Grammar Formalisms
Motivation for LTAG

Laura Kallmeyer, Timm Lichte, Wolfgang Maier
Universität Tübingen

25.04.2007

Outline

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

$S \rightarrow NP\ VP \quad NP \rightarrow John \quad NP \rightarrow Mary$
$VP \rightarrow V \quad VP \rightarrow V\ NP \quad V \rightarrow sleeps \quad V \rightarrow likes$

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

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

$NP_{3sg-nom} \rightarrow John$ $\quad$ $NP_{3sg-acc} \rightarrow Mary$
$V_{3sg-itr} \rightarrow sleeps$ $\quad$ $V_{3sg-tr} \rightarrow likes$
$S \rightarrow NP_{3sg-nom}\ VP_{3sg-itr}$ $\quad$ $S \rightarrow NP_{3sg-nom}\ VP_{3sg-tr}$
$VP_{3sg-itr} \rightarrow V_{3sg-itr}$ $\quad$ $VP_{3sg-tr} \rightarrow V_{3sg-tr}\ NP_{3sg-acc}$

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

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!

$\Longrightarrow$ 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: $S \rightarrow aS$, $S \rightarrow a$

- **Computationally interesting:** the number of analyses for a sentence is finite (if the grammar is finite of course).
- **Linguistically interesting:** each lexical item comes with the possibility of certain partial syntactic constructions, therefore one would like to associate it to a set of substructures.

Why CFG is not enough (2)

Lexicalizing a CFG:

- Greibach normal form: $A \rightarrow aB_1...B_k$ $(k \geq 0)$
- 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: $S \rightarrow aS$, $S \rightarrow a$

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

(1)

... dat Wim Jan Marie de kinderen zag helpen leren zwemmen
... that Wim Jan Marie the children saw help teach swim
'... that Wim saw Jan help Marie teach the children to swim'

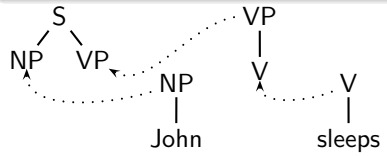
A formalism that can generate cross-serial dependencies must be able to generate the copy language $\{ww \mid w \in \{a, b\}^*\}$.
The copy language is not context-free.

Tree Substitution Grammar (TSG)

- a tree rewriting version of CFG
- weakly equivalent generative capacity

$S \rightarrow NP\ VP \quad NP \rightarrow John$
$VP \rightarrow V \quad V \rightarrow sleeps$
Possible derivation:
$S \Rightarrow NP\ VP \Rightarrow John\ VP \Rightarrow John\ V \Rightarrow John\ sleeps$

- A production means a tree with the LHS as root and the RHS as daughters.
- Applying a production means substituting a non-terminal leaf for a new tree.



Tree Substitution Grammar (2)

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

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

The trees can be combined into larger trees by substitution.

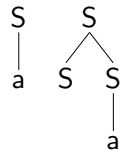
The **tree language** of a TSG is the set of trees generated in this way that do not contain any remaining non-terminal leaves.

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

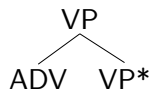
- The definition of TAG goes back to Joshi et al. (1975).
- TAG is among the most frequently used grammar formalisms in computational linguistics.
- TAG is interesting both for its computational properties (**mildly context-sensitivity**) and for its linguistic applications.
- There are large coverage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

Tree Adjoining Grammar - Adjunction (1)

Tree Adjoining Grammars (TAG): Attachment operations:

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

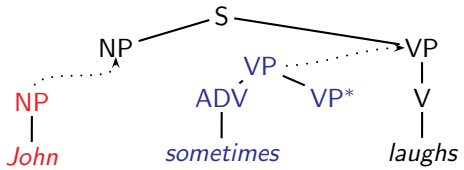
Trees that may adjoin are called auxiliary trees and have a special leaf, the footnode (marked by *).



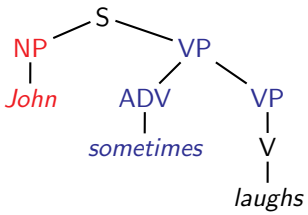
Notation: $\gamma[p, \gamma']$ is the tree one obtains from replacing the node at position p in γ with the tree γ' (by substitution or adjunction).

Tree Adjoining Grammar - Adjunction (2)

(2) John sometimes laughs



derived tree
$laugh[1, john][2, sometimes]$:



Tree Adjoining Grammar - Adjunction (3)

A **Tree Adjoining Grammar (TAG)** 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)

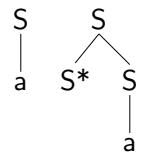
A derivation starts with an initial tree. In a final derived tree, all leaves must have terminal labels:
Let $G = \langle I, A, N, T \rangle$ be a TAG. Let γ and γ' be finite trees.

- $\gamma \Rightarrow \gamma'$ in G iff there is a node position p and an instance ${\gamma_0}'$ 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 CFGs and TAGs.

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



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,
- $O : \{\mu \mid \mu$ is an internal node or a foot node in a tree in $I \cup A\} \rightarrow \{1, 0\}$ is a function, and
- $C : \{\mu \mid \mu$ is an internal node or a foot node in a tree in $I \cup A\} \rightarrow P(A)$ is a function.

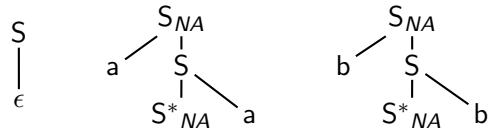
Tree Adjoining Grammar - Adjunction constraints (3)

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 $C(\mu) \neq A$

Tree Adjoining Grammar - Expressivity challenge

TAG for the copy language $\{ww \mid w \in \{a, b\}^*\}$:



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
  - cross-serial dependencies