

Parsing Beyond Context-Free Grammars: Tree Adjoining Grammar Parsing

Laura Kallmeyer, Patrick Hommers
Heinrich-Heine-Universität Düsseldorf

Sommersemester 2013

Parsing Beyond CFG	1	TAG Parsing
--------------------	---	-------------

Overview

1. Parsing as deduction
 - (a) Parsing schemata
 - (b) Chart parsing
2. A CYK recognizer for TAG
 - (a) Items
 - (b) Inference rules
 - (c) Complexity
3. Earley Parsing for TAG
 - (a) Introduction
 - (b) Items
 - (c) Inference rules

Parsing Beyond CFG	2	TAG Parsing
--------------------	---	-------------

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 Beyond CFG	3	TAG Parsing
--------------------	---	-------------

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 Beyond CFG	4	TAG Parsing
--------------------	---	-------------

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)

$\mathcal{C} = \mathcal{A} = \emptyset$

for all items I resulting from 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' resulting from a rule application

with antecedents I and items from \mathcal{C} :

if $I' \notin \mathcal{C}$:

add I' to \mathcal{C} and to \mathcal{A}

if there is a goal item in \mathcal{C} : output true

else: output 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 i th daughter of the node at address p),
- subscript $t \in \{\top, \perp\}$ specifies whether substitution or adjunction has already taken place (\top) or not (\perp) 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)

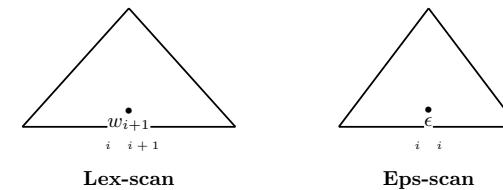
Goal items: $[\alpha, \epsilon_{\top}, 0, -, -, n]$ where $\alpha \in I$

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

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

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

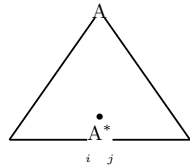
(Notation: $l(\gamma, p)$ is the label of the node at address p in γ .)

CYK for TAG: Inference rules (2)

CYK for TAG: 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:

$$\text{Foot-predict: } \frac{}{[\beta, p_{\top}, i, i, j, j]} \quad \beta \in A, p \text{ foot node address in } \beta, i \leq j$$



CYK for TAG: 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:

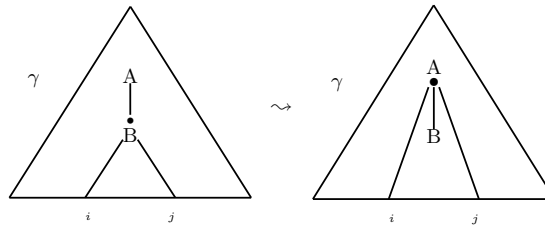
$$\frac{[\gamma, (p \cdot 1)_{\top}, i, f_1, f_2, j]}{[\gamma, p_{\perp}, i, f_1, f_2, j]} \quad \text{node address } p \cdot 2 \text{ does not exist in } \gamma$$

$$\text{Move-binary: } \frac{[\gamma, (p \cdot 1)_{\top}, i, f_1, f_2, k], [\gamma, (p \cdot 2)_{\top}, k, f'_1, f'_2, j]}{[\gamma, p_{\perp}, i, f_1 \oplus f'_1, f_2 \oplus f'_2, j]}$$

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

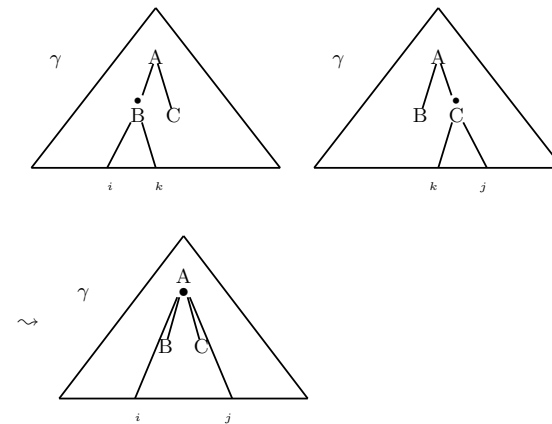
CYK for TAG: Inference rules (5)

Move-unary:



CYK for TAG: Inference rules (6)

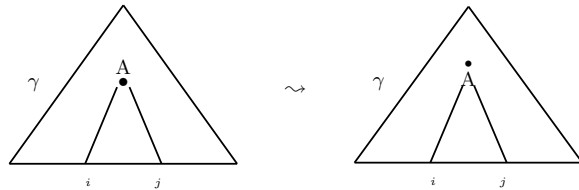
Move-binary:



CYK for TAG: Inference rules (7)

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

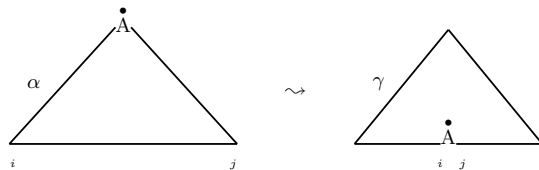
Null-adjoin:
$$\frac{[\gamma, p_{\perp}, i, f_1, f_2, j]}{[\gamma, p_{\top}, i, f_1, f_2, j]} f_{OA}(\gamma, p) = 0$$



CYK for TAG: Inference rules (8)

The rule **substitute** performs a substitution:

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



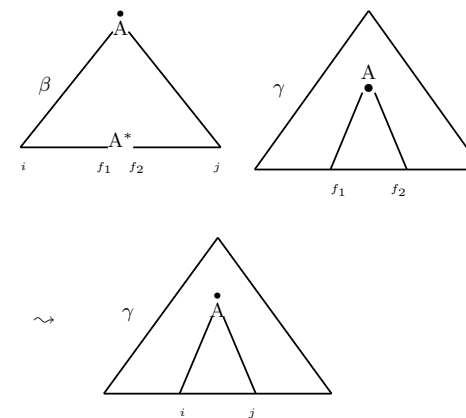
CYK for TAG: Inference rules (9)

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:
$$\frac{[\beta, \epsilon_{\top}, i, f_1, f_2, j], [\gamma, p_{\perp}, f_1, f'_1, f'_2, f_2]}{[\gamma, p_{\top}, i, f'_1, f'_2, j]} \beta \in f_{SA}(\gamma, p)$$

CYK for TAG: Inference rules (10)

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

Earley for TAG: Introduction (1)

- Left-to-right CYK parser very slow: $\mathcal{O}(n^6)$ worst case **and** best case (just as in CFG version of CYK, too many partial trees not pertinent to the final tree are produced).
- Behaviour is due to pure bottom-up approach, no predictive information whatsoever is used.
- Goal: Earley-style parser! First in [Schabes and Joshi, 1988]. Here, we present the algorithm from [Joshi and Schabes, 1997].

We assume a TAG without substitution nodes.

Earley for TAG: Introduction (2)

- Earley Parsing: Left-to-right scanning of the string (using predictions to restrict hypothesis space)
- Traversal of elementary trees, current position marked with a dot.

The dot can have exactly four positions with respect to the node: left above (la), left below (lb), right above (ra), right below (rb).

Earley for TAG: Introduction (3)

General idea: Whenever we are

- left above a node, we can predict an adjunction and start the traversal of the adjoined tree;
- left of a foot node, we can move back to the adjunction site and traverse the tree below it;
- right of an adjunction site, we continue the traversal of the adjoined tree at the right of its foot node;
- right above the root of an auxiliary tree, we can move back to the right of the adjunction site.

Earley for TAG: Items (1)

What kind of information do we need in an item characterizing a partial parsing result?

$$[\alpha, dot, pos, i, j, k, l, sat?]$$

where

- $\alpha \in I \cup A$ is a (dotted) tree, dot and pos the address and location of the dot
- i, j, k, l are indices on the input string, where $i, l \in \{0, \dots, n\}$, $j, k \in \{0, \dots, n\} \cup \{-\}$, $n = |w|$, $-$ means unbound value
- $sat?$ is a flag. It controls (prevents) multiple adjunctions at a single node ($sat? = 1$ means that something has already been adjoined to the dotted node)

Earley for TAG: Items (2)

What do the items mean?

- $[\alpha, dot, la, i, j, k, l, 0]$: In α part left of the dot ranges from i to l . If α is an auxiliary tree, part below foot node ranges from j to k .
- $[\alpha, dot, lb, i, -, -, i, 0]$: In α part below dotted node starts at position i .
- $[\alpha, dot, rb, i, j, k, l, sat?]$: In α part below dotted node ranges from i to l . If α is an auxiliary tree, part below foot node ranges from j to k . If $sat? = 0$, nothing was adjoined to dotted node, $sat? = 1$ means that adjunction took place.
- $[\alpha, dot, ra, i, j, k, l, 0]$: In α part left and below dotted node ranges from i to l . If α is an auxiliary tree, part below foot node ranges from j to k .

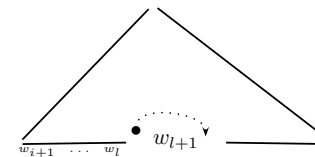
Earley for TAG: Items (3)

Some notational conventions:

- We use Gorn addresses for the nodes: 0 is the address of the root, i ($1 \leq i$) is the address of the i th daughter of the root, and for $p \neq 0$, $p \cdot i$ is the address of the i th daughter of the node at address p .
- For a tree α and a Gorn address dot , $\alpha(dot)$ denotes the node at address dot in α (if defined).

Earley for TAG: Inference Rules (1)

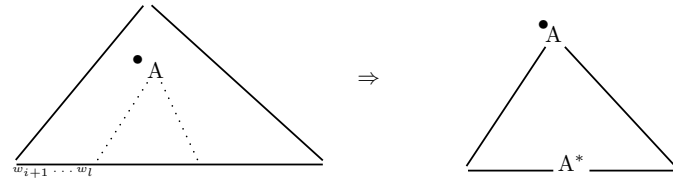
$$\text{ScanTerm} \frac{[\gamma, dot, la, i, j, k, l, 0]}{[\gamma, dot, ra, i, j, k, l + 1, 0]} \quad l(\gamma, dot) = w_{l+1}$$



$$\text{Scan-}\varepsilon \frac{[\gamma, dot, la, i, j, k, l, 0]}{[\gamma, dot, ra, i, j, k, l, 0]} \quad l(\gamma, dot) = \varepsilon$$

Earley for TAG: Inference Rules (2)

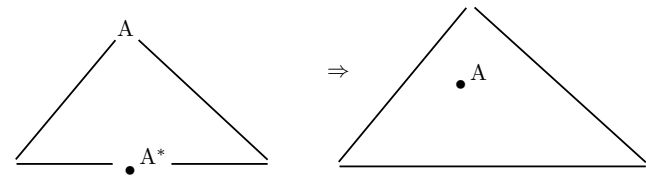
$$\text{PredictAdjoinable} \frac{[\gamma, \text{dot}, \text{la}, i, j, k, l, 0]}{[\beta, 0, \text{la}, l, -, -, l, 0]} \beta \in f_{SA}(\gamma, \text{dot})$$



$$\text{PredictNoAdj} \frac{[\gamma, \text{dot}, \text{la}, i, j, k, l, 0]}{[\gamma, \text{dot}, \text{lb}, l, -, -, l, 0]} f_{OA}(\gamma, \text{dot}) = 0$$

Earley for TAG: Inference Rules (3)

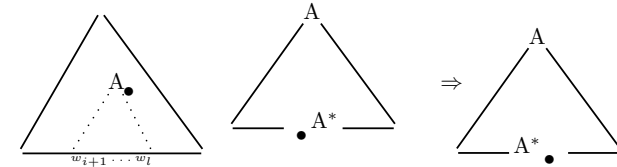
$$\text{PredictAdjoined} \frac{[\beta, \text{dot}, \text{lb}, l, -, -, l, 0]}{[\gamma, \text{dot}', \text{lb}, l, -, -, l, 0]} \text{dot} = \text{foot}(\beta), \beta \in f_{SA}(\gamma, \text{dot}')$$



Earley for TAG: Inference Rules (4)

CompleteFoot

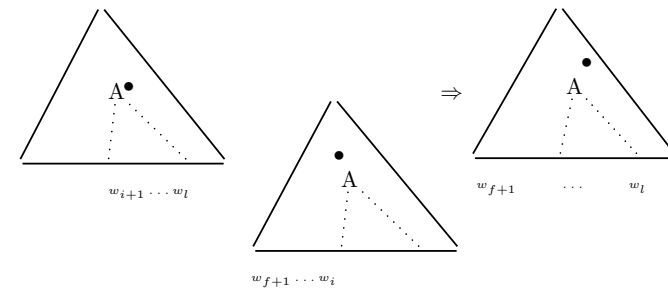
$$\frac{[\gamma, \text{dot}, \text{rb}, i, j, k, l, 0], [\beta, \text{dot}', \text{lb}, i, -, -, i, 0]}{[\beta, \text{dot}', \text{rb}, i, i, l, l, 0]} \begin{matrix} \text{dot}' = \text{foot}(\beta), \\ \beta \in f_{SA}(\gamma, \text{dot}') \end{matrix}$$



Earley for TAG: Inference Rules (5)

CompleteNode

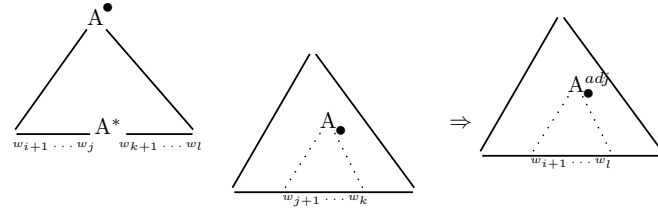
$$\frac{[\gamma, \text{dot}, \text{la}, f, g, h, i, 0], [\gamma, \text{dot}, \text{rb}, i, j, k, l, \text{sat?}]}{[\gamma, \text{dot}, \text{ra}, f, g \oplus j, h \oplus k, l, 0]} l(\beta, \text{dot}) \in N$$



Earley for TAG: Inference Rules (6)

Adjoin

$$\frac{[\beta, \varepsilon, ra, i, j, k, l, 0], [\gamma, dot, rb, j, p, q, k, 0]}{[\gamma, dot, rb, i, p, q, l, 1]} \quad \beta \in f_{SA}(\gamma, p)$$



$sat? = 1$ prevents the new item from being reused in another Adjoin application.

Earley for TAG: Inference Rules (7)

Move the dot to daughter/sister/mother:

$$\text{MoveDown: } \frac{[\gamma, dot, lb, i, j, k, l, 0]}{[\gamma, dot \cdot 1, la, i, j, k, l, 0]} \quad \gamma(dot \cdot 1) \text{ is defined}$$

$$\text{MoveRight: } \frac{[\gamma, dot, ra, i, j, k, l, 0]}{[\gamma, dot + 1, la, i, j, k, l, 0]} \quad \gamma(dot + 1) \text{ is defined}$$

$$\text{MoveUp: } \frac{[\gamma, dot \cdot m, ra, i, j, k, l, 0]}{[\gamma, dot, rb, i, j, k, l, 0]} \quad \gamma(dot \cdot m + 1) \text{ is not defined}$$

Earley for TAG: Inference Rules (8)

Rules for substitution:

$$\text{PredictSubst: } \frac{[\gamma, p, lb, i, -, -, i, 0]}{[\alpha, \varepsilon, la, i, -, -, i, 0]} \quad \begin{array}{l} \gamma(p) \text{ a substitution node,} \\ \alpha \in I, l(\gamma, p) = l(\alpha, \varepsilon) \end{array}$$

$$\text{Substitute: } \frac{[\alpha, \varepsilon, ra, i, -, -, j, 0]}{[\gamma, p, rb, i, -, -, j, 0]} \quad \begin{array}{l} \gamma(p) \text{ a substitution node,} \\ \alpha \in I, l(\gamma, p) = l(\alpha, \varepsilon) \end{array}$$

Note that **substitute** does not check whether a corresponding γ -item which had triggered the prediction of α exists. This check is done in the next step, when applying **completeNode** in order to combine the part to the left of the substitution node with the part below it.

Earley for TAG: Inference Rules (9)

$$\text{Initialize: } \frac{}{[\alpha, \varepsilon, la, 0, -, -, 0, 0]} \quad \alpha \in I, l(\alpha, \varepsilon) = S$$

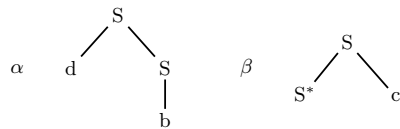
$$\text{Goal item: } [\alpha, 0, ra, 0, -, -, n, 0], \alpha \in I$$

Earley for TAG: The Valid Prefix Property (VPP) (1)

- The Earley algorithm, as presented, does not have the VPP.
- In other words, there are items which are not part of a derivation from an initial α with the span of the derived tree up to the dotted node being a prefix of a word in the language.

Earley for TAG: The Valid Prefix Property (VPP) (2)

Example:



Every word in the language starts with d .

Earley for TAG: The Valid Prefix Property (VPP) (3)

Input *bccc* leads (among others) to the following items:

	Item	Rule
1.	$[\alpha, \varepsilon, la, 0, -, -, 0, 0]$	initialize
2.	$[\beta, \varepsilon, la, 0, -, -, 0, 0]$	predictAdjoinable from 1.
...	...	
3.	$[\beta, 1, lb, 0, -, -, 0, 0]$	
4.	$[\alpha, 2, lb, 0, -, -, 0, 0]$	predictAdjoined from 3.
...	...	
5.	$[\alpha, 2, rb, 0, -, -, 1, 0]$	
6.	$[\beta, 1, rb, 0, 0, 1, 1, 0]$	completeFoot from 3. and 5.
...	...	
7.	$[\beta, \varepsilon, ra, 0, 0, 3, 4, 0]$	(after repeated adjunctions of β)
8.	$[\alpha, 2, rb, 0, -, -, 4, 1]$	adjoin from 7. and 4.

Earley for TAG: The Valid Prefix Property (VPP) (4)

- Reason for lack of VPP: neither **predictAdjoined** nor **completeFoot** nor **adjoin** check for the existence of an item that has triggered the prediction of this adjunction.
- Maintaining the VPP leads to deduction rules with more indices. It was therefore considered to be costly: $\mathcal{O}(n^9)$ [Schabes and Joshi, 1988].
- But: in some rules, some of the indices are not relevant for the rule and can be factored out (treated as “don’t care”-values). Therefore, a $\mathcal{O}(n^6)$ VPP Earley algorithm is actually possible [Nederhof, 1997].

References

- [Joshi and Schabes, 1997] Joshi, A. K. and Schabes, Y. (1997). Tree-Adjoining Grammars. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages*, pages 69–123. Springer, Berlin.
- [Kallmeyer, 2010] Kallmeyer, L. (2010). *Parsing Beyond Context-Free Grammars*. Cognitive Technologies. Springer, Heidelberg.
- [Kallmeyer and Satta, 2009] Kallmeyer, L. and Satta, G. (2009). A polynomial-time parsing algorithm for tt-mctag. In *Proceedings of ACL*, Singapore.
- [Nederhof, 1997] 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.
- [Pereira and Warren, 1983] Pereira, F. C. N. and Warren, D. (1983). Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, MIT, Cambridge, Massachusetts.
- [Schabes and Joshi, 1988] Schabes, Y. and Joshi, A. K. (1988). An Earley-type

Parsing Beyond CFG	41	TAG Parsing
--------------------	----	-------------

- parsing algorithm for Tree Adjoining Grammars. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 258–269.
- [Shieber et al., 1995] 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, 1997] Sikkel, K. (1997). *Parsing Schemata*. Texts in Theoretical Computer Science. Springer-Verlag, Berlin, Heidelberg, New York.
- [Vijay-Shanker and Joshi, 1985] 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.

Parsing Beyond CFG	42	TAG Parsing
--------------------	----	-------------