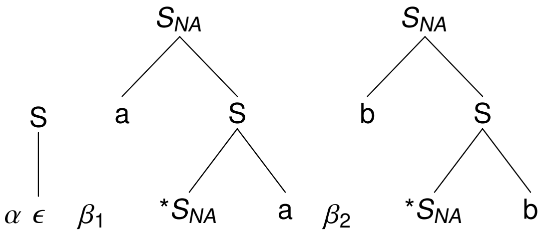
Adjunction

In a picture



Example

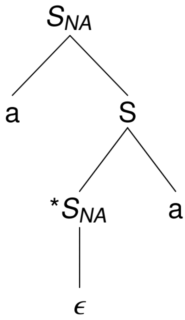
Copy Language



Adjunction

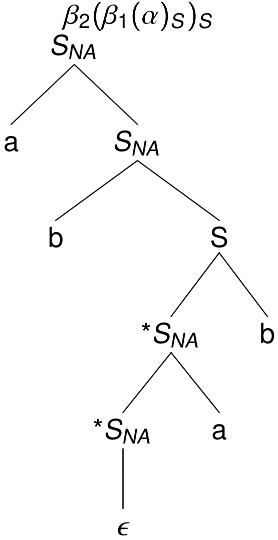
Derived tree

$\beta_1(\alpha)_S$



Adjunction

Derived tree



- Tree Substitution Grammars (TSG)

- Tree Substitution Grammars (TSG)
  - CFG can be seen as a special case of TSG

- Tree Substitution Grammars (TSG)
  - CFG can be seen as a special case of TSG
  - TSG's are not enough to strongly lexicalize CFG

- Tree Substitution Grammars (TSG)
  - CFG can be seen as a special case of TSG
  - TSG's are not enough to strongly lexicalize CFG
- TAG'S can be restricted to only the Adjunction operation

- Tree Substitution Grammars (TSG)
  - CFG can be seen as a special case of TSG
  - TSG's are not enough to strongly lexicalize CFG
- TAG'S can be restricted to only the Adjunction operation
  - Substitution can be simulated by adjunction.

- Tree Substitution Grammars (TSG)
  - CFG can be seen as a special case of TSG
  - TSG's are not enough to strongly lexicalize CFG
- TAG'S can be restricted to only the Adjunction operation
  - Substitution can be simulated by adjunction.
  - Restrictions over adjunction. No Adjunction. Obligatory Adjunction, Selected Adjunction.

- Tree Substitution Grammars (TSG)
  - CFG can be seen as a special case of TSG
  - TSG's are not enough to strongly lexicalize CFG
- TAG'S can be restricted to only the Adjunction operation
  - Substitution can be simulated by adjunction.
  - Restrictions over adjunction. No Adjunction. Obligatory Adjunction, Selected Adjunction.
  - TAG's do Strongly lexicalize CFG

- Tree Substitution Grammars (TSG)
  - CFG can be seen as a special case of TSG
  - TSG's are not enough to strongly lexicalize CFG
- TAG'S can be restricted to only the Adjunction operation
  - Substitution can be simulated by adjunction.
  - Restrictions over adjunction. No Adjunction. Obligatory Adjunction, Selected Adjunction.
  - TAG's do Strongly lexicalize CFG
- Parsable in $\mathcal{O}(n^6)$ over the length of the string to be parsed

- Tree Substitution Grammars (TSG)
  - CFG can be seen as a special case of TSG
  - TSG's are not enough to strongly lexicalize CFG
- TAG'S can be restricted to only the Adjunction operation
  - Substitution can be simulated by adjunction.
  - Restrictions over adjunction. No Adjunction. Obligatory Adjunction, Selected Adjunction.
  - TAG's do Strongly lexicalize CFG
- Parsable in $\mathcal{O}(n^6)$ over the length of the string to be parsed
- Constant growth property ?

Properties

- Let L, and L' be tree adjoining languages, then the following are TAL as well:

Properties

- Let L, and L' be tree adjoining languages, then the following are TAL as well:
- $L \cap L'$

Properties

- Let L, and L' be tree adjoining languages, then the following are TAL as well:
- $L \cap L'$
- $L \cup L'$

Properties

- Let L, and L' be tree adjoining languages, then the following are TAL as well:
- $L \cap L'$
- $L \cup L'$
- L^*

Properties

- Let L, and L' be tree adjoining languages, then the following are TAL as well:
- $L \cap L'$
- $L \cup L'$
- L^*
- $L \cdot L'$

Properties

- Let L, and L' be tree adjoining languages, then the following are TAL as well:
- $L \cap L'$
- $L \cup L'$
- L^*
- $L \cdot L'$
- $L \cap R$ with R a regular language

- Extended domain of locality. A domain over which various dependencies (syntactic and semantic) can be specified. Joshi, A. K. ‘Domains of Locality’, *Data and Knowledge Engineering* Vol. 50 Issue 3 2004

- Extended domain of locality. A domain over which various dependencies (syntactic and semantic) can be specified. Joshi, A. K. ‘Domains of Locality’, *Data and Knowledge Engineering* Vol. 50 Issue 3 2004
- Altough here and there it has been stated that TAG’s are closed under strong lexicalization, recently, it has been shown that TAG’s are NOT closed under strong lexicalization:
  Kuhlman and Satta ‘Tree-Adjoining Grammars are not closed under strong lexicalization’*Computational Linguistics* 2012

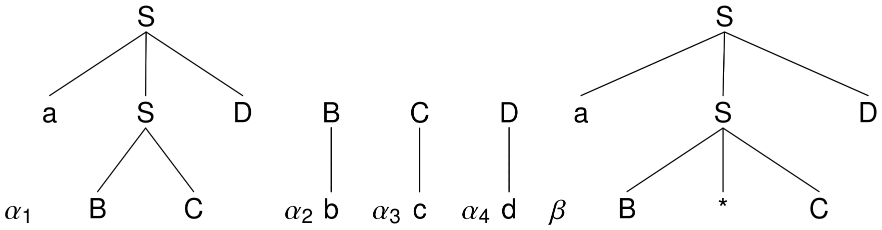
- The class of dependency structures induced by this formalism is the class of well-nested structures with block degree at most two.
  $\mathcal{D}(\mathit{TAG}) = \mathcal{D}_2 \cap \mathcal{D}_{wn}$

- The class of dependency structures induced by this formalism is the class of well-nested structures with block degree at most two. $\mathcal{D}(TAG) = \mathcal{D}_2 \cap \mathcal{D}_{wn}$
- Traversal Strategy: BLOCK-ORDER-COLLECT

- The class of dependency structures induced by this formalism is the class of well-nested structures with block degree at most two. $\mathcal{D}(TAG) = \mathcal{D}_2 \cap \mathcal{D}_{wn}$
- Traversal Strategy: BLOCK-ORDER-COLLECT
- Derivation Trees are terms over the signature of elementary trees.

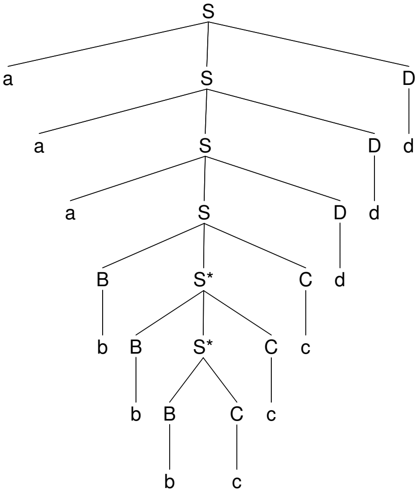
$\{a^n b^n c^n d^n | n \in \mathbb{N}\}$

Running Example



Derived Tree

aaabbbccc

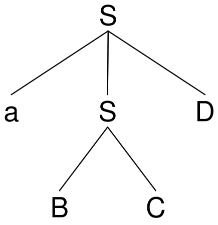


Linearization

- For the linearization of the elementaries trees in the grammar we should keep in mind the way we will produce a string by means fo that tree.

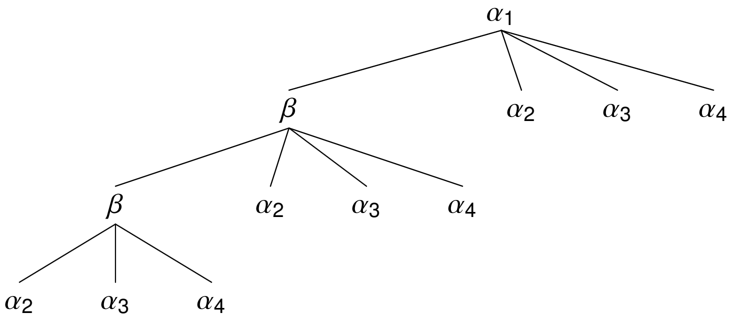
Linearization

- For the linearization of the elementaries trees in the grammar we should keep in mind the way we will produce a string by means fo that tree.

- α_1 

  aS_1BCS_2D

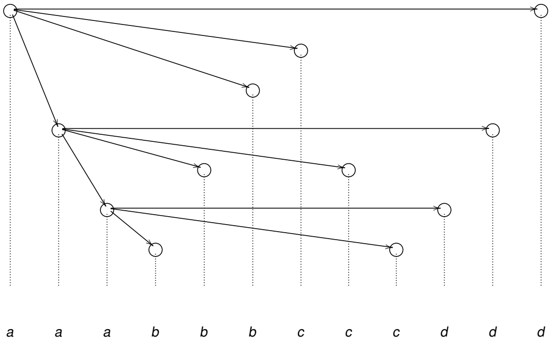
Term



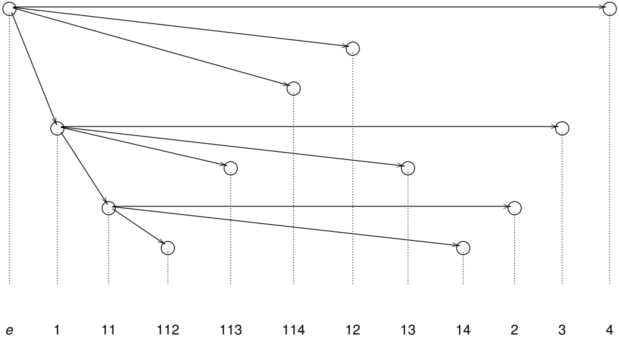
- Recall that an order anotation ω is well-nested if it does not contain a string of the form *ijij* as a scattered substring, for $i \neq j \in \mathbb{N}$ Denote by Ω_{wn} the set of all well-nested order anotations.

- Recall that an order anotation ω is well-nested if it does not contain a string of the form *ijij* as a scattered substring, for $i \neq j \in \mathbb{N}$ Denote by Ω_{wn} the set of all well-nested order anotations.
- The set of order anotations for the terms of this formalism is well-nested.

- Recall that an order anotation ω is well-nested if it does not contain a string of the form *ijij* as a scattered substring, for $i \neq j \in \mathbb{N}$ Denote by Ω_{wn} the set of all well-nested order anotations.
- The set of order anotations for the terms of this formalism is well-nested.
- **Theorem 5.2.1** A dependency structure $\mathcal{D}$ is well-nested if and only if $term(\mathcal{D}) \in T_{\Omega_{wn}}$


a a a b b b c c c d d d

Linearization



Block Degree

- Recall the coarsest congruence relation over a set that we are using: **Lemma 4.1.1** Let $\mathfrak{C} = (A; \leq)$ be a chain. $S \subseteq A$. Define $\equiv_S$ as follows: $a \equiv_S b$ iff $\forall c \in [a, b]$, $c \in S$

Block Degree

- Recall the coarsest congruence relation over a set that we are using: **Lemma 4.1.1** Let $\mathfrak{C} = (A; \leq)$ be a chain. $S \subseteq A$. Define $\equiv_S$ as follows: $a \equiv_S b$ iff $\forall c \in [a, b], c \in S$
- To visualize the segmentation of D, consider the the congruence relation of $\lfloor u \rfloor / \equiv_{\lfloor u \rfloor}$

Block Degree

- Recall the coarsest congruence relation over a set that we are using: **Lemma 4.1.1** Let $\mathfrak{C} = (A; \leq)$ be a chain. $S \subseteq A$. Define $\equiv_S$ as follows: $a \equiv_S b$ iff $\forall c \in [a, b], c \in S$
- To visualize the segmentation of D, consider the the congruence relation of $\lfloor u \rfloor / \equiv_{\lfloor u \rfloor}$
- Note that $\lfloor 11 \rfloor / \equiv_{\lfloor 11 \rfloor}$ consists of two blocks, and any other node has block degree one.

Block Degree

- Recall the coarsest congruence relation over a set that we are using: **Lemma 4.1.1** Let $\mathfrak{C} = (A; \leq)$ be a chain. $S \subseteq A$. Define $\equiv_S$ as follows: $a \equiv_S b$ iff $\forall c \in [a, b]$, $c \in S$
- To visualize the segmentation of D, consider the the congruence relation of $\lfloor u \rfloor / \equiv_{\lfloor u \rfloor}$
- Note that $\lfloor 11 \rfloor / \equiv_{\lfloor 11 \rfloor}$ consists of two blocks, and any other node has block degree one.
- Thus the block degree of our dependency structure is two.

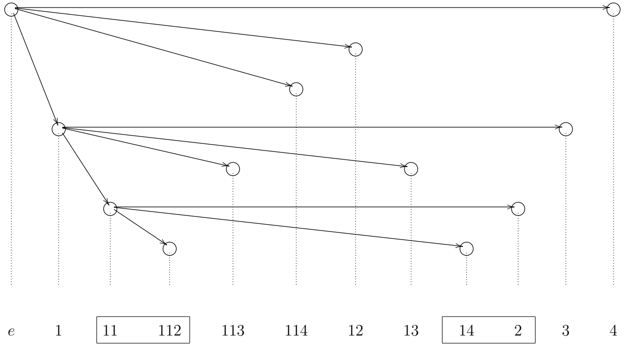


Figure 1: Blocks of node 11

Thanks