Introduction to Tree Adjoining Grammar

Natural Language Syntax with TAG

Wolfgang Maier and Timm Lichte
University of Düsseldorf

DGfS-CL Fall School 2011

1st week, 2nd session
Aug 30, 2011

Outline

Adjunction and substitution (1)

We have seen that

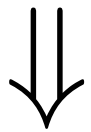
1. Using larger trees and allowing only substitution does not extend the weak capacity of CFG.
2. With TSGs, we cannot build satisfying lexicalized grammars.

Adjunction and substitution (1)

We have seen that

1. Using larger trees and allowing only substitution does not extend the weak capacity of CFG.
2. With TSGs, we cannot build satisfying lexicalized grammars.

We cannot add modifiers in the middle of elementary trees, only the leaves can be replaced with new trees.

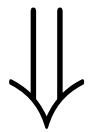


Adjunction and substitution (1)

We have seen that

1. Using larger trees and allowing only substitution does not extend the weak capacity of CFG.
2. With TSGs, we cannot build satisfying lexicalized grammars.

We cannot add modifiers in the middle of elementary trees, only the leaves can be replaced with new trees.



Therefore, we now add a second operation on trees called adjunction.

Adjunction and substitution (2)

Tree Adjoining Grammars (TAG)

Tree-rewriting system: set of elementary trees with two operations:

- adjunction: replacing an internal node with a new tree. The new tree is an auxiliary tree and has a special leaf, the foot node.
- substitution: replacing a leaf with a new initial tree.

Adjunction and substitution (2)

Tree Adjoining Grammars (TAG)

Tree-rewriting system: set of elementary trees with two operations:

- adjunction: replacing an internal node with a new tree. The new tree is an auxiliary tree and has a special leaf, the foot node.
- substitution: replacing a leaf with a new initial tree.

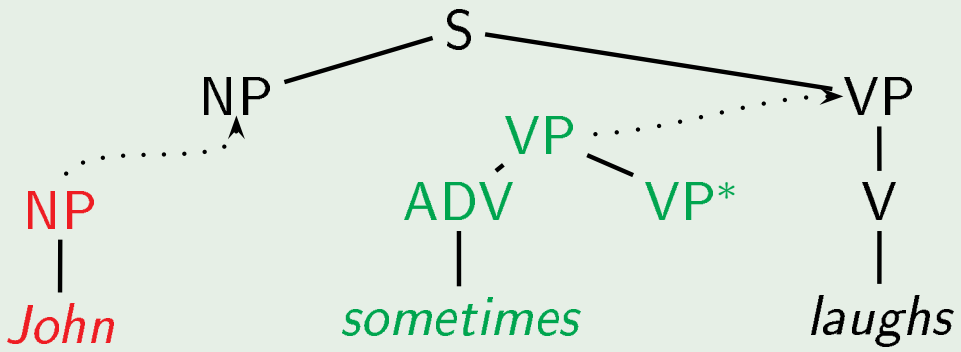
[Joshi et al., 1975, Joshi and Schabes, 1997]

Adjunction and substitution (3)

(1) John sometimes laughs

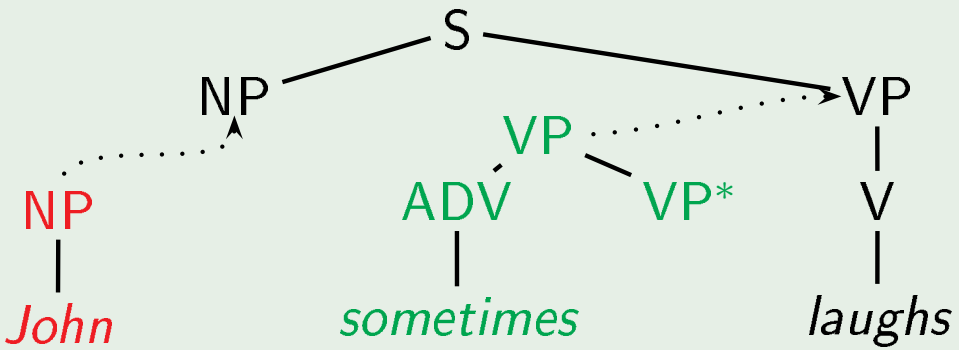
Adjunction and substitution (3)

(1) John sometimes laughs

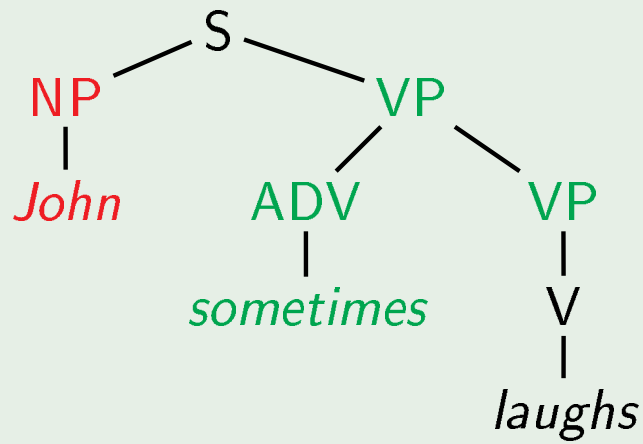


Adjunction and substitution (3)

(1) John sometimes laughs



derived tree:



Adjunction and substitution (4)

Definition (Auxiliary and initial trees)

1. *A syntactic tree is an ordered labeled tree such that $l(v) \in N$ for each vertex v with out-degree at least 1 and $l(v) \in (N \cup T \cup \{\varepsilon\})$ for each leaf v.*
2. *An auxiliary tree is a syntactic tree that has a unique leaf marked as foot node. The foot node must have the same label as the root node.*
3. *An initial tree is a non-auxiliary syntactic tree.*

As a convention, the foot node is marked with a "*".

Tree Adjoining Grammar (1)

Definition (Tree Adjoining Grammar)

A Tree Adjoining Grammar (TAG) is a tuple $G = \langle N, T, S, I, A \rangle$ *such that*

- T *and* N *are disjoint alphabets, the terminals and nonterminals,*
- $S \in N$ *is the start symbol,*
- I *is a finite set of initial trees, and*
- A *is a finite set of auxiliary trees.*

The trees in $I \cup A$ *are called elementary trees.*
G *is lexicalized iff each elementary tree has at least one leaf with a terminal label.*

Tree Adjoining Grammar (2)

- Every elementary tree is considered a derived tree in a TAG. Depending on whether is has a foot node or not, it is a derived auxiliary or a derived initial tree.

Tree Adjoining Grammar (2)

- Every elementary tree is considered a derived tree in a TAG. Depending on whether is has a foot node or not, it is a derived auxiliary or a derived initial tree.
- In every derivation step, we pick a fresh instance of an elementary tree from the grammar and we add derived trees (by substitution or adjunction) to certain nodes in this tree.

Tree Adjoining Grammar (2)

- Every elementary tree is considered a derived tree in a TAG. Depending on whether is has a foot node or not, it is a derived auxiliary or a derived initial tree.
- In every derivation step, we pick a fresh instance of an elementary tree from the grammar and we add derived trees (by substitution or adjunction) to certain nodes in this tree.
- The trees in the tree language are the derived initial trees with root label S and only with terminal leaf labels.

We write a tree obtained by substituting or adjoining γ' into γ at node v as $\gamma[v, \gamma']$.

Tree Adjoining Grammar (3)

Definition (Derived tree)

Let $G = \langle N, T, S, I, A \rangle$ be a TAG.

1. *Every instance γ of a $\gamma_e \in I \cup A$ is a derived tree in G.*
2. *For pairwise disjoint $\gamma_1, \ldots, \gamma_n, \gamma$ such that $\gamma_1, \ldots, \gamma_n$ are derived trees in G $(1 \leq i \leq n)$ and γ is an instance of a $\gamma_e \in I \cup A$ containing pairwise different nodes $v_1, \ldots, v_n$: if $\gamma' = \gamma[v_1, \gamma_1] \ldots [v_n, \gamma_n]$ is defined then γ' is a derived tree in G.*
3. *These are all derived trees in G.*

Note that this definition adopts a bottom-up perspective: derived trees are added to elementary trees.

Tree Adjoining Grammar (4)

Definition (TAG language)

Let $G = \langle N, T, S, I, A \rangle$ *be a TAG.*

Tree Adjoining Grammar (4)

Definition (TAG language)

Let $G = \langle N, T, S, I, A \rangle$ *be a TAG.*

1. *The tree language* $L_T(G)$ *of* G *is the set of all derived initial trees* γ *in* G *with root label* S *and only terminal leaf labels.*

Tree Adjoining Grammar (4)

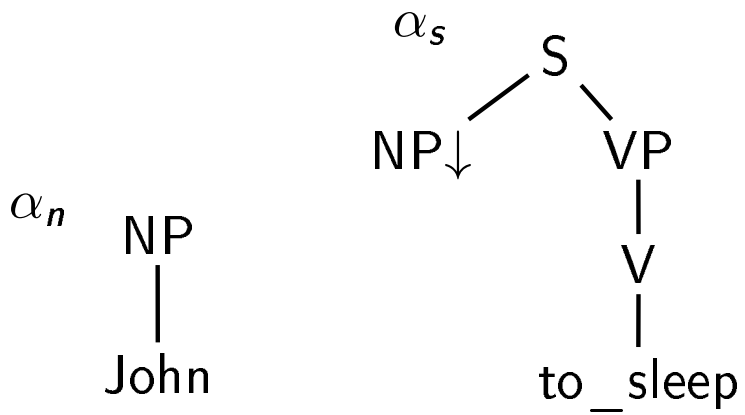
Definition (TAG language)

Let $G = \langle N, T, S, I, A \rangle$ *be a TAG.*

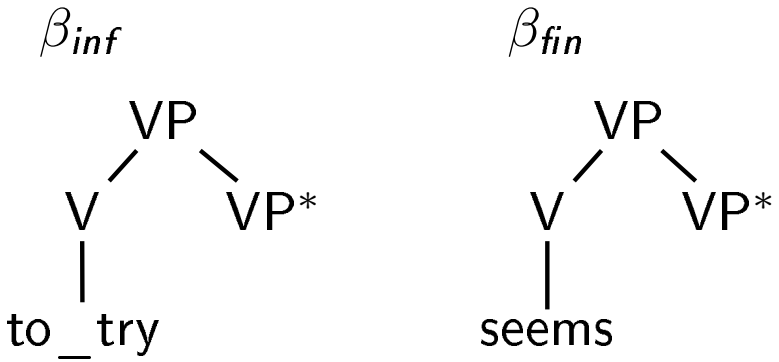
1. *The tree language* $L_T(G)$ *of* G *is the set of all derived initial trees* γ *in* G *with root label* S *and only terminal leaf labels.*
2. *The string language* $L_S(G)$ *of* G *is* $\{w \mid \text{there is a } \gamma \in L_T(G) \text{ such that } w = yield(\gamma)\}$.

Tree Adjoining Grammar (5)

Initial trees:



Auxiliary trees:



Adjunction constraints (1)

TAGs as defined above are more powerful than CFG but they cannot generate the copy language.

Adjunction constraints (1)

TAGs as defined above are more powerful than CFG but they cannot generate the copy language.
In order to increase the expressive power, adjunction constraints are introduced that specify for each node

1. whether adjunction is mandatory and
2. which trees can be adjoined.

Adjunction constraints (1)

TAGs as defined above are more powerful than CFG but they cannot generate the copy language.
In order to increase the expressive power, adjunction constraints are introduced that specify for each node

1. whether adjunction is mandatory and
2. which trees can be adjoined.

For a given node,

1. the function f_{OA} specifies whether adjunction is obligatory (value 1) or not (value 0) and
2. the function f_{SA} gives the set of auxiliary trees that can be adjoined.

Adjunction constraints (2)

Definition (TAG with adjunction constraints)

A TAG with adjunction constraints is a tuple $G = \langle N, T, S, I, A, f_{OA}, f_{SA} \rangle$ *where*

- $\langle N, T, S, I, A \rangle$ *is a TAG as defined above and*
- $f_{OA} : \{v \mid v \text{ is a node in some } \gamma \in I \cup A\} \rightarrow \{0, 1\}$ *and* $f_{SA} : \{v \mid v \text{ is a node in some } \gamma \in I \cup A\} \rightarrow \mathcal{P}(A)$ *where* $\mathcal{P}(A)$ *is the set of subsets of* A *are functions such that* $f_{OA}(v) = 0$ *and* $f_{SA}(v) = \emptyset$ *for every leaf* v.

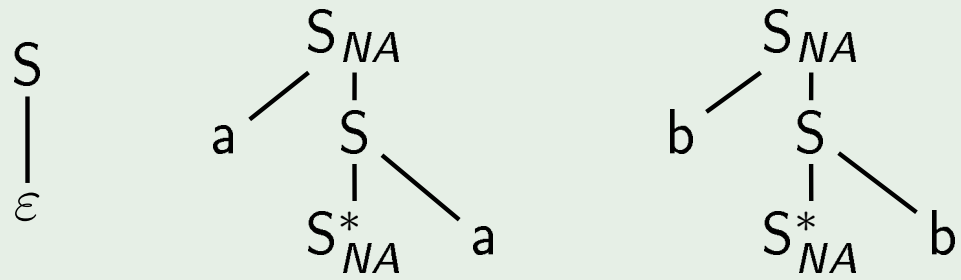
Adjunction constraints (3)

Three types of constraints are distinguished:

- A node v with $f_{OA}(v) = 1$ is said to carry a obligatory adjunction (OA) constraint.
- A node v with $f_{OA}(v) = 0$ and $f_{SA}(v) = \emptyset$ is said to carry a null adjunction (NA) constraint.
- A node v with $f_{OA}(v) = 0$ and $f_{SA}(v) \neq \emptyset$ and $f_{SA}(v) \neq A$ is said to carry a selective adjunction (SA) constraint.

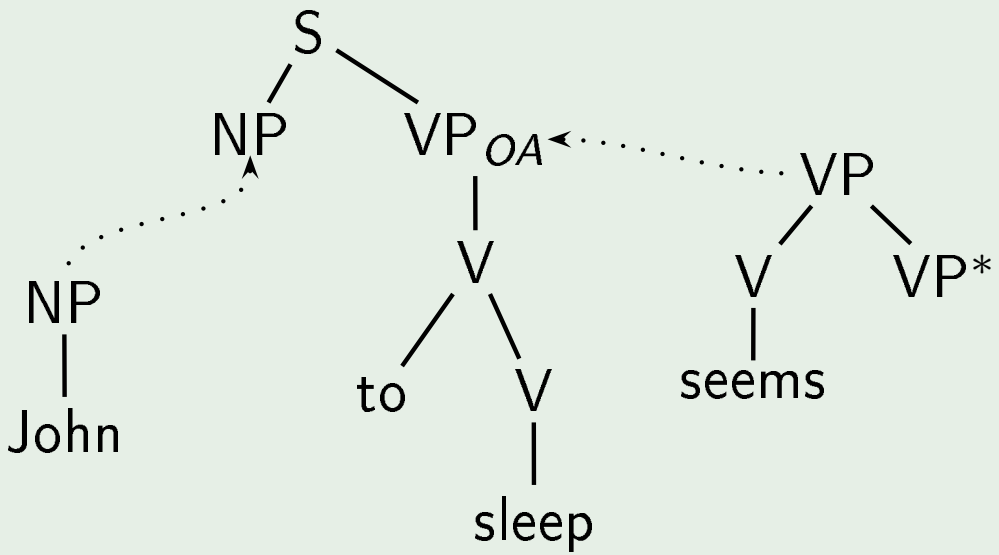
Adjunction constraints (4)

TAG for the copy language



Adjunction constraints (5)

(2) John seems to sleep



Adjunction constraints (6)

Definition (Derived tree)

Let $G = \langle N, T, S, I, A, f_{OA}, f_{SA} \rangle$ be a TAG with adjunction constraints.

1. *Every instance of a $\gamma_e \in I \cup A$ is a derived tree obtained from γ_e in G.*
2. *For pairwise disjoint $\gamma_1, \ldots, \gamma_n, \gamma$ such that a) $\gamma_1, \ldots, \gamma_n$ are derived trees obtained from $\gamma_1^e, \ldots, \gamma_n^e$ in G respectively, and b) γ is an instance of a $\gamma_e \in I \cup A$ such that $v_1, \ldots, v_n \in V$ are pairwise different nodes: If*
   - *$\gamma' = \gamma[v_1, \gamma_1] \ldots [v_n, \gamma_n]$ is defined, and*
   - *for all $1 \leq i \leq n$: if γ_i is an auxiliary tree, then $\gamma_i^e \in f_{SA}(v_i)$*

   then γ' is a derived tree obtained from γ_e in G
3. *These are all derived trees in G.*

Adjunction constraints (7)

Definition (Tree language)

Let $G = \langle N, T, S, I, A, f_{OA}, f_{SA} \rangle$ be a TAG with adjunction constraints.
The tree language of G is the set of all derived initial trees γ in G such that

- *the root label of γ is S,*
- *$f_{OA}(v) = 0$ for all nodes v in γ, and*
- *all leaves in γ have terminal labels.*

In the following, whenever we use the term "TAG", this means "TAG with adjunction constraints".

Derivation trees (1)

TAG derivations are described by derivation trees:

- For each derivation in a TAG there is a corresponding derivation tree. This tree contains
  - nodes for all elementary trees used in the derivation, and
  - edges for all adjunctions and substitutions performed throughout the derivation.

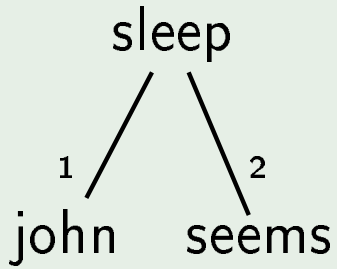
Derivation trees (1)

TAG derivations are described by derivation trees:

- For each derivation in a TAG there is a corresponding derivation tree. This tree contains
  - nodes for all elementary trees used in the derivation, and
  - edges for all adjunctions and substitutions performed throughout the derivation.
- Whenever an elementary tree γ was attached to the node at address p in the elementary tree γ', there is an edge from γ' to γ labeled with p.

Derivation trees (2)

derivation tree for the derivation of (2) John seems to sleep



Derivation trees (3)

Derivation trees are defined in parallel to the derived trees:

Definition (Derived tree)

Let $G = \langle N, T, S, I, A, f_{OA}, f_{SA} \rangle$ *be a TAG.*

1. *Every instance of a* $\gamma_e \in I \cup A$ *is a derived tree in* G.
   The corresponding derivation tree is a single node with label γ_e.

Derivation trees (3)

Definition (Derived tree)

1. *For pairwise disjoint $\gamma_1, \ldots, \gamma_n, \gamma$ such that a) $\gamma_1, \ldots, \gamma_n$ are derived trees whose derivation trees $D_1, \ldots, D_n$ have root labels $\gamma_1^e, \ldots, \gamma_n^e$ resp. and γ is an elementary tree instance such that $v_1, \ldots, v_n$ are pairwise different nodes in γ with Gorn addresses $p_1, \ldots, p_n$: if*

   - $\gamma' = \gamma[v_1, \gamma_1] \ldots [v_n, \gamma_n]$ *is defined and*
   - *for all $1 \leq i \leq n$: if γ_i is an auxiliary tree, then $\gamma_i^e \in f_{SA}(v_i)$*

   then γ' is a derived tree in G with a corresponding derivation tree having a root r_0 with label γ_e and the n daughter trees $D_1, \ldots, D_n$ resp. such that the edge from r_0 to the root of D_i is labeled with p_i for all $1 \leq i \leq n$.

2. *These are all pairs of derived trees and derivation trees in G.*

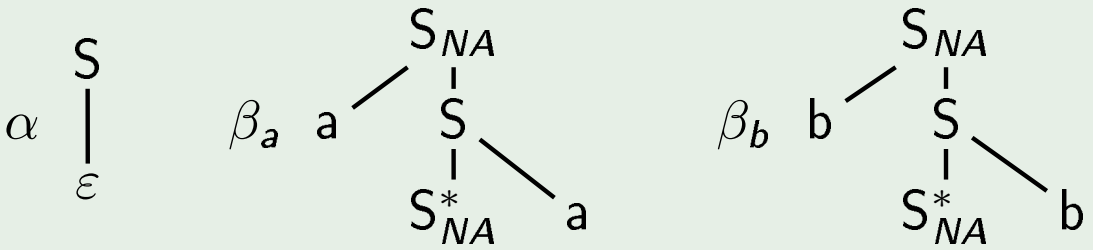
Derivation trees (4)

Derivation trees are unordered trees. They

- uniquely determine a derived tree (but not vice versa), and
- are context-free, i.e., for every TAG, one can find a CFG whose tree language is, except for adding ε-leaves and ordering the trees, the set of derivation trees of the TAG.

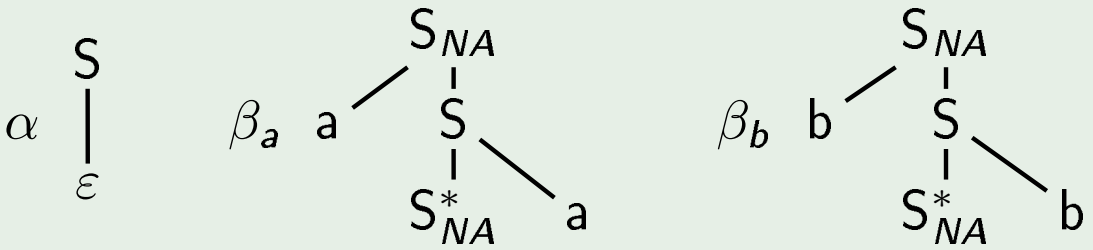
Derivation trees (5)

Example

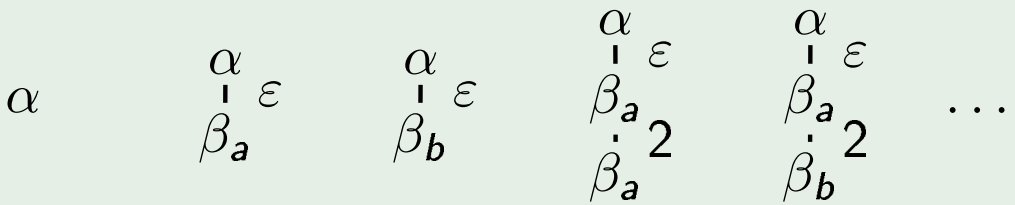


Derivation trees (5)

Example

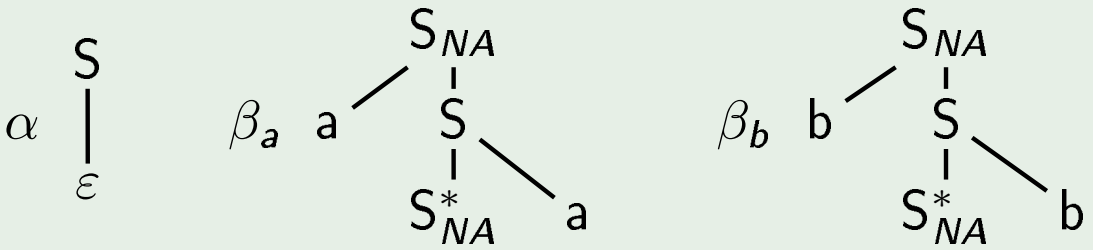


Derivation trees:

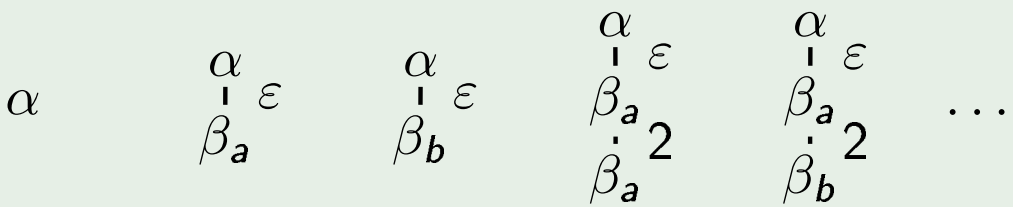


Derivation trees (5)

Example



Derivation trees:



CFG: $\alpha \to \varepsilon$ $\alpha \to \beta_a$ $\alpha \to \beta_b$ $\beta_a \to \varepsilon$ $\beta_a \to \beta_a$ $\beta_a \to \beta_b$ $\beta_b \to \varepsilon$ $\beta_b \to \beta_a$ $\beta_b \to \beta_b$

Formal Properties (1)

Languages TAG can generate

- $\{ww \mid w \in \{a, b\}^*\}$
- $L_4 := \{a^n b^n c^n d^n \mid n \geq 0\}$

Formal Properties (1)

Languages TAG can generate

- $\{ww \mid w \in \{a, b\}^*\}$
- $L_4 := \{a^n b^n c^n d^n \mid n \geq 0\}$

Languages TAG cannot generate

- $\{w^n \mid w \in \{a, b\}^*\}$ for any $n > 2$.
  $\Rightarrow$ TAG generate only a limited amount of cross-serial dependencies
- $L_k := \{a_1^n a_2^n a_3^n \ldots a_k^n \mid n \geq 0\}$ for any $k > 4$.
  $\Rightarrow$ TAG can "count up to 4, not further".
- $L := \{a^{2^n} \mid n \geq 0\}$.
  $\Rightarrow$ TAG cannot generate languages whose word lengths grow exponentially.

Formal Properties (2)

$\Rightarrow$ TAG extend CFG but only in a limited way.

In order to situate a class of languages with respect to other classes, one needs to know something about the properties of this class. Particularly useful:

- Pumping Lemmas
- Closure Properties

Pumping Lemma for TAL (1)

Definition (Pumping lemma for CFL)

Let L be a context-free language. Then there is a constant c such that for all $w \in L$ with $|w| \geq c$: $w = xv_1yv_2z$ with

- $|v_1 v_2| \geq 1$,
- $|v_1 y v_2| \leq c$, *and*
- *for all* $i \geq 0$: $xv_1^i yv_2^i z \in L$.

Pumping Lemma for TAL (1)

Definition (Pumping lemma for CFL)

Let L be a context-free language. Then there is a constant c such that for all $w \in L$ with $|w| \geq c$: $w = xv_1yv_2z$ with

- $|v_1 v_2| \geq 1$,
- $|v_1 y v_2| \leq c$, *and*
- *for all* $i \geq 0$: $xv_1^i yv_2^i z \in L$.

- In the context-free tree, from a certain tree height on, there is always a path with two occurences of the same non-terminal.

Pumping Lemma for TAL (1)

Definition (Pumping lemma for CFL)

Let L be a context-free language. Then there is a constant c such that for all $w \in L$ with $|w| \geq c$: $w = xv_1yv_2z$ with

- $|v_1 v_2| \geq 1$,
- $|v_1 y v_2| \leq c$, *and*
- *for all* $i \geq 0$: $xv_1^i yv_2^i z \in L$.

- In the context-free tree, from a certain tree height on, there is always a path with two occurences of the same non-terminal.
- Then the part between the two occurrences can be iterated. This means that the strings to left and the right of this part are pumped.

Pumping Lemma for TAL (2)

How about TAL?

- The TAG derivation trees are context-free.
- Therefore, the same iteration is possible here.

Pumping Lemma for TAL (3)

One can show the following:

Proposition (Pumping Lemma for TAL)

If L is a TAL, then there is a constant c such that if $w \in L$ and $|w| \geq c$, then there are $x, y, z, v_1, v_2, w_1, w_2, w_3, w_4 \in T^*$ s. t.

- $|v_1 v_2 w_1 w_2 w_3 w_4| \leq c$,
- $|w_1 w_2 w_3 w_4| \geq 1$,
- $x = xv_1 yv_2 z$, *and*
- $xw_1^n v_1 w_2^n y w_3^n v_2 w_4^n z \in L(G)$ *for all* $n \geq 0$.

Pumping Lemma for TAL (4)

Pumping lemmas can be used to show that certain languages are not in a certain class.

Example

- To show: $L = \{a^n b^n c^n d^n e^n \mid n \geq 0\}$ is not a TAL.

Pumping Lemma for TAL (4)

Pumping lemmas can be used to show that certain languages are not in a certain class.

Example

- To show: $L = \{a^n b^n c^n d^n e^n \mid n \geq 0\}$ is not a TAL.
- Assume that L is a TAL and therefore satisfies the pumping lemma with a constant c. Take $w = a^{c+1} b^{c+1} c^{c+1} d^{c+1} e^{c+1}$.

Pumping Lemma for TAL (4)

Pumping lemmas can be used to show that certain languages are not in a certain class.

Example

- To show: $L = \{a^n b^n c^n d^n e^n \mid n \geq 0\}$ is not a TAL.
- Assume that L is a TAL and therefore satisfies the pumping lemma with a constant c. Take $w = a^{c+1} b^{c+1} c^{c+1} d^{c+1} e^{c+1}$.
- According to the P.L. one can find $w_1, \ldots, w_4$ such that
  - at least one of them is not empty, and
  - they can be inserted repeatedly at 4 positions into w yielding a word in L.

Pumping Lemma for TAL (4)

Pumping lemmas can be used to show that certain languages are not in a certain class.

Example

- To show: $L = \{a^n b^n c^n d^n e^n \mid n \geq 0\}$ is not a TAL.
- Assume that L is a TAL and therefore satisfies the pumping lemma with a constant c. Take $w = a^{c+1} b^{c+1} c^{c+1} d^{c+1} e^{c+1}$.
- According to the P.L. one can find $w_1, \ldots, w_4$ such that
  - at least one of them is not empty, and
  - they can be inserted repeatedly at 4 positions into w yielding a word in L.
- One of $w_1, \ldots, w_4$ must contain two different terminal symbols since altogether they must contain equal numbers of as, bs, cs, ds, and es.

Pumping Lemma for TAL (4)

Pumping lemmas can be used to show that certain languages are not in a certain class.

Example

- To show: $L = \{a^n b^n c^n d^n e^n \mid n \geq 0\}$ is not a TAL.
- Assume that L is a TAL and therefore satisfies the pumping lemma with a constant c. Take $w = a^{c+1} b^{c+1} c^{c+1} d^{c+1} e^{c+1}$.
- According to the P.L. one can find $w_1, \ldots, w_4$ such that
  - at least one of them is not empty, and
  - they can be inserted repeatedly at 4 positions into w yielding a word in L.
- One of $w_1, \ldots, w_4$ must contain two different terminal symbols since altogether they must contain equal numbers of as, bs, cs, ds, and es.
- Contradiction since at the second insertion, letters get mixed, i.e., we get a word $\notin L$.

Closure Properties (1)

It is often useful to reduce a language L to a simpler language before showing that it is not in a certain class C. This can be done with closure properties.

TAL are closed under

- union, concatenation, Kleene closure and substitution.
- homomorphisms, intersection with regular languages, and inverse homomorphisms.

$\Rightarrow$ TALs form a substitution closed Full Abstract Family of Languages (AFL). (Full AFL = closed under intersection with regular languages, homomorphisms, inverse homomorphisms, union, concatenation and Kleene star.)
[Vijay-Shanker and Joshi, 1985, Vijay-Shanker, 1987]

Closure Properties (1)

It is often useful to reduce a language L to a simpler language before showing that it is not in a certain class C. This can be done with closure properties.

TAL are closed under

- union, concatenation, Kleene closure and substitution.
- homomorphisms, intersection with regular languages, and inverse homomorphisms.

$\Rightarrow$ TALs form a substitution closed Full Abstract Family of Languages (AFL). (Full AFL = closed under intersection with regular languages, homomorphisms, inverse homomorphisms, union, concatenation and Kleene star.)
[Vijay-Shanker and Joshi, 1985, Vijay-Shanker, 1987]

Closure Properties (1)

It is often useful to reduce a language L to a simpler language before showing that it is not in a certain class C. This can be done with closure properties.

TAL are closed under

- union, concatenation, Kleene closure and substitution.
- homomorphisms, intersection with regular languages, and inverse homomorphisms.

$\Rightarrow$ TALs form a substitution closed Full Abstract Family of Languages (AFL). (Full AFL = closed under intersection with regular languages, homomorphisms, inverse homomorphisms, union, concatenation and Kleene star.)
[Vijay-Shanker and Joshi, 1985, Vijay-Shanker, 1987]

Closure Properties (1)

It is often useful to reduce a language L to a simpler language before showing that it is not in a certain class C. This can be done with closure properties.

TAL are closed under

- union, concatenation, Kleene closure and substitution.
- homomorphisms, intersection with regular languages, and inverse homomorphisms.

$\Rightarrow$ TALs form a substitution closed Full Abstract Family of Languages (AFL). (Full AFL = closed under intersection with regular languages, homomorphisms, inverse homomorphisms, union, concatenation and Kleene star.)
[Vijay-Shanker and Joshi, 1985, Vijay-Shanker, 1987]

Closure Properties (2)

The argumentation to show that L is not in a class C goes then as follows:

- Assume that L is in C.
- Then (supposing C is closed under operation f), $L' = f(L)$ is also in C.
- If we know that L' is not in C, this is a contradiction.

Consequently, L is not in C.

Closure Properties (3)

To show: the double copy language $L = \{www \mid w \in \{a, b\}^*\}$ is not in *TAL*.

- Assume that L is in *TAL*.
- Then (since *TAL* is closed under intersection with regular languages), the language $L' := L \cap a^*b^*a^*b^*a^*b^* = \{a^n b^m a^n b^m a^n b^m \mid n, m \geq 0\}$ is in *TAL* as well.
- Contradiction since L' does not satisfy the pumping lemma for *TAL*.

Consequently, L is not in *TAL*.

Closure Properties (3)

To show: the double copy language $L = \{www \mid w \in \{a, b\}^*\}$ is not in *TAL*.

- Assume that L is in *TAL*.
- Then (since *TAL* is closed under intersection with regular languages), the language $L' := L \cap a^*b^*a^*b^*a^*b^* = \{a^n b^m a^n b^m a^n b^m \mid n, m \geq 0\}$ is in *TAL* as well.
- Contradiction since L' does not satisfy the pumping lemma for *TAL*.

Consequently, L is not in *TAL*.

Closure Properties (3)

To show: the double copy language $L = \{www \mid w \in \{a, b\}^*\}$ is not in *TAL*.

- Assume that L is in *TAL*.
- Then (since *TAL* is closed under intersection with regular languages), the language $L' := L \cap a^*b^*a^*b^*a^*b^* = \{a^n b^m a^n b^m a^n b^m \mid n, m \geq 0\}$ is in *TAL* as well.
- Contradiction since L' does not satisfy the pumping lemma for *TAL*.

Consequently, L is not in *TAL*.

Closure Properties (3)

To show: the double copy language $L = \{www \mid w \in \{a, b\}^*\}$ is not in *TAL*.

- Assume that L is in *TAL*.
- Then (since *TAL* is closed under intersection with regular languages), the language $L' := L \cap a^*b^*a^*b^*a^*b^* = \{a^n b^m a^n b^m a^n b^m \mid n, m \geq 0\}$ is in *TAL* as well.
- Contradiction since L' does not satisfy the pumping lemma for *TAL*.

Consequently, L is not in *TAL*.

Closure Properties (3)

To show: the double copy language $L = \{www \mid w \in \{a, b\}^*\}$ is not in *TAL*.

- Assume that L is in *TAL*.
- Then (since *TAL* is closed under intersection with regular languages), the language $L' := L \cap a^*b^*a^*b^*a^*b^* = \{a^n b^m a^n b^m a^n b^m \mid n, m \geq 0\}$ is in *TAL* as well.
- Contradiction since L' does not satisfy the pumping lemma for *TAL*.

Consequently, L is not in *TAL*.


Joshi, A. K., Levy, L. S., and Takahashi, M. (1975).
Tree Adjunct Grammars.
Journal of Computer and System Science, 10:136–163.


Joshi, A. K. and Schabes, Y. (1997).
Tree-Adjoining Grammars.
In Rozenberg, G. and Salomaa, A., editors, Handbook of Formal Languages, pages 69–123. Springer, Berlin.


Vijay-Shanker, K. (1987).
A Study of Tree Adjoining Grammars.
PhD thesis, University of Pennsylvania.


Vijay-Shanker, K. and Joshi, A. K. (1985).
Some computational properties of Tree Adjoining Grammars.
In Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics, pages 82–93.