Tree Adjoining Grammars

TAG: Parsing and formal properties

Laura Kallmeyer & Benjamin Burkhardt

HHU Düsseldorf

WS 2017/2018

Outline

Parsing as deduction: Parsing Schemata (1)

Pereira and Warren (1983); Shieber et al. (1995); Sikkel (1997)
Parsing Schemata understand parsing as a deductive process.
Deduction of new items from existing ones can be described using inference rules.
General form:

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

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

Parsing as deduction: Parsing Schemata (2)

A parsing schema 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.

Parsing as deduction: Parsing schemata (3)

Example: CYK-Parsing for CFG in Chomsky Normal Form.
Goal item: $[S, 1, n]$
Deduction rules:

Scan: $$\frac{}{[A, i, 1]} \; A \rightarrow w_i \in P$$

Complete: $$\frac{[B, i, l_1], [C, i + l_1, l_2]}{[A, i, l_1 + l_2]} \; A \rightarrow B\,C \in P$$

Parsing as deduction: Chart parsing (1)

Chart parsing:
We have two structures,

- the chart $\mathcal{C}$
- and an agenda $\mathcal{A}$.

Both are initialized as empty.

- We start by computing all items that are axioms, i.e., that can be obtained by applying rules with empty antecedents.
- Starting from these items, we extend the set $\mathcal{C}$ as far as possible by subsequent applications of the deduction rules.
- The agenda contains items that are waiting to be used in further deduction rules. It avoids multiple applications of the same instance of a deduction rule.

Parsing as deduction: Chart parsing (2)

Chart parsing

```
$\mathcal{C} = \mathcal{A} = \emptyset$
for all items $I$ resulting form a rule
application with empty antecedent set:
    add $I$ to $\mathcal{C}$ and to $\mathcal{A}$
while $\mathcal{A} \neq \emptyset$:
    remove an item $I$ from $\mathcal{A}$
    for all items $I'$ deduced from $I$ and items
    from $\mathcal{C}$ as antecedents:
        if $I' \notin \mathcal{C}$:
            add $I'$ to $\mathcal{C}$ and to $\mathcal{A}$
if there is a goal item in $\mathcal{C}$:
    return true
else return false
```

CYK for TAG: Items (1)

CYK-Parsing for TAG:

- First presented in Vijay-Shanker and Joshi (1985), formulation with deduction rules in Kallmeyer and Satta (2009); Kallmeyer (2010).
- 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 for TAG: 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 for TAG: 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.

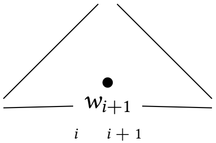
CYK for TAG: Inference rules (1)

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

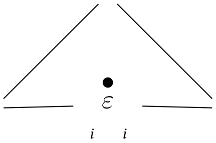
We need two rules to process leaf nodes while scanning their labels, depending on whether they have terminal labels or labels ϵ:

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

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



Lex-scan



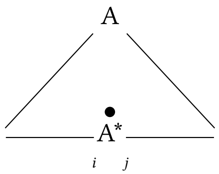
Eps-scan

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

CYK for TAG: Inference rules (2)

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 β, $i \leq j$



CYK for TAG: Inference rules (3)

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:

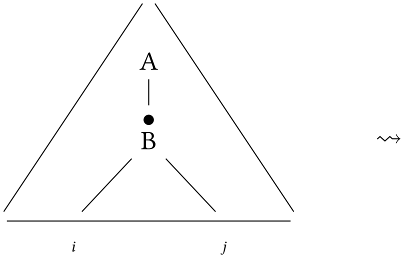
Move-unary: $\dfrac{[\gamma, (p \cdot 1)_\top, i, f_1, f_2, j]}{[\gamma, p_\bot, i, f_1, f_2, j]}$ node address $p \cdot 2$ does not exist in γ

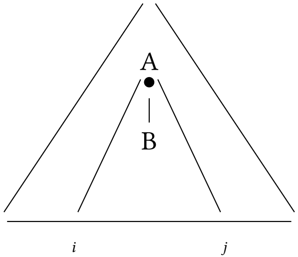
Move-binary: $\dfrac{[\gamma, (p \cdot 1)_\top, i, f_1, f_2, k], [\gamma, (p \cdot 2)_\top, k, f_1', f_2', j]}{[\gamma, p_\bot, i, f_1 \oplus f_1', f_2 \oplus f_2', j]}$

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

CYK for TAG: Inference rules (4)

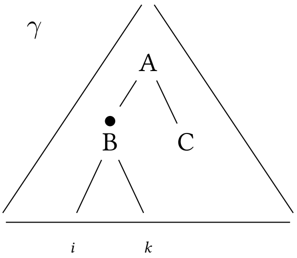
Move-unary:

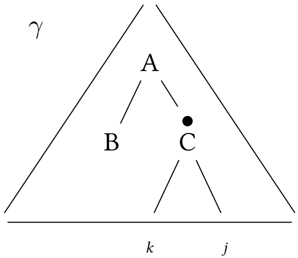




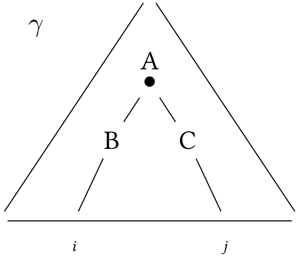
CYK for TAG: Inference rules (5)

Move-binary:





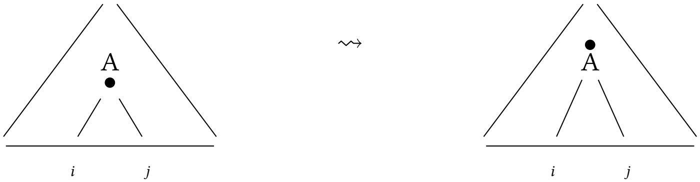
$\rightsquigarrow$



CYK for TAG: Inference rules (6)

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

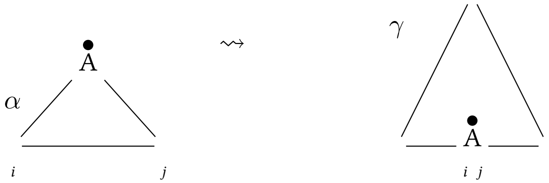
Null-adjoin: $\dfrac{[\gamma, p_\perp, i, f_1, f_2, j]}{[\gamma, p_\top, i, f_1, f_2, j]}\ f_{OA}(\gamma, p) = 0$



CYK for TAG: Inference rules (9)

The rule **substitute** performes a substitution:

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



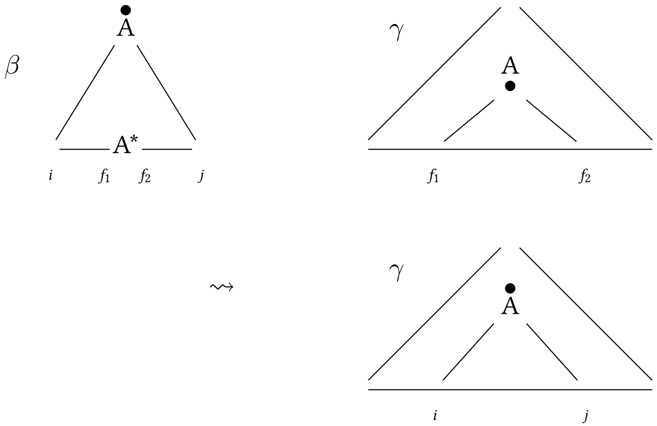
CYK for TAG: Inference rules (8)

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

CYK for TAG: Inference rules (9)

Adjoin:



CYK for TAG: 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 $\mathcal{O}(n^6)$.

Closure properties (1)

One of the reasons why the TAG formalism is appealing from a formal point of view is the fact that it has nice closure properties Vijay-Shanker and Joshi (1985); Vijay-Shanker (1987).

Proposition

TALs are closed under union.

This can be easily shown as follows: Assume the two sets of non-terminals to be disjoint. Then build a large TAG putting the initial and auxiliary trees from the two grammars together.

Closure properties (2)

Proposition

TALs are closed under concatenation.

In order to show this, assume again the sets of non-terminals to be disjoint. Then

- build the unions of the initial and auxiliary trees,
- introduce a new start symbol S and
- add one initial tree with root label S and two daughters labeled with the start symbols of the original grammars.

Closure properties (3)

Proposition

TALs are closed under Kleene closure.

The idea of the proof is as follows: We add an initial tree with the empty word and an auxiliary tree that can be adjoined to the roots of initial trees with the start symbol and that has a new leaf with the start symbol.

Closure properties (4)

Proposition

TALs are closed under substitution.

In order to obtain the TAG that yields the language after substitution, we replace all terminals by start symbols of the corresponding TAGs.
As a corollary one obtains:

Proposition

TALs are closed under arbitrary homomorphisms.

Closure properties (5)

Proposition

TALs are closed under intersection with regular languages.

The proof in Vijay-Shanker (1987) uses extended push-down automata (EPDA), the automata that recognize TALs. We will introduce EPDAs later. Vijay-Shanker combines such an automaton with the finite state automaton for a regular language in order to construct a new EPDA that recognizes the intersection.

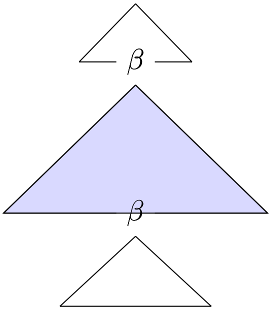
Pumping lemma (1)

- In CFLs, from a certain string length on, two parts of the string can be iterated ("pumped").
- The proof idea is the following: Context-free derivation trees from a certain maximal path length on have the property that a non-terminal occurs twice on this path. Then the part between the two occurrences can be iterated. This means that the strings to the left and right of this part are pumped.

The same kind of iteration is possible in TAG derivation trees since TAG derivation trees are context-free. This leads to a pumping lemma for TALs Vijay-Shanker (1987).

Pumping lemma (2)

Iteration on TAG derivation trees:



The blue part can be iterated.

Pumping lemma (3)

In other words,

- A derived auxiliary tree β' can be repeatedly adjoined into itself.
- Into the lowest β' (low in the sense of the derivation tree) another auxiliary tree β'' derived from β is adjoined.

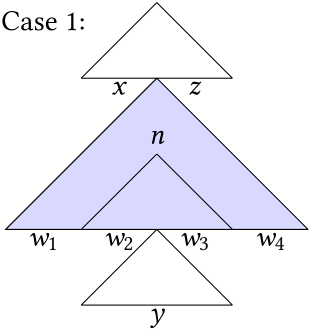
Pumping lemma (4)

What does that mean for the derived tree?
Let n be the node in β' to which β' can be adjoined and to which the final β'' is adjoined as well. There are three cases for the corresponding derived trees before adjoining the final β'':

1. n is on the spine (i.e., on the path from the root to the foot node),
2. n is on the left of the spine, or
3. n is on the right of the spine.

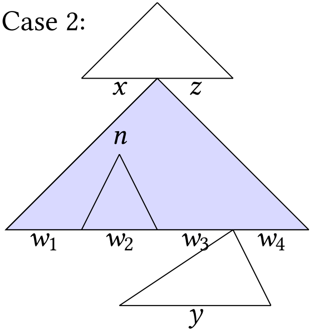
Pumping lemma (5)

Case 1:



$\rightsquigarrow xw_1^n v_1 w_2^n y w_3^n v_2 w_4^n z$

Case 2:



$\rightsquigarrow xw_1^{n+1} v_1 w_2 v_2 w_3 (w_2 w_4 w_3)^n y w_4 z$

(Case 3 is exactly like case 2, except that everything is mirrored.)

Pumping lemma (6)

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 $x, y, z, v_1, v_2, w_1, w_2, w_3, w_4 \in T^*$ such that

- $|v_1 v_2 w_1 w_2 w_3 w_4| \leq c$, $|w_1 w_2 w_3 w_4| \geq 1$, *and*
- *one of the following three cases holds:*
  1. $w = x w_1 v_1 w_2 y w_3 v_2 w_4 z$ *and* $x w_1^n v_1 w_2^n y w_3^n v_2 w_4^n z$ *is in the string language for all* $n \geq 0$, *or*
  2. $w = x w_1 v_1 w_2 v_2 w_3 y w_4 z$ *and* $x w_1^{n+1} v_1 w_2 v_2 w_3 (w_2 w_4 w_3)^n y w_4 z$ *is in the string language for all* $n \geq 0$, *or*
  3. $w = x w_1 y w_2 v_1 w_3 v_2 w_4 z$ *and* $x w_1 y (w_2 w_1 w_3)^n w_2 v_1 w_3 v_2 w_4^{n+1} z$ *is in the string language for all* $n \geq 0$.

Pumping lemma (7)

As a corollary, the following weaker pumping lemma holds:

Proposition (Weak 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 $x, y, z, v_1, v_2, w_1, w_2, w_3, w_4 \in T^*$ such that

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

In this weaker version, the w_1, w_2, w_3, w_4 need not be substrings of the original word w.

Pumping lemma (8)

A pumping lemma can be used to show that certain languages are not in the class of the string languages satisfying the pumping proposition.

Proposition

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

Pumping lemma (9)

Proof: Assume that L is a TAL.
Then $L' := L \cap a^* b^* a^* b^* a^* b^* = \{a^n b^m a^n b^m a^n b^m \mid n, m \geq 0\}$ is a TAL as well. Assume that L' satisfies the weak pumping lemma with a constant c.
Consider the word $w = a^{c+1} b^{c+1} a^{c+1} b^{c+1} a^{c+1} b^{c+1}$.
None of the w_i, $1 \leq i \leq 4$ from the pumping lemma can contain both as and bs. Furthermore, at least three of them must contain the same letters and be inserted into the three different a^{c+1} respectively or into the three different b^{c+1}. This is a contradiction since then either $|v_1| \geq c + 1$ or $|v_2| \geq c + 1$.

Pumping lemma (10)

Another example of a language that can be shown not to be a TAL, using the pumping lemma, is $L_5 := \{a^n b^n c^n d^n e^n \mid n \geq 0\}$.

Kallmeyer, L. (2010). Parsing Beyond Context-Free Grammars. Cognitive Technologies. Springer, Heidelberg.

Kallmeyer, L. and Satta, G. (2009). A polynomial-time parsing algorithm for tt-mctag. In Proceedings of ACL, Singapore.

Pereira, F. C. N. and Warren, D. (1983). Parsing as deduction. In 21st Annual Meeting of the Association for Computational Linguistics, pages 137–âĂŞ144, MIT, Cambridge, Massachusetts.

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

Sikkel, K. (1997). Parsing Schemata. Texts in Theoretical Computer Science. Springer-Verlag, Berlin, Heidelberg, New York.

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.