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, 4th session
Sep 1, 2011

Outline

1. Parsing as deduction
2. A CYK recognizer for TAG
   - Items
   - Inference rules
   - Complexity

Deduction systems (1)

- Algorithmic descriptions of parsing algorithms (e.g., in pseudo-code) introduce *data structures* and *control structures*.
- The parsing strategy of the algorithm does not depend on them.

Question: Can we separate the parsing strategy from the control strategy?
Answer: Deduction systems [Shieber et al., 1995, Sikkel, 1997]

Deduction systems (2)

Advantages:

- Concentration on parsing strategy;
- Facilitation of proofs (e.g., soundness and completeness of an algorithm).
  Soundness: if algo yields *true* for w, then $w \in L(G)$.
  Completeness: if $w \in L(G)$, then algo yields *true* for w.
- Complexity of an algorithm sometimes easier to determine.
- Facilitates tabulation and an implementation as a chart parser.

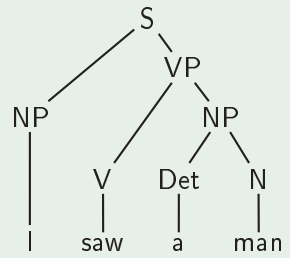
Deduction systems (3)

How characterize a single parsing step?
During parsing, the parser produces trees (*parse trees*, partial results) and tries to combine them to new trees, until some tree rooted by the goal category (e.g. S) comes out.

- We can characterize parse trees
- We can characterize how new parse trees can be deduced from existing ones
- We can fix a goal: We want to deduce a tree with root S that spans the entire input sentence

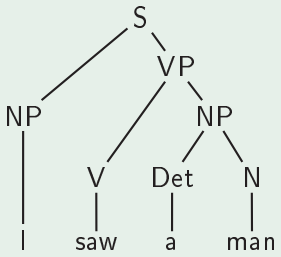
Deduction systems (4)

We characterize a parse tree rooted by some nonterminal X by the terminals X spans.



Deduction systems (4)

We characterize a parse tree rooted by some nonterminal X by the terminals X spans.



We write parse trees/partial parse results in the form of *items*, e.g., $[X, i, j]$, meaning that X derives the terminals between i and j.

Here: $[S, 0, 4]$

Deduction systems (6)

- Parsing can be viewed as a deductive process
- Deduction of new items from existing ones can be described using inference rules.

Deduction systems (6)

- Parsing can be viewed as a deductive process
- Deduction of new items from existing ones can be described using inference rules.

General form

$$\frac{antecedent}{consequent}\ side\ conditions$$

(antecedent, consequent: lists of items)

Deduction systems (6)

- Parsing can be viewed as a deductive process
- Deduction of new items from existing ones can be described using inference rules.

General form

$$\frac{antecedent}{consequent}\ side\ conditions$$

(antecedent, consequent: lists of items)

Application: if antecedent can be deduced and side condition holds, then the consequent can be deduced as well.

Deduction systems (7)

A deduction system consists of

Deduction systems (7)

A deduction system consists of

- deduction rules;

Deduction systems (7)

A deduction system consists of

- deduction rules;
- an axiom (or axioms), can be written as a deduction rule with empty antecedent;

Deduction systems (7)

A deduction system consists of

- deduction rules;
- an axiom (or axioms), can be written as a deduction rule with empty antecedent;
- and a goal item.

Deduction systems (7)

A deduction system consists of

- deduction rules;
- an axiom (or axioms), can be written as a deduction rule with empty antecedent;
- and a goal item.

The parsing algorithm succeeds if, for a given input, it is possible to deduce the goal item.

Deduction systems (8)

Deduction system for CYK (CFG-Parsing)

Items have three elements:

- $X \in N \cup T$: the nonterminal/terminal that spans a substring $w_i, \ldots, w_j$ of w;
- the index i of the first terminal in the subsequence;
- the length $l = j - i$ of the subsequence.

Item form: $[X, i, l]$ with $X \in N \cup T$, $i \in [1..n+1]$, $l \in [0..n]$.

Deduction systems (9)

Goal item: $[S, 1, n]$.
Deduction rules:

Scan: $\dfrac{}{[a, i, 1]}$ $w_i = a$

ε-productions: $\dfrac{}{[A, i, 0]}$ $A \to \varepsilon \in P, i \in [1..n+1]$

Complete: $\dfrac{[A_1, i_1, l_1], \ldots, [A_k, i_k, l_k]}{[A, i_1, l]}$ $A \to A_1 \ldots A_k \in P$, $l = l_1 + \cdots + l_k$, $i_j = i_1 + l_1 \cdots + l_{j-1}$ for $1 < j \leq k$

CYK: Items (1)

CYK-Parsing for TAG:

- First presented in [Vijay-Shanker and Joshi, 1985], formulation with deduction rules in [Kallmeyer and 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: Items (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: Items (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.

Natural Language Syntax with TAG 12/25

CYK: Items (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: Items (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.

Natural Language Syntax with TAG 13/25

CYK: Items (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.

Natural Language Syntax with TAG 13/25

CYK: Items (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.

Natural Language Syntax with TAG 13/25

CYK: Inference rules (1)

Goal items

$[\alpha, \varepsilon_\top, 0, -, -, n]$ 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

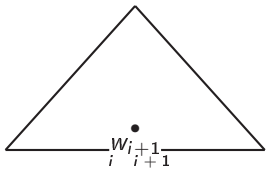
$$\frac{}{[\gamma, p_\top, i, -, -, i+1]} \; l(\gamma, p) = w_{i+1}$$

Eps-scan

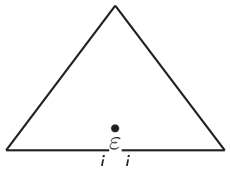
$$\frac{}{[\gamma, p_\top, i, -, -, i]} \; l(\gamma, p) = \varepsilon$$

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

CYK: Inference rules (2)



Lex-scan



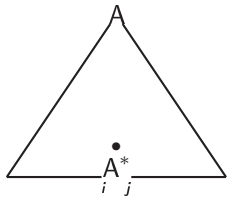
Eps-scan

CYK: Inference rules (3)

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

Foot-predict

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



CYK: Inference rules (4)

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

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

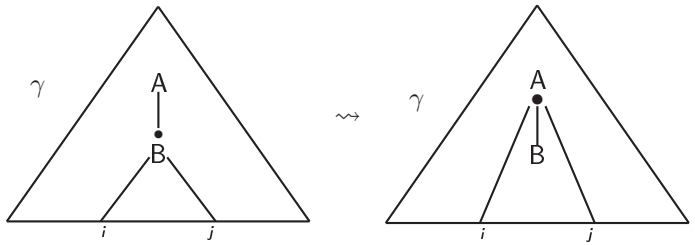
Move-binary

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

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

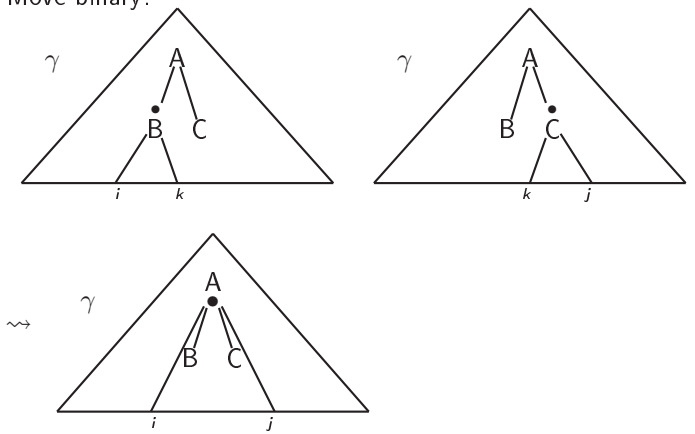
CYK: Inference rules (5)

Move-unary:



CYK: Inference rules (6)

Move-binary:

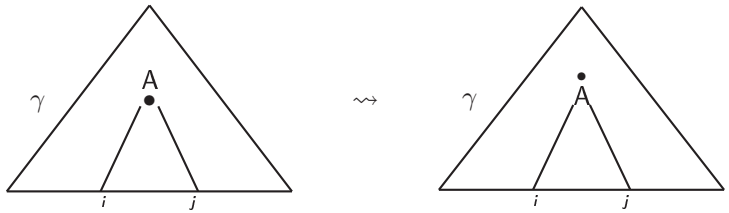


CYK: Inference rules (7)

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

Null-adjoin

$$\frac{[\gamma, p_\perp, i, f_1, f_2, j]}{[\gamma, p_\top, i, f_1, f_2, j]} \quad f_{OA}(\gamma, p) = 0$$

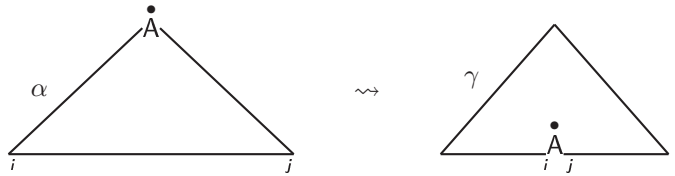


CYK: Inference rules (8)

The rule substitute performes a substitution:

Substitute

$$\frac{[\alpha, \varepsilon_\top, i, -, -, j]}{[\gamma, p_\top, i, -, -, j]} \quad l(\alpha, \varepsilon) = l(\gamma, p)$$



CYK: Inference rules (9)

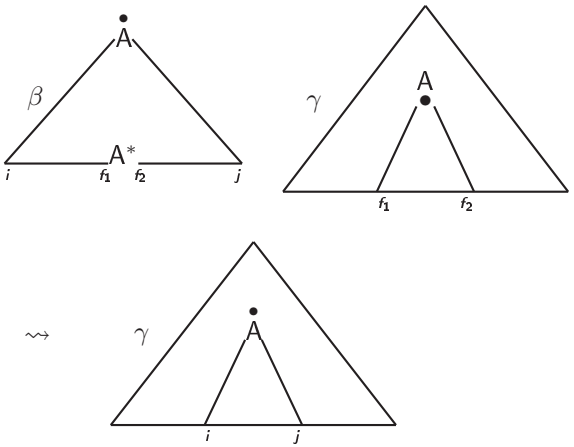
The rule adjoin adjoins an auxiliary tree β at p in γ, under the precondition that the adjunction of β at p in γ is allowed:

Adjoin

$$\frac{[\beta, \varepsilon_\top, i, f_1, f_2, j], [\gamma, p_\bot, f_1, f_1', f_2', f_2]}{[\gamma, p_\top, i, f_1', f_2', j]} \quad \beta \in f_{SA}(\gamma, p)$$

CYK: Inference rules (10)

Adjoin:



CYK: Complexity

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 $\mathcal{O}(n^6)$.

CYK: Complexity

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 $\mathcal{O}(n^6)$.


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


Kallmeyer, L. and Satta, G. (2009).
A Polynomial-Time Parsing Algorithm for TT-MCTAG.
In Proceedings of ACL, Singapore.


Nederhof, M.-J. (1997).
Solving the correct-prefix property for TAGs.
In Becker, T. and Krieger, H.-U., editors, Proceedings of the Fifth Meeting on Mathematics of Language, pages 124–130, Schloss Dagstuhl, Saarbrücken.


Schabes, Y. and Joshi, A. K. (1988).
An Earley-type parsing algorithm for Tree Adjoining Grammars.
In Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics, pages 258–269.


Shieber, S. M., Schabes, Y., and Pereira, F. C. N. (1995).
Principles and implementation of deductive parsing.
Journal of Logic Programming, 24(1&2):3–36.


Sikkel, K. (1997).
Parsing Schemata.
Springer, Berlin, Heidelberg, New York.


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.