english

Parsing Beyond Context-Free Grammars: Tree Adjoining Grammar Parsing

Laura Kallmeyer
Heinrich-Heine-Universität Düsseldorf

Winter 2021/22

Overview

english

1. **Parsing as deduction**
   - Parsing schemata
   - Chart parsing
2. **CYK parsing for TAG**
   - Items
   - Inference rules
   - Complexity
   - CYK with dotted productions
3. **Earley Parsing for TAG**
   - Introduction
   - Items
   - Inference rules

[Kal10]

Parsing as deduction: Parsing Schemata (1)

[PW83, SSP95, Sik97]

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: $$\frac{}{[A, i, 1]} \quad A \to w_i \in P$$

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

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 binarized TAG: Items (1)

CYK-Parsing for TAG:

- First presented in [VSJ85], formulation with deduction rules in [KS09, Kal10].
- 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 binarized 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 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 $\alpha \in I$

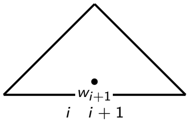
We need two rules to process leaf nodes while scanning their labels, depending on whether they have terminal labels or labels ϵ:

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

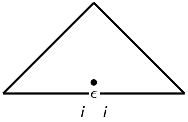
Eps-scan: $\dfrac{}{[\gamma, p_\top, i, -, -, i]} \quad l(\gamma, p) = \epsilon$

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

CYK for TAG: Inference rules (2)



Lex-scan

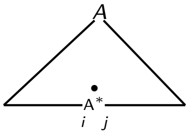


Eps-scan

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:

Foot-predict: $\dfrac{}{[\beta, p_\top, i, i, j, j]}$ $\beta \in A, p$ 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:

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

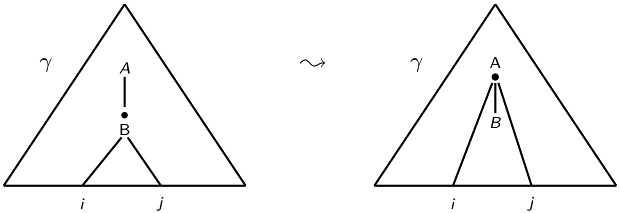
Move-binary:

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

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

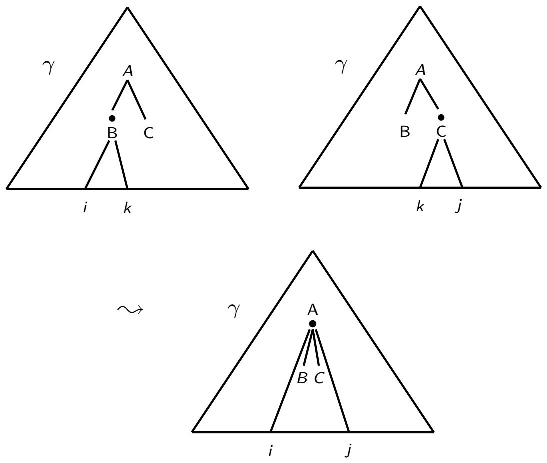
CYK for TAG: Inference rules (5)

Move-unary:



CYK for TAG: Inference rules (6)

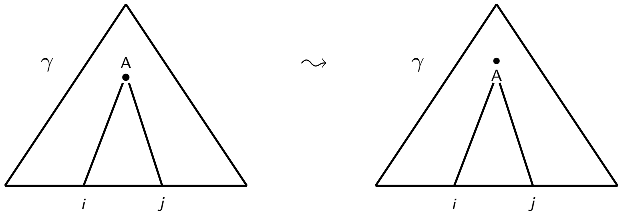
Move-binary:



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.

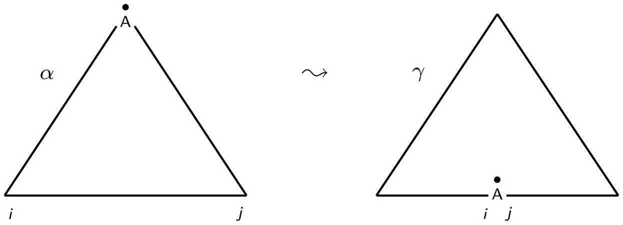
Null-adjoin: $\frac{[\gamma, p_\perp, i, f_1, f_2, j]}{[\gamma, p_\top, i, f_1, f_2, j]}$ $f_{OA}(\gamma, p) = 0$



CYK for TAG: Inference rules (8)

The rule **substitute** performes a substitution:

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



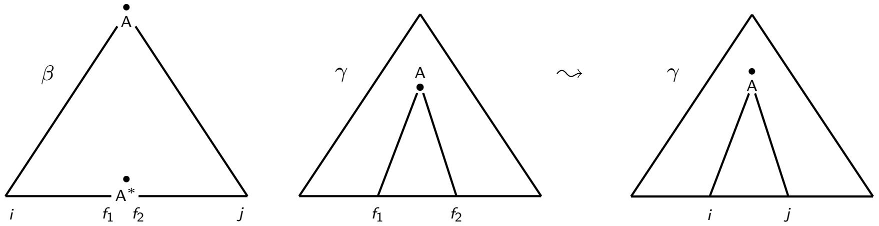
CYK for TAG: Inference rules (9)

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

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

CYK for TAG with dotted productions (1)

Alternative inspired by [Was17, WSP17, BWKJ19] that avoids the requirement of binary trees:

- We have passive and active items:
  - passive items talk about a node v in an elementary tree and the span below;
  - active items talk about a position between the daughters $v_1 \dots v_n$ of a node v, notated $v \to v_1 \dots v_i \bullet v_{i+1} \dots v_n$, and the span of the part to the left of that position, i.e., below $v_1 \dots v_i$
- Deduction rules move through the daughters from left to right.
- Items have the form $[v \to v_1 \dots v_k \bullet v_{k+1} \dots v_n, i, f_1, f_2, j]$ or $[v_t, i, f_1, f_2, j]$ with indices as above, v a node and t either $\bot$ or $\top$.[1]

Every such $v \to v_1 \dots v_n$ is called a *rule*.

[1] Nodes v can, as before, be encoded as pairs $\langle \gamma, p \rangle$.

CYK for TAG with dotted productions (2)

Goal items: all $[v_\top, 0, -, -, n]$ with $root(v)$ and $l(v) = S$.

Axioms: $$\frac{}{[v \to \bullet\gamma, i, -, -, i]} \quad v \to \gamma \text{ is a rule}$$

Lex-scan: $$\frac{[v \to \gamma_1 \bullet w\gamma_2, i, f_1, f_2, j]}{[v \to \gamma_1 w \bullet \gamma_2, i, f_1, f_2, j+1]} \quad l(w) = w_{j+1}$$

Eps-scan: $$\frac{[v \to \gamma_1 \bullet w\gamma_2, i, f_1, f_2, j]}{[v \to \gamma_1 w \bullet \gamma_2, i, f_1, f_2, j]} \quad l(w) = \varepsilon$$

Convert: $$\frac{[v \to \gamma\bullet, i, f_1, f_2, j]}{[v_\perp, i, f_1, f_2, j]}$$

CYK for TAG with dotted productions (3)

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

Move right: $$\frac{[v \to \gamma_1 \bullet w\gamma_2, i, f_1, f_2, j], [w_\top, j, f_3, f_4, k]}{[v \to \gamma_1 w \bullet \gamma_2, i, f_1 \oplus f_3, f_2 \oplus f_4, k]}$$

Substitute:
$$\frac{[v \to \gamma_1 \bullet w\gamma_2, i, f_1, f_2, j], [u, \top, j, -, -, k]}{[v \to \gamma_1 w \bullet \gamma_2, i, f_1, f_2, k]}$$
$l(w) = l(u)$, $root(u)$, w substitution node

Foot adjunction:
$$\frac{[v \to \gamma_1 \bullet w\gamma_2, i, -, -, j], [u_\perp, j, f_1, f_2, k]}{[v \to \gamma_1 w \bullet \gamma_2, i, j, k, k]}$$
$l(w) = l(u), foot(w)$, adj. of auxiliary tree with w allowed at u

Root adjunction:
$$\frac{[v_\top, i, j, k, l], [u, \perp, j, f_1, f_2, k]}{[u_\top, i, f_1, f_2, l]}$$
$l(v) = l(u), root(v)$, adj. of auxiliary tree with v allowed at u

CYK for TAG with dotted productions (4)

Differences between the two CYK algorithms:

- The binary CYK requires a binarization, which has to be undone after parsing.
- The dotted production CYK has more complex items.
- The binary CYK blindly predicts the span below foot nodes (see foot-predict, which does not have any antecedents) while the dotted production CYK starts processing a foot node only if the part below a possible adjunction site has been found.

CYK for TAG with dotted productions (5)

- The dotted production CYK blindly predicts all rules in the grammar, starting at any position in the input (see axioms). In contrast, the binary CYK considers a node only if its daughters have been found.
- The binary CYK blindly scans terminals and ε as soon as the input matches, no matter whether anything else from the tree has been found. In contrast, the dotted production CYK scans only if all left sisters of the leaf have been found.

Early Parsing for CFG

- bottom-up parser with top-down control, i.e., a bottom-up parsing that does only reductions that can be top-down predicted from S, or a
- top-down parser with bottom-up recognition
- at each time of parsing, one production $A \rightarrow X_1 \dots X_k$ is considered such that
  - some part $X_1 \dots X_i$ has already been bottom-up recognized (completed)
  - while some part $X_{i+1} \dots X_k$ has been top-down predicted.

This situation can be characterized by a dotted production (sometimes called Earley item) $A \rightarrow X_1 \dots X_i \bullet X_{i+1} \dots X_k$.
Dotted productions are called active items. Productions of the form $A \rightarrow \alpha\bullet$ are called completed items.

Early parsing: idea

The Earley parser simulates a top-down left-to-right depth-first traversal of the parse tree while moving the dot such that for each node

- first, the dot is to its left (the node is predicted),
- then the dot traverses the tree below,
- then the dot is to its right (the subtree below the node is completed)

Algorithm (1)

The items describing partial results of the parser contain a dotted production and the start and end index of the completed part of the rhs:

Item form: $[A \rightarrow \alpha \bullet \beta, i, j]$ with $A \rightarrow \alpha\beta \in P, 0 \leq i \leq j \leq n$.

Parsing starts with predicting all S-productions:

Axioms: $$\frac{}{[S \rightarrow \bullet\alpha, 0, 0]} \; S \rightarrow \alpha \in P$$

Algorithm (2)

If the dot of an item is followed by a non-terminal symbol B, a new B-production can be predicted. The completed part of the new item (still empty) starts at the index where the completed part of the first item ends.

$$\text{Predict:} \quad \frac{[A \rightarrow \alpha \bullet B\beta, i, j]}{[B \rightarrow \bullet\gamma, j, j]} \quad B \rightarrow \gamma \in P$$

If the dot of an item is followed by a terminal symbol a that is the next input symbol, then the dot can be moved over this terminal (the terminal is scanned). The end position of the completed part is incremented.

$$\text{Scan:} \quad \frac{[A \rightarrow \alpha \bullet a\beta, i, j]}{[A \rightarrow \alpha a \bullet \beta, i, j+1]} \quad w_{j+1} = a$$

Algorithm (3)

If the dot of an item is followed by a non-terminal symbol B and if there is a second item with a dotted B-production and a fully completed rhs and if, furthermore, the completed part of the second item starts at the position where the completed part of the first ends, then the dot in the first can be moved over the B while changing the end index to the end index of the completed B-production.

Complete: $$\frac{[A \rightarrow \alpha \bullet B\beta, i, j], [B \rightarrow \gamma\bullet, j, k]}{[A \rightarrow \alpha B \bullet \beta, i, k]}$$

The parser is successfull if a completed S-production spanning the entire input can be deduced:

Goal items: $[S \rightarrow \alpha\bullet, 0, n]$ for some $S \rightarrow \alpha \in P$.

Earley for TAG: Introduction (1)

- Left-to-right CYK parser very slow: 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 [SJ88]. Here, we present the algorithm from [JS97].

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 four different positions per 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.

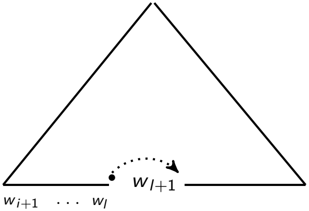
Earley for TAG: Items (3)

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 ith daughter of the root, and for $p \neq 0$, $p \cdot i$ is the address of the ith 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)

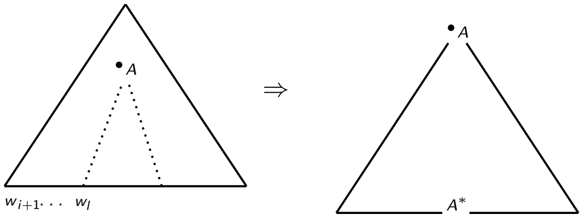
$$\text{ScanTerm} \quad \frac{[\gamma, dot, la, i, j, k, l, 0]}{[\gamma, dot, ra, i, j, k, l+1, 0]} \quad l(\gamma, dot) = w_{l+1}$$



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

Earley for TAG: Inference Rules (2)

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

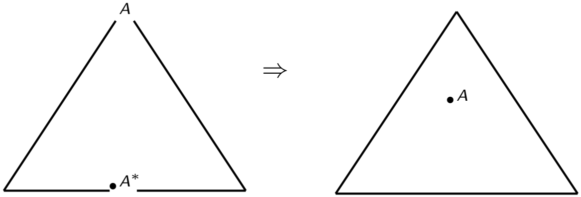


PredictNoAdj $$\frac{[\gamma, dot, la, i, j, k, l, 0]}{[\gamma, dot, lb, l, -, -, l, 0]} \quad f_{OA}(\gamma, dot) = 0$$

Earley for TAG: Inference Rules (3)

PredictAdjoined

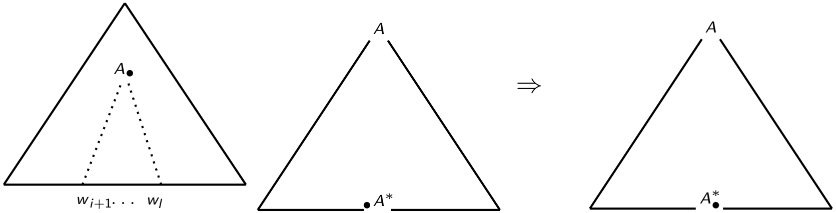
$$\frac{[\beta, dot, lb, l, -, -, l, 0]}{[\gamma, dot', lb, l, -, -, l, 0]} \quad dot = foot(\beta), \beta \in f_{SA}(\gamma, dot')$$



Earley for TAG: Inference Rules (4)

CompleteFoot

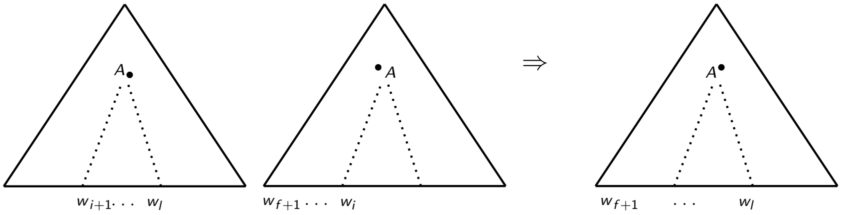
$$\frac{[\gamma, dot, rb, i, j, k, l, 0], [\beta, dot', lb, i, -, -, i, 0]}{[\beta, dot', rb, i, i, l, l, 0]} \quad \begin{array}{l} dot' = foot(\beta), \\ \beta \in f_{SA}(\gamma, dot') \end{array}$$



Earley for TAG: Inference Rules (5)

CompleteNode

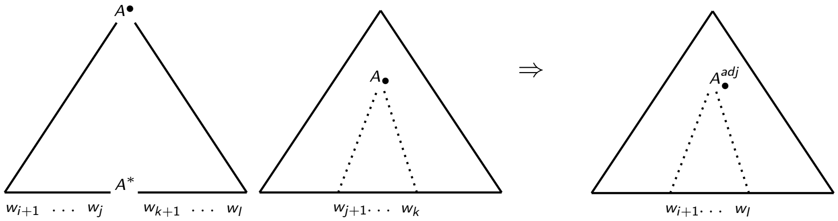
$$\frac{[\gamma, dot, la, f, g, h, i, 0], [\gamma, dot, rb, i, j, k, l, sat?]}{[\gamma, dot, ra, f, g \oplus j, h \oplus k, l, 0]} \quad l(\beta, dot) \in N$$



Earley for TAG: Inference Rules (6)

Adjoin

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



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

Earley for TAG: Inference Rules (7)

Move the dot to daughter/sister/mother:

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

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

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:

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

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

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)

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

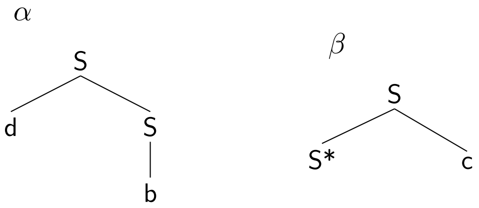
Goal item: $[\alpha, \epsilon, ra, 0, -, -, n, 0]$, $\alpha \in I$

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.	$[\alpha, \epsilon, la, 0, -, -, 0, 0]$	initialize
2.	$[\beta, \epsilon, la, 0, -, -, 0, 0]$	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 form 3. and 5.
	...	
7.	$[\beta, \epsilon, ra, 0, 0, 3, 4, 0]$	(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: $\mathcal{O}(n^9)$ [SJ88].
- 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 [Ned97].

References I

[BWKJ19] Tatiana Bladier, Jakub Waszczuk, Laura Kallmeyer, and Jörg Janke.
From partial neural graph-based LTAG parsing towards full parsing.
Computational Linguistics in the Netherlands Journal, 9:3–26, Dec. 2019.

[JS97] Aravind K. Joshi and Yves Schabes.
Tree-Adjoning Grammars.
In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, pages 69–123. Springer, Berlin, 1997.

[Kal10] Laura Kallmeyer.
Parsing Beyond Context-Free Grammars.
Cognitive Technologies. Springer, Heidelberg, 2010.

[KS09] Laura Kallmeyer and Giorgio Satta.
A polynomial-time parsing algorithm for tt-mctag.
In *Proceedings of ACL*, Singapore, 2009.

[Ned97] Mark-Jan Nederhof.
Solving the correct-prefix property for TAGs.
In T. Becker and H.-U. Krieger, editors, *Proceedings of the Fifth Meeting on Mathematics of Language*, pages 124–130, Schloss Dagstuhl, Saarbrücken, August 1997.

[PW83] Fernando C. N. Pereira and David Warren.
Parsing as deduction.
In *21st Annual Meeting of the Association for Computational Linguistics*, pages 137—144, MIT, Cambridge, Massachusetts, 1983.

[Sik97] Klaas Sikkel.
Parsing Schemata.
Texts in Theoretical Computer Science. Springer-Verlag, Berlin, Heidelberg, New York, 1997.

References II

[SJ88] Yves Schabes and Aravind K. Joshi.
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, 1988.

[SSP95] Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira.
Principles and implementation of deductive parsing.
Journal of Logic Programming, 24(1 and 2):3–36, 1995.

[VSJ85] K. Vijay-Shanker and Aravind K. Joshi.
Some computational properties of Tree Adjoining Grammars.
In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 82–93, 1985.

[Was17] Jakub Waszczuk.
Leveraging MWEs in practical TAG parsing: towards the best of the two world.
PhD thesis, 2017.

[WSP17] Jakub Waszczuk, Agata Savary, and Yannick Parmentier.
Multiword expression-aware A* TAG parsing revisited.
In *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms*, pages 84–93, Umeå, Sweden, September 2017. Association for Computational Linguistics.