Parsing beyond context-free grammar: Tree Adjoining Grammar Parsing I

Laura Kallmeyer, Wolfgang Maier

Sommersemester 2009

Overview

1. Adjunction and substitution
2. Adjunction constraints
3. TAG derivation trees
4. Formal properties
5. Elementary trees for natural languages
6. A CYK recognizer for TAG

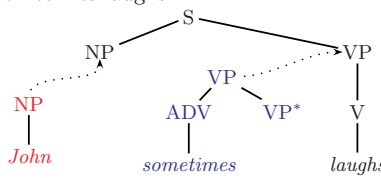
Adjunction and substitution (1)

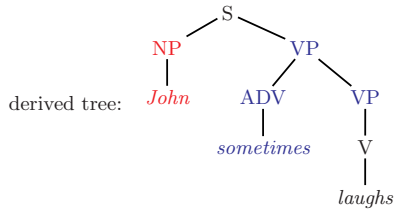
Tree Adjoining Grammars (TAG) Joshi et al. (1975), Joshi & Schabes (1997): Tree-rewriting system: set of *elementary* trees with two operations:

- 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 tree. The new tree is an initial tree

Adjunction and substitution (2)

(1) John sometimes laughs





Adjunction and substitution (3)

Definition 1 (Tree Adjoining Grammar) *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.

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.

Adjunction constraints (2)

Definition 2 (TAG with adjunction constraints) *A TAG with adjunction constraints is a tuple $\langle N, T, I, A, f_{OA}, f_{SA}\rangle$ such that*

- *$\langle N, T, I, A\rangle$ is a TAG,*
- *$f_{OA} : \{\mu \,|\, \mu$ is an internal node or a foot node in a tree in $I \cup A\} \rightarrow \{1, 0\}$ is a function, and*
- *$f_{SA} : \{\mu \,|\, \mu$ is an internal node or a foot node in a tree in $I \cup A\} \rightarrow P(A)$ is a function.*

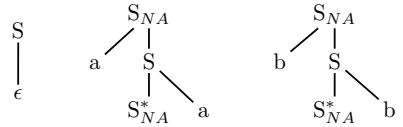
Adjunction constraints (3)

Three types of constraints are distinguished:

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

Adjunction constraints (4)

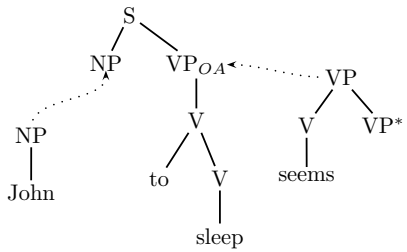
Example: TAG for the copy language:



Adjunction constraints (4)

Example:

(2) John seems to sleep



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

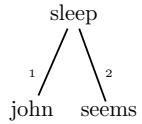
- nodes for all elementary trees used in the derivation, and
- edges for all adjunctions and substitutions performed throughout the derivation.

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)

Example:

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



Some 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 generated 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.

Some formal properties (2)

TAGs are mildly context-sensitive:

- TAGs are slightly more powerful than CFG, they can describe a limited amount of cross-serial dependencies.
- TAGs are polynomially parsable (complexity $O(n^6)$).
- TALs are of constant growth.

Elementary trees (1)

Important features of LTAG:

- Grammar is lexicalized
- Recursive parts are put into separate elementary trees that can be adjoined (Factoring of recursion, FR)
- Elementary trees can be arbitrarily large, in particular (because of FR) they can contain elements that are far apart in the final derived tree (Extended domain of locality)

Elementary trees (2)

Elementary trees are extended projections of lexical items.
Recursion is factored away $\Rightarrow$ finite set of elementary trees.

The elementary tree of a lexical predicate contains slots for all arguments of the predicate, for nothing more.

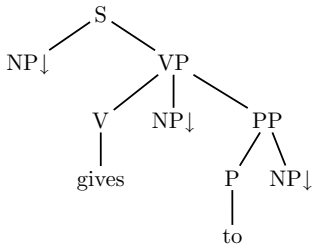
Besides lexical predicates, there are functional elements (complementizers, determiners, auxiliaries, negation) whose treatment in LTAG is less clear. They can be

- either in separate elementary trees (e.g., XTAG grammar)
- or in the elementary tree of the lexical item they are associated with.

Elementary trees (3)

Example:

(3) John gives a book to Mary

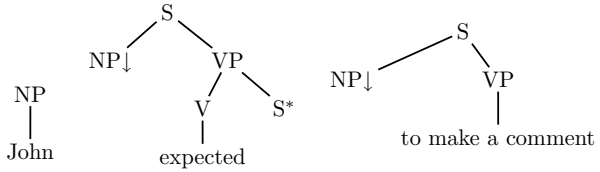


Elementary trees (4)

Example:

(4) John expected Mary to make a comment

expected selects for a subject NP and an infinitival sentence:

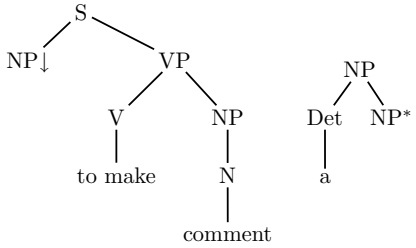


The sentential object is realised as a foot node in order to allow extractions:

(5) whom does John expect to come?

Elementary trees (5)

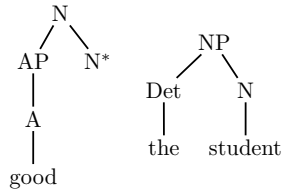
to make a comment: *make* and *comment* in the same elementary tree since they form a light verb construction:



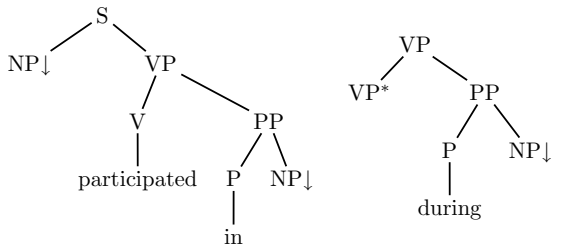
Elementary trees (6)

Example with modifiers:

(6) the good student participated in every course during the semester



Elementary trees (7)



CYK Recognition (1)

- First presented in Vijay-Shanker & Joshi (1985), formulation with deduction rules in Kallmeyer & Satta (2009).
- Assumption: elementary trees are such that each node has at most two daughters. (Any TAG can be transformed into an equivalent TAG satisfying this condition.)
- The algorithm simulates a bottom-up traversal of the derived tree.

CYK Recognition (2)

- At each moment, we are in a specific node in an elementary tree and we know about the yield of the part below. Either there is a foot node below, then the yield is separated into two parts. Or there is no foot node below and the yield is a single substring of the input.
- We need to keep track of whether we have already adjoined at the node or not since at most one adjunction per node can occur. For this, we distinguish between a bottom and a top position for the dot on a node. Bottom signifies that we have not performed an adjunction.

CYK Recognition (3)

Item form: $[\gamma, p_t, i, f_1, f_2, j]$ where

- $\gamma \in I \cup A$,
- p is the Gorn address of a node in γ (ϵ for the root, pi for the ith daughter of the node at address p),
- subscript $t \in \{\top, \bot\}$ specifies whether substitution or adjunction has already taken place ($\top$) or not ($\bot$) at p, and
- $0 \leq i \leq f_1 \leq f_2 \leq j \leq n$ are indices with i, j indicating the left and right boundaries of the yield of the subtree at position p and f_1, f_2 indicating the yield of a gap in case a foot node is dominated by p. We write $f_1 = f_2 = -$ if no gap is involved.

CYK Recognition (4)

Goal items: $[\alpha, \epsilon_\top, 0, -, -, n]$ where $\alpha \in I$

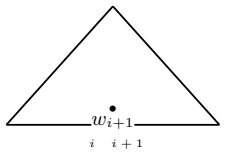
We need two rules to process leaf nodes while scanning their labels, depending on whether they have terminal labels or labels ϵ:

Lex-scan: $\dfrac{}{[\gamma, p_\top, i, -, -, i+1]}$ $l(\gamma, p) = w_{i+1}$

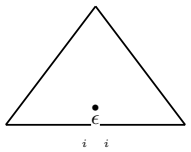
Eps-scan: $\dfrac{}{[\gamma, p_\top, i, -, -, i]}$ $l(\gamma, p) = \epsilon$

(Notation: $l(\gamma, p)$ is the label of the node at address p in γ.)

CYK Recognition (5)



Lex-scan



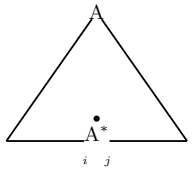
Eps-scan

CYK Recognition (6)

The rule **foot-predict** processes the foot node of auxiliary trees $\beta \in A$ by guessing the yield below the foot node:

Foot-predict:

$\dfrac{}{[\beta, p_\top, i, i, j, j]}$ $\beta \in A, p$ foot node address in $\beta, i \leq j$



CYK Recognition (7)

When moving up inside a single elementary tree, we either move from only one daughter to its mother, if this is the only daughter, or we move from the set of both daughters to the mother node:

Move-unary:

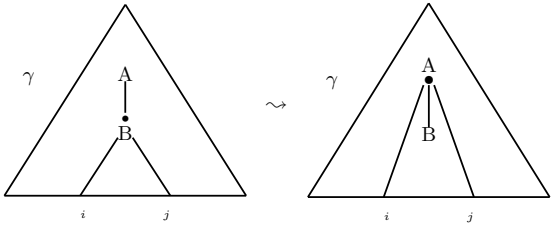
$\dfrac{[\gamma, (p \cdot 1)_\top, i, f_1, f_2, j]}{[\gamma, p_\bot, i, f_1, f_2, j]}$ node address $p \cdot 2$ does not exist in γ

Move-binary: $\dfrac{[\gamma, (p \cdot 1)_\top, i, f_1, f_2, k], [\gamma, (p \cdot 2)_\top, k, f_1', f_2', j]}{[\gamma, p_\bot, i, f_1 \oplus f_1', f_2 \oplus f_2', j]}$

($f' \oplus f'' = f$ where $f = f'$ if $f'' = -$, $f = f''$ if $f' = -$, and f is undefined otherwise)

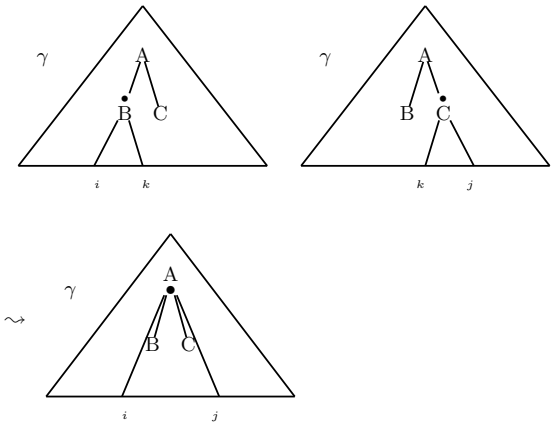
CYK Recognition (8)

Move-unary:



CYK Recognition (9)

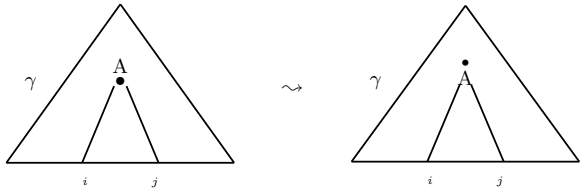
Move-binary:



CYK Recognition (10)

For nodes that do not require adjunction, we can move from the bottom position of the node to its top position.

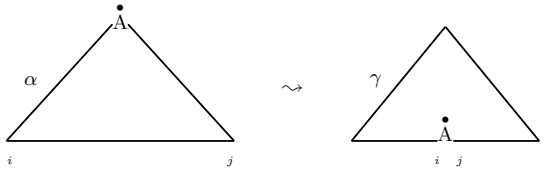
Null-adjoin: $\dfrac{[\gamma, p_\bot, i, f_1, f_2, j]}{[\gamma, p_\top, i, f_1, f_2, j]}$ $f_{OA}(\gamma, p) = 0$



CYK Recognition (11)

The rule **substitute** performes a substitution:

Substitute: $\dfrac{[\alpha, \epsilon_\top, i, -, -, j]}{[\gamma, p_\top, i, -, -, j]}$ $l(\alpha, \epsilon) = l(\gamma, p)$



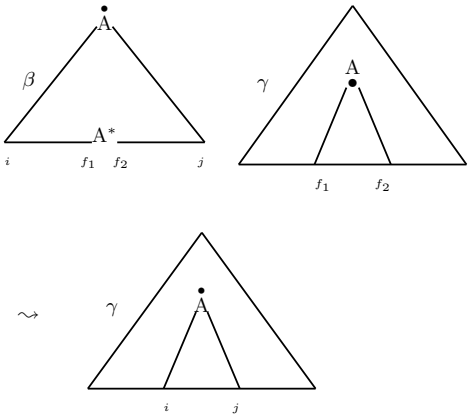
CYK Recognition (12)

The rule **adjoin** adjoins an auxiliary tree β at p in γ, under the precondition that the adjunction of β at p in γ is allowed:

Adjoin: $\dfrac{[\beta, \epsilon_\top, i, f_1, f_2, j], [\gamma, p_\bot, f_1, f_1', f_2', f_2]}{[\gamma, p_\top, i, f_1', f_2', j]}$ $\beta \in f_{SA}(\gamma, p)$

CYK Recognition (13)

Adjoin:



CYK Recognition (14)

Complexity of the algorithm: What is the upper bound for the number of applications of the **adjoin** operation?

- We have $|A|$ possibilities for β, $|A \cup I|$ for γ, m for p where m is the maximal number of internal nodes in an elementary tree.
- The six indices $i, f_1, f_1', f_2', f_2, j$ range from 0 to n.

Consequently, **adjoin** can be applied at most $|A||A \cup I|m(n+1)^6$ times and therefore, the time complexity of this algorithm is $\mathcal{O}(n^6)$.