Tree Adjoining Grammars

Motivation for TAG

Laura Kallmeyer & Benjamin Burkhardt

HHU Düsseldorf

WS 2017/2018

Outline

1. Why CFG is not enough
2. Tree Substitution Grammars
3. Tree Adjoining Grammars
   - Adjunction and substitution
   - Adjunction constraints

Why CFG is not enough

... for treating natural language:

1. only atomic non-terminals
2. only weak lexicalization (lexicalization challenge)
3. expressive power is too low (expressivity challenge)

Why CFG is not enough (1) - Atomic non-terminals

$S \rightarrow NP\ VP$ $NP \rightarrow \mathit{John}$ $NP \rightarrow \mathit{Mary}$
$VP \rightarrow V$ $VP \rightarrow V\ NP$ $V \rightarrow \mathit{sleeps}$ $V \rightarrow \mathit{likes}$

Possible derivation:
$S \Rightarrow NP\ VP \Rightarrow \mathit{John}\ VP \Rightarrow \mathit{John}\ V \Rightarrow \mathit{John\ sleeps}$
$S \overset{*}{\Rightarrow} \mathit{John\ likes\ Mary}$
$S \overset{*}{\Rightarrow} \mathit{John\ sleeps\ Mary}$

How to treat subcategorization frames, number agreement, and case marking?

(1) a. Kim depends on Sandy.
*Kim depends Sandy.
*Kim depends.
b. *The children depends on Sandy.
c. Kim depends on her/*she.

Why CFG is not enough (1)

How to treat subcategorization frames, number agreement, and case marking?

$\Longrightarrow$ encode the necessary information into the non-terminal symbols

$$S \to NP_{3sg/nom}\ VP_{3sg/itr} \qquad S \to NP_{3sg/nom}\ VP_{3sg/tr}$$
$$VP_{3sg/itr} \to V_{3sg/itr} \qquad VP_{3sg/tr} \to V_{3sg/tr}\ NP_{3sg/acc}$$
$$NP_{3sg/nom} \to \textit{John} \qquad NP_{3sg/acc} \to \textit{Mary}$$
$$V_{3sg/itr} \to \textit{sleeps} \qquad V_{3sg/tr} \to \textit{likes}$$

$S \overset{*}{\Rightarrow} \textit{John likes Mary}$
$S \overset{*}{\Rightarrow} \textit{John sleeps}$

Drawback: Every possible combination of subcategorization frame, number agreement, and case marking necessitates its own rule (let alone the number of non-terminal symbols).

Why CFG is not enough (1)

Example from German: $NP \rightarrow D\ N$ (determiner noun pairs)
Müller(2007) presents a CFG with 48 non-terminal symbols and 24 rules!

$NP_{3sg/nom} \rightarrow D_{fem/sg/nom}\ N_{fem/sg/nom}$
$NP_{3sg/nom} \rightarrow D_{masc/sg/nom}\ N_{masc/sg/nom}$
$NP_{3sg/nom} \rightarrow D_{neu/sg/nom}\ N_{neu/sg/nom}$
$NP_{3pl/nom} \rightarrow D_{fem/pl/nom}\ N_{fem/pl/nom}$
$NP_{3pl/nom} \rightarrow D_{masc/pl/nom}\ N_{masc/pl/nom}$
$NP_{3pl/nom} \rightarrow D_{neu/pl/nom}\ N_{neu/pl/nom}$
...

$\Longrightarrow$ grammar writing is tedious and error prone
$\Longrightarrow$ generalizations are hardly expressible

Remedy: feature structures instead of atomic non-terminal symbols, unification, underspecification

Why CFG is not enough (2) - Only weak lexicalization

Lexicalization

In a lexicalized grammar, each element of the grammar contains at least one lexical item (terminal symbol).

G_1: $S \rightarrow SS$, $S \rightarrow a$
G_2: $S \rightarrow aS$, $S \rightarrow a$

- **Formally interesting:** A finite lexicalized grammar provides finitely many analyses for each string (finitely ambiguous).
- **Linguistically interesting:** Syntactic properties of lexical items can be accounted for more directly.
- **Computationally interesting:** The search space during parsing can be delimited (grammar filtering).

Why CFG is not enough (2)

Lexicalizing a CFG

- Greibach normal form: $A \rightarrow aB_1 \ldots B_k \ (k \geq 0)$
- weak lexicalization: string language is preserved
- strong lexicalization: tree structure is preserved

Question: can CFGs be lexicalized such that the set of trees remains the same (strong lexicalization)?

Answer: No. Only weak lexicalization (same string language).

G_1: $S \rightarrow SS$, $S \rightarrow a$
G_2: $S \rightarrow aS$, $S \rightarrow a$

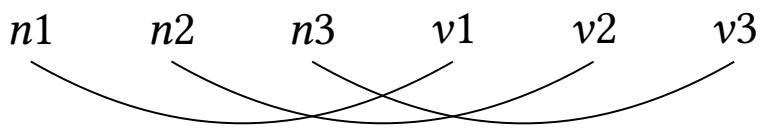
G_1 cannot be strongly lexicalized with some finite CFG, e.g. G_2.

Why CFG is not enough (3) - Low expressive power

Question: Are CFGs powerful enough to describe all natural language phenomena? **Answer:** No.

Example: **cross-serial dependencies** in Dutch and in Swiss German (Shieber, 1985)

(2) …mer d'chind em Hans es huus lönd hälfe aastriiche
…we children.ACC the Hans.DAT the house.ACC let help paint
'…we let the children help Hans paint the house'



A formalism that can generate cross-serial dependencies must be able to generate the copy language $\{ww \mid w \in \{a, b\}^*\}$.

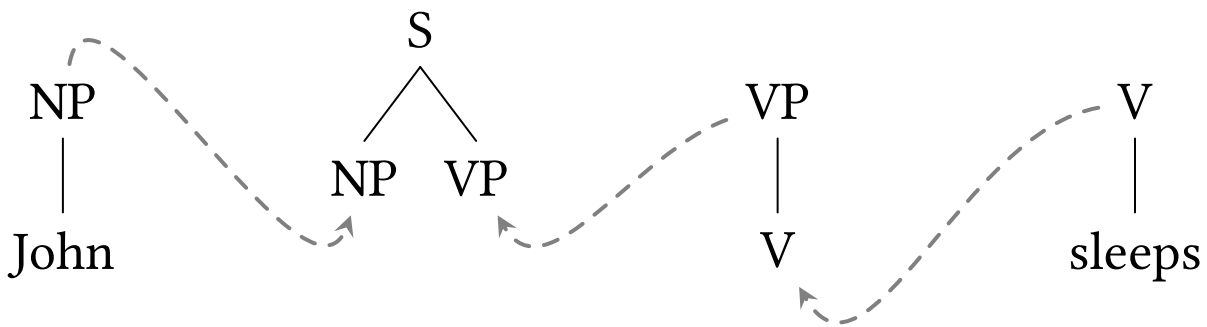
But: The copy language is not context-free.

Tree Substitution Grammar (TSG)

A tree rewriting version of CFG

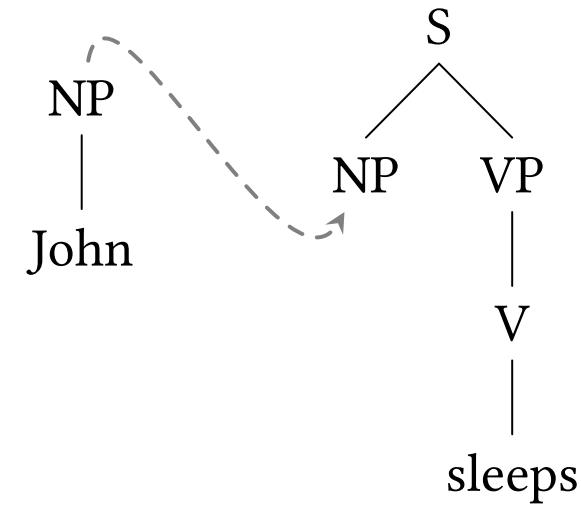
- A CFG-production corresponds to a TSG-tree with the LHS as root and the RHS as daughters.
- Applying a CFG-production corresponds to **substituting** a non-terminal leaf for a new tree.

$$S \rightarrow NP\ VP \qquad NP \rightarrow John$$
$$VP \rightarrow V \qquad V \rightarrow sleeps$$
$$\Rightarrow$$



Tree Substitution Grammar (TSG)

TSG-trees can be "higher" than CFG-productions:



$$\Rightarrow \quad \begin{aligned} S &\rightarrow NP\ VP \\ NP &\rightarrow \mathit{John} \\ VP &\rightarrow V \\ V &\rightarrow \mathit{sleeps} \end{aligned}$$

$\Rightarrow$ TSG comes with an **extended domain of locality**.

$\Rightarrow$ **But:** recursion cannot be factored away.

Tree Substitution Grammar (2)

A **Tree Substitution Grammar (TSG)** is a triple $G = \langle N, T, I \rangle$ such that

- T and N are disjoint alphabets, the terminals and nonterminals, and
- I is a finite set of **initial** trees.

The trees can be combined into larger trees by **substitution**.

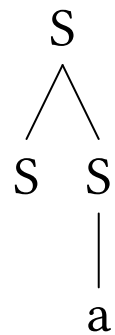
The **tree language** of a TSG is the set of trees generated in this way that do not contain any remaining non-terminal leaves.

Tree Substitution Grammar (3)

Some important facts:

- TSG is weakly equivalent to CFG (same string language).
- TSG is not powerful enough to describe cross-serial dependencies.
- It is not possible to find a strongly equivalent (same trees) lexicalized TSG for each CFG.

$S \rightarrow SS$
$S \rightarrow a$



$\Longrightarrow$ Solution: **adjunction operation** and **adjunction constraints**!

Tree Adjoining Grammar (TAG)

TAG = TSG + adjunction + adjunction constraints

- The definition of TAG goes back to (Joshi et al., 1975).
- TAG is among the most frequently used grammar formalisms in computational linguistics.
- TAG is interesting both for its computational properties (**mildly context-sensitivity**) and for its linguistic applications.
- There are large coverage TAG grammars for English (XTAG, Philadelphia) and French (FTAG, Paris).

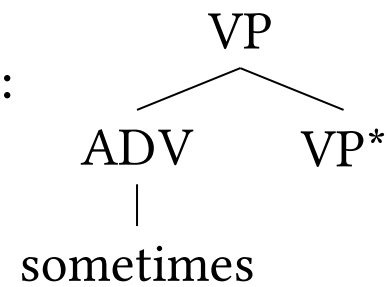
Tree Adjoining Grammar - Adjunction (1)

Rewriting operations:

- substitution: replacing a leaf with a new tree.
- adjunction: replacing an internal node with a new tree.

Trees that may adjoin are called auxiliary trees and have a special leaf, the footnode (marked by *). After adjuntion, the subtree below the target node appears below the footnode.

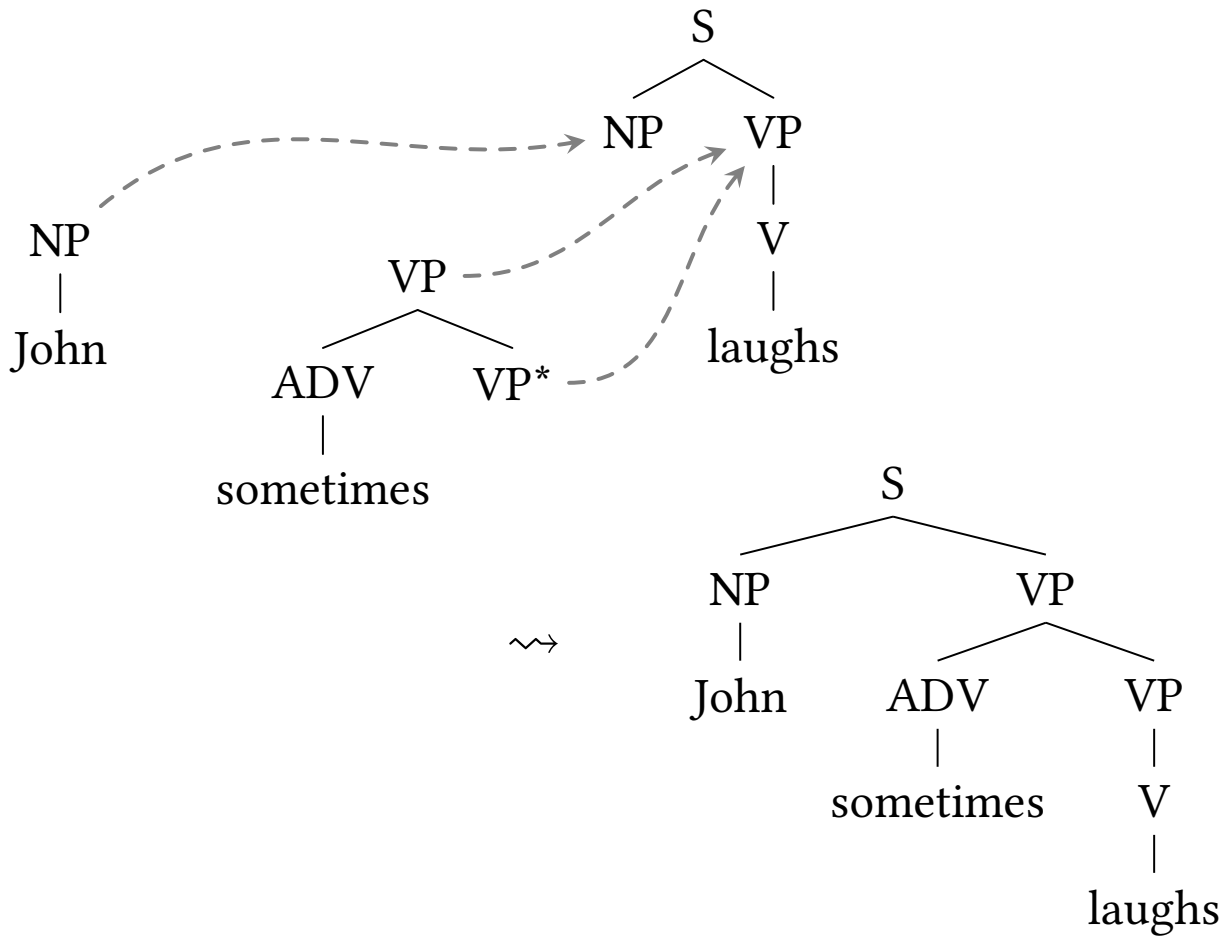
Example:



The root node and the footnode are required to carry the same label. The path from the root node to the footnode is called the **spine**.

Notation: $\gamma[p, \gamma']$ is the tree one obtains from replacing the node at position p in γ with the tree γ' (by substitution or adjunction).

Tree Adjoining Grammar - Adjunction (2)



⇒ TAG comes with an **extended domain of locality**.

⇒ **And:** recursion can be factored away by means of adjunction!

Tree Adjoining Grammar - Adjunction (3)

A **Tree Adjoining Grammar (TAG)** (Joshi & Schabes, 1997) is a quadruple $G = \langle N, T, I, A \rangle$ such that

- T and N are disjoint alphabets, the terminals and nonterminals,
- 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 (LTAG).

Tree Adjoining Grammar - Adjunction (4)

A derivation starts with an initial tree.
In a final derived tree, all leaves must have terminal labels:

Let $G = \langle I, A, N, T \rangle$ be a TAG. Let γ and γ' be finite trees.

- $\gamma \Rightarrow \gamma'$ in G iff there is a node position p and an instance γ'_0 of a tree (possibly derived from some) $\gamma_0 \in I \cup A$ such that $\gamma' = \gamma[p, \gamma_0]$.
  $\overset{*}{\Rightarrow}$ is the reflexive transitive closure of $\Rightarrow$.
- The tree language of G is $L_T(G) := \{\gamma \mid$ there is an $\alpha \in I$ such that $\alpha \overset{*}{\Rightarrow} \gamma$ and all leaves in γ have terminal labels$\}$.

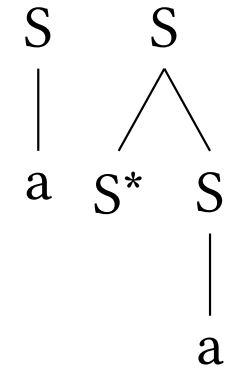
Tree Adjoining Grammar - Lexicalization challenge

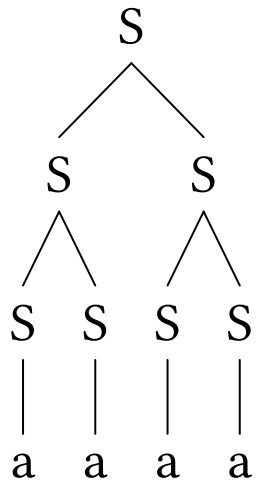
LTAGs strongly lexicalize (finitely ambiguous) CFGs, but not TAGs.

Example:

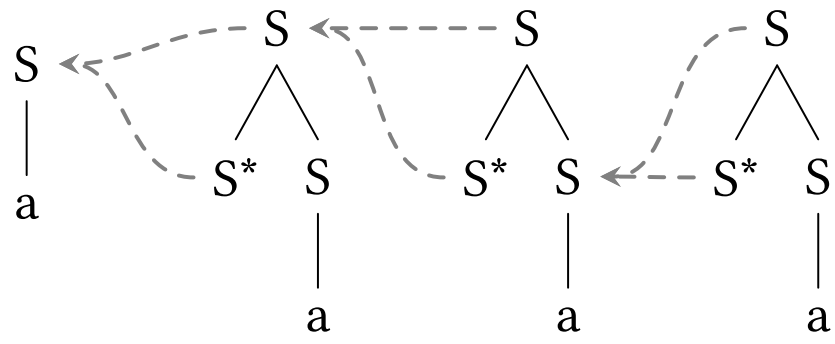
$S \to SS$
$S \to a$

is strongly equivalent to





$\Leftarrow$

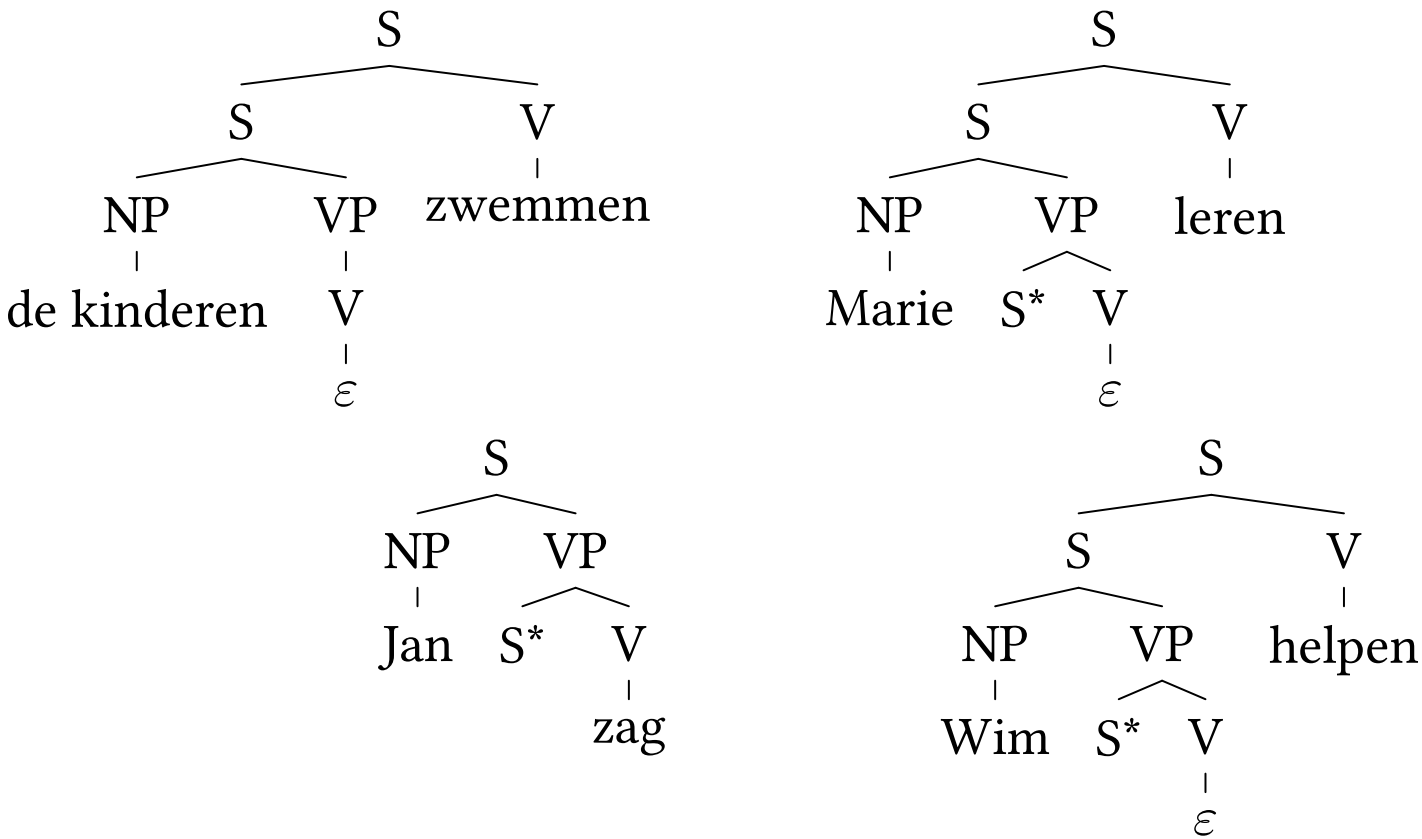


Tree Adjoining Grammar - Expressivity challenge

TAG can generate cross-serial dependencies in Dutch.

(3) …dat Jan Wim Marie de kinderen zag helpen leren zwemmen
…that Jan Wim Marie the children saw help teach swim
'…that Jan saw Wim help Mary teach the children to swim'

Tree Adjoining Grammar - Expressivity challenge



But: Also non-crossing dependencies are generated, since it's not possible to block adjunction at the root nodes!

Tree Adjoining Grammar - Adjunction constraints (1)

TAG 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.

Tree Adjoining Grammar - Adjunction constraints (2)

A **TAG with adjunction constraints** is a tuple $\langle N, T, I, A, O, C \rangle$ such that

- $\langle N, T, I, A \rangle$ is a TAG,
- $O : \{\mu \mid \mu \text{ is a node in a tree in } I \cup A\} \to \{1, 0\}$ is a function, and
- $C : \{\mu \mid \mu \text{ is a node in a tree in } I \cup A\} \to P(A)$ is a function.

Tree Adjoining Grammar - Adjunction constraints (3)

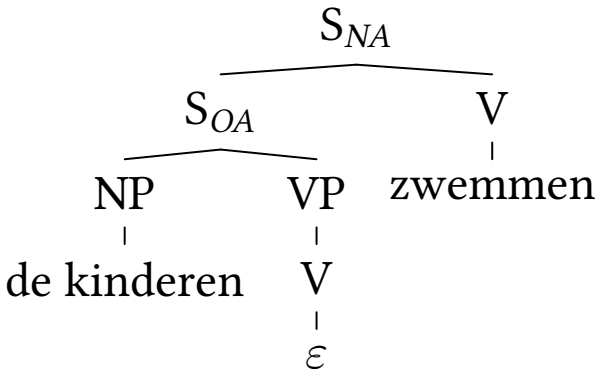
Three types of constraints are distinguished:

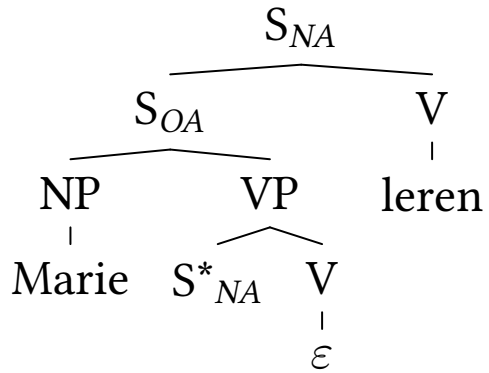
- **Obligatory Adjunction (OA):**
  a node μ with $O(\mu) = 1$
- **Null Adjunction (NA):**
  a node μ with $O(\mu) = 0$ and $C(\mu) = \emptyset$
- **Selective Adjunction (SA):**
  a node μ with $O(\mu) = 0$ and $C(\mu) \neq \emptyset$ and $C(\mu) \neq A$

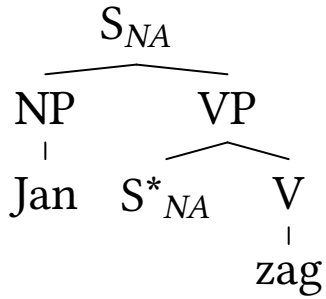
It is common practice to let the leaves carry the NA-constraint.

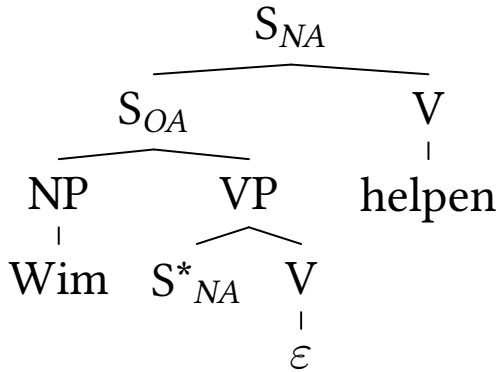
Tree Adjoining Grammar - Expressivity challenge

TAG with adjunction contraints for cross-serial dependencies in Dutch:



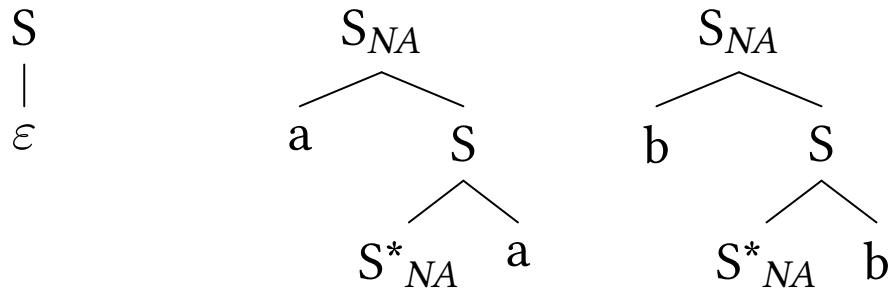






Tree Adjoining Grammar - Expressivity challenge

TAG with adjunction constraints for the copy language $\{ww | w \in \{a, b\}^*\}$:



Summary

Starting point: can we describe natural languages with CFGs?

- CFGs: string rewriting formalism, no strong lexicalization, no cross-serial dependencies.
- TSGs: tree rewriting formalism, no strong lexicalization, no cross-serial dependencies.
- TAG = TSG + adjunction + adjunction constraints
  - strong lexicalization (at least of CFGs)
  - cross-serial dependencies

References

Joshi, Aravind K., Leon S. Levy & Masako Takahashi. 1975. Tree Adjunct Grammars. Journal of Computer and System Science 10. 136–163.

Joshi, Aravind K. & Yves Schabes. 1997. Tree-Adjoning Grammars. In G. Rozenberg & A. Salomaa (eds.), Handbook of formal languages, 69–123. Berlin: Springer.

Savitch, Walter J., Emmon Bach, William Marxh & Gila Safran-Naveh (eds.). 1987. The formal complexity of natural language Studies in Linguistics and Philosophy. Dordrecht, Holland: Reidel.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. Linguistics and Philosophy 8. 333–343. Reprinted in Savitch et al. (1987).