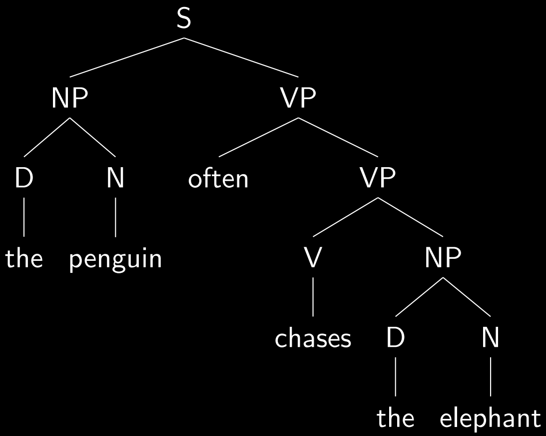
Lexicalized Tree-Adjoining Grammars (LTAG)

Fabian Reiter

January 18, 2012

Our Goal

We want to generate syntax trees like this:



Context-Free Grammars

Definition (Context-Free Grammar)

A context-free grammar (CFG) is a 4-tuple $G = (N, T, P, S)$ where:

- N is a finite set of non-terminal symbols.
- T is a finite set of terminal symbols, $N \cap T = \emptyset$.
- $P \subseteq N \times (N \cup T)^*$ is a finite set of production rules.
- $S \in N$ is a specific start symbol.

Example

$G = (N, T, P, \mathrm{S})$
where:

$N = \{\mathrm{S}, \mathrm{NP}, \mathrm{D}, \mathrm{N}, \mathrm{VP}, \mathrm{V}\}$

$T = \{\text{often, chases, helps, the, penguin, elephant}\}$

$$
\begin{aligned}
P: \quad \mathrm{S} &\rightarrow \mathrm{NP}\ \mathrm{VP} \\
\mathrm{NP} &\rightarrow \mathrm{D}\ \mathrm{N} \\
\mathrm{D} &\rightarrow \text{the} \\
\mathrm{N} &\rightarrow \text{penguin} \mid \text{elephant} \\
\mathrm{VP} &\rightarrow \text{often VP} \mid \mathrm{V}\ \mathrm{NP} \\
\mathrm{V} &\rightarrow \text{chases} \mid \text{helps}
\end{aligned}
$$

Context-Free Grammars

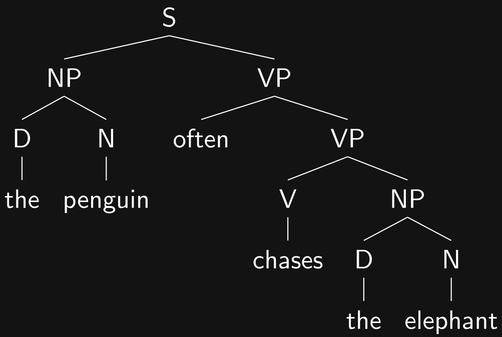
Example (Derivation with a CFG)

$G = (N, T, P, \mathrm{S})$
where:

$N = \{\mathrm{S, NP, D, N, VP, V}\}$

$T = \{\text{often, chases, helps, the, penguin, elephant}\}$

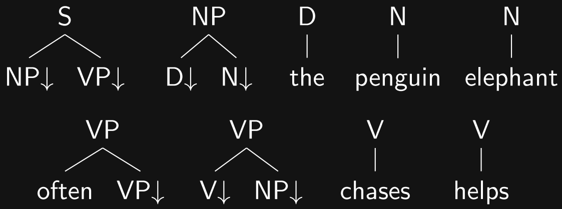
$$
\begin{aligned}
P: \quad \mathrm{S} &\rightarrow \mathrm{NP\ VP} \\
\mathrm{NP} &\rightarrow \mathrm{D\ N} \\
\mathrm{D} &\rightarrow \text{the} \\
\mathrm{N} &\rightarrow \text{penguin} \mid \text{elephant} \\
\mathrm{VP} &\rightarrow \text{often VP} \mid \mathrm{V\ NP} \\
\mathrm{V} &\rightarrow \text{chases} \mid \text{helps}
\end{aligned}
$$



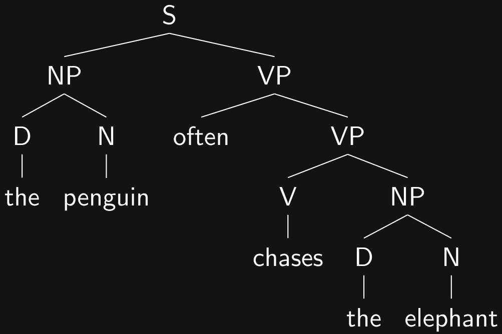
Tree-Substitution Grammars

Example (Derivation with a TSG)

Initial trees:

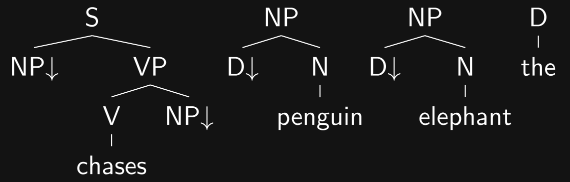


A derived tree:

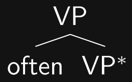


Example (Derivation with a TAG)

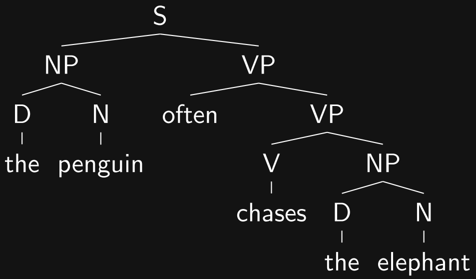
Initial trees:



Auxiliary tree:



A derived tree:



Outline

Outline

1. Why CFGs are not enough (for linguists)
   - Generative capacity
   - Lexicalization
2. Introduction to Tree-Adjoining Grammars
3. An Algorithm for Parsing TAGs
4. LTAG-Spinal Parser

Cross-Serial Dependencies

Example (Swiss German; Shieber, 1985)

			b	a	b	a
…	das	mer	em Hans	es huus	hälfed	aastriiche
…	that	we	Hans_{DAT}	house_{ACC}	helped	paint

'… that we helped Hans paint the house'

			a	b	a	a	b	a
…	das	mer	d'chind	em Hans	es huus	lönd	hälfe	aastriiche
…	that	we	the children_{ACC}	Hans_{DAT}	house_{ACC}	let	help	paint

'… that we let the children help Hans paint the house'

This can be reduced to the copy language $\{ww \mid w \in \{a, b\}^*\}$ which is not context-free.

Lexicalization

A grammar is lexicalized if each elementary structure is associated with at least one lexical item (terminal symbol), called its anchor.

Example (Lexicalized CFG)

$$S \rightarrow \text{Mary V} \mid \text{John V}$$
$$V \rightarrow \text{runs}$$

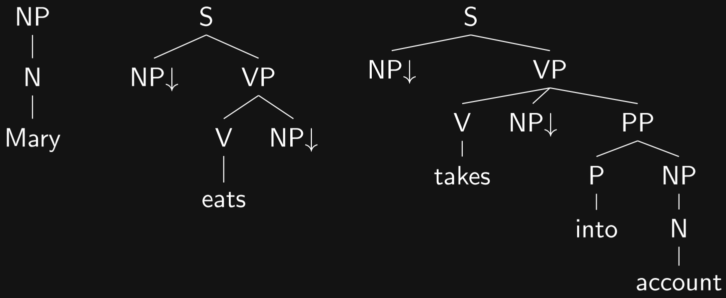
Example (Non-lex. CFG)

$$S \rightarrow \text{N V}$$
$$N \rightarrow \text{Mary} \mid \text{John}$$
$$V \rightarrow \text{runs}$$

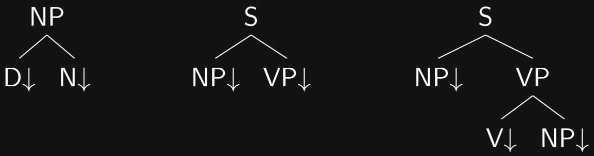
- Weak lexicalization of a grammar:
  Find a lexicalized grammar generating the same string language.
- Strong lexicalization of a grammar:
  Find a lexicalized grammar generating the same tree language.

Lexicalization

Example (Lexicalized initial trees)



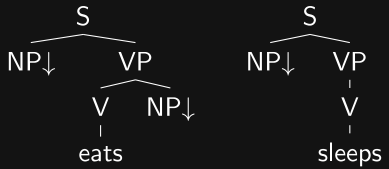
Example (Non-lexicalized initial trees)



Why Lexicalization?

- Syntactic structures associated with single words can be seen as more powerful POS-tags (“supertags”).

Example (Transitive vs. intransitive verb)



- (Finite) lexicalized grammars are finitely ambiguous.
  ⇒ The generated string languages are decidable.
- Lexicalization is useful for parsing since it allows us to drastically restrict the search space (as a preprocessing step).

Lexicalization

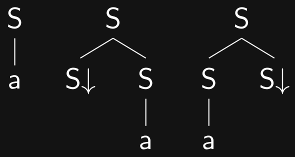
Example (CFG which is not strongly lexicalizable with a TSG)

Consider the following CFG:

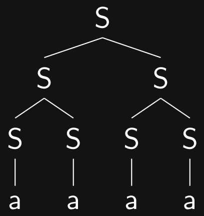
$S \rightarrow SS$

$S \rightarrow a$

An intuitive approach to lexicalize it with a TSG might be:



But this is not a strong lexicalization because it cannot generate the following tree (which the CFG can generate):



Problem: In TSGs the distance between two nodes in the same initial tree cannot increase during derivation.

Linguistic Shortcomings of CFG

Proposition (Shieber, 1985)

The language L of Swiss German is not context-free.

Proposition (Joshi and Schabes, 1997)

CFG cannot be strongly lexicalized by TSG (or CFG).

Outline

Definition (Tree-Adjoining Grammar)

A tree-adjoining grammar (TAG) is a 5-tuple $G = (N, T, I, A, S)$ where:

- N is a finite set of non-terminal symbols.
- T is a finite set of terminal symbols, $N \cap T = \emptyset$.
- I is a finite set of initial trees.
- A is a finite set of auxiliary trees.
- $S \in N$ is a specific start symbol.

The trees in $I \cup A$ are called elementary trees.

A Tree-Substitution Grammar (TSG) is defined analogously as a 4-tuple $G = (N, T, I, S)$, i.e. a TAG without auxiliary trees.

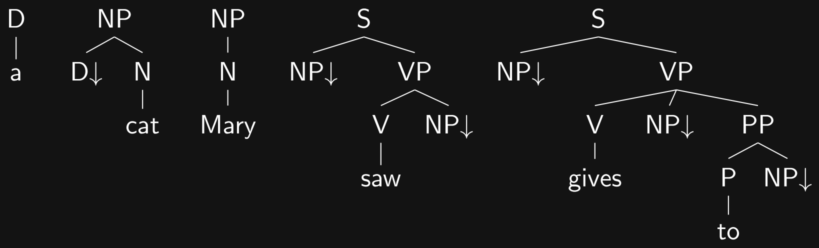
Initial Trees

Definition (Initial Tree)

An initial tree is characterized as follows:

- Internal nodes are only labeled by non-terminal symbols.
- Leaf nodes are labeled by terminals or non-terminals. If a leaf is labeled by a non-terminal, it is marked as substitution node (indicated by the symbol "↓").

Example



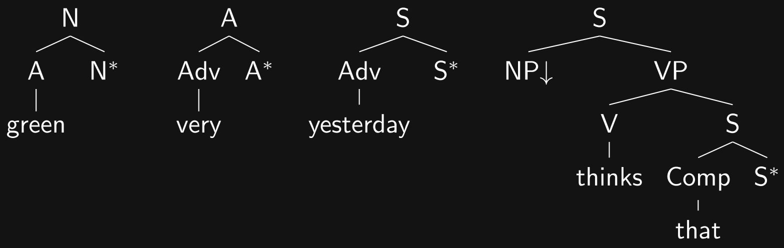
Auxiliary Trees

Definition (Auxiliary Tree)

An auxiliary tree has the same properties as an initial tree apart from one exception:

- Exactly one of the leaves labeled by a non-terminal is marked as the foot node (indicated by the symbol "$*$") instead of being marked for substitution. The label of the foot node must be identical to the label of the root node.

Example

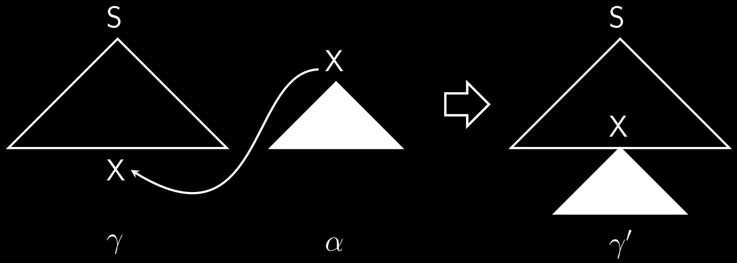


Substitution

Definition (Substitution)

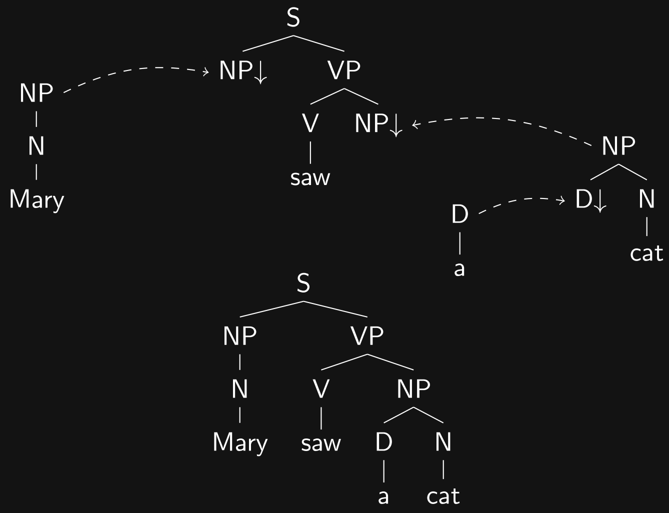
Let γ be a tree containing a substitution node *n* labeled by X and α an initial tree whose root node is also labeled by X.

By applying the substitution operation on (γ, n) and α, one gets a copy γ' of γ in which *n* has been replaced by α. If γ, *n*, α do not fulfill the above conditions, the operation is undefined.



Substitution

Example (Substitution)

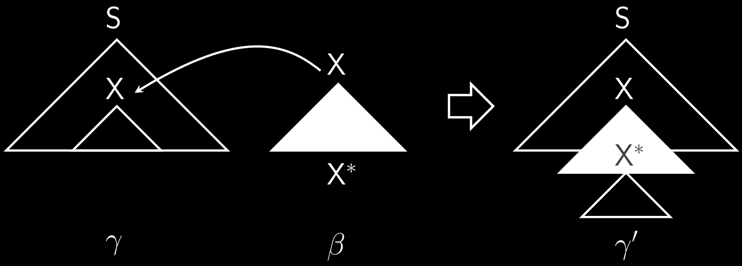


Adjunction

Definition (Adjunction)

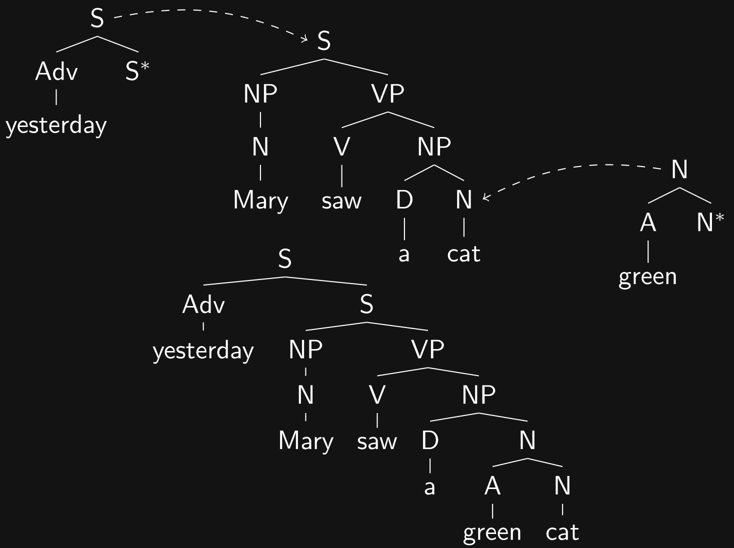
Let γ be a tree containing an <u>internal</u> node n labeled by X and β an <u>auxiliary</u> tree whose root node is also labeled by X.

By applying the adjunction operation on (γ, n) and β, one gets a copy γ' of γ in which β has taken the place of the subtree t rooted by n and t has been attached to the foot node of β. If γ, n, β do not fulfill the above conditions, the operation is undefined.



Adjunction

Example (Adjunction)



Adjunction Constraints

Given TAG $G = (N, T, I, A, S)$
We specify for each node n of a tree in $I \cup A$:

- $OA \in \{\bot, \top\}$: obligatory adjunction
  Boolean specifying whether adjunction at n is mandatory
- $SA \subseteq A$: selective adjunction
  Set of auxiliary trees authorized for adjunction at n

Also often used:

- $NA \in \{\bot, \top\}$: null adjunction
  Shorthand for the special case $OA = \bot \wedge SA = \emptyset$

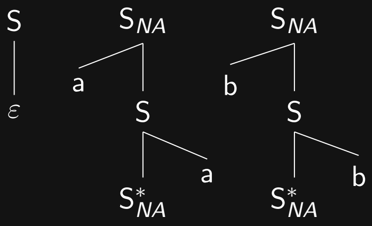
Remarks

- $OA = \top \wedge SA = \emptyset$ is not allowed.
- $\beta \in SA(n)$ only if root label of β equal to label of n.
- Substitution nodes must have $NA = \top$.

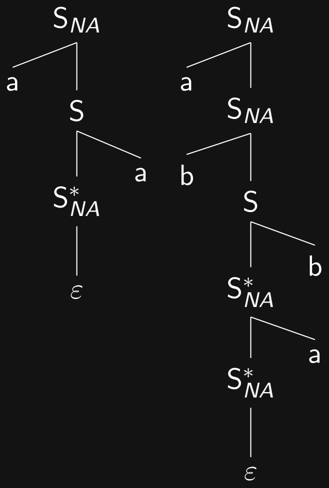
Cross-Serial Dependencies

Example (TAG for the copy language)

Generated string language:
$\{ww \mid w \in \{a, b\}^*\}$



Elementary trees



Some derived trees

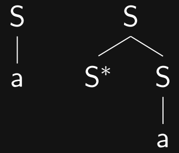
Lexicalization

Example (strong lexicalization of a CFG with a TAG)

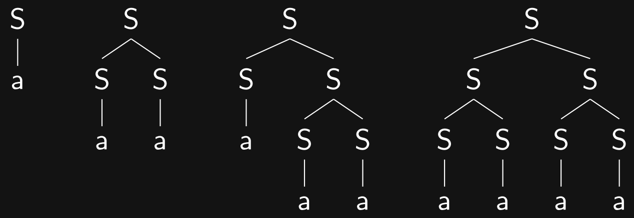
Consider again the following CFG:

$$S \rightarrow SS$$
$$S \rightarrow a$$

It can be easily lexicalized with a TAG by using adjunction:



By successive adjunction we get the following derived trees:



Lexicalization

Proposition (Joshi and Schabes, 1997)

Finitely ambiguous CFGs can be strongly lexicalized by TAGs.

Proposition (Joshi and Schabes, 1997)

Finitely ambiguous TAGs are closed under strong lexicalization.

Further Formal Properties of TAL

Tree-Adjoining Languages (TAL) have interesting formal properties, similar to those of context-free languages:

- TALs are closed under union, concatenation, iteration, substitution and intersection with regular languages.
- There is a pumping lemma for TAL.
- There is a class of automata which recognizes TAL: Embedded Push-Down Automata (EPDA).
- TALs can be parsed in polynomial time.

Outline

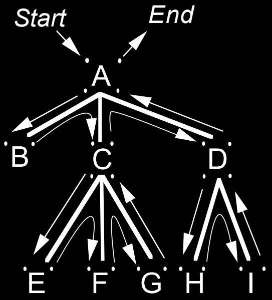
1. Why CFGs are not enough (for linguists)
2. Introduction to Tree-Adjoining Grammars
3. An Algorithm for Parsing TAGs
   - Preliminaries
   - The RECOGNIZER Algorithm
   - Complexity and Extensibility
4. LTAG-Spinal Parser

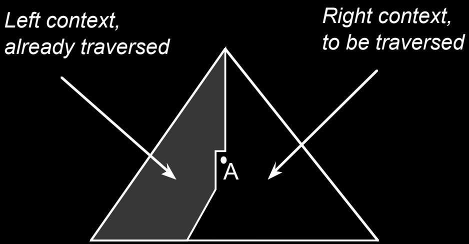
TAG Parsing

- Parser: Given a string s and a TAG $G = (N, T, I, A, S)$, find all derived trees in $L_{tree}(G)$ which yield s.
- We will start with a simpler problem:
  Recognizer: Given a string s and a TAG $G = (N, T, I, A, S)$, decide whether $s \in L_{string}(G)$.
- Further simplification:
  We will only consider the adjunction operation for now.

Tree Traversal

The algorithm will traverse every eligible derived tree (Euler tour) while scanning the input string from left to right.

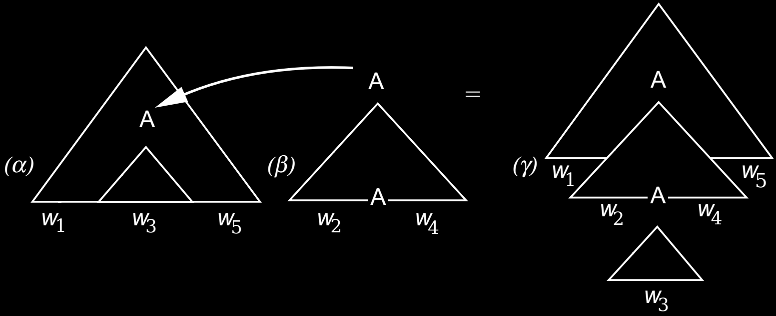




Recognizing Adjunction

But the algorithm never builds derived trees! It only uses the elementary trees of the input grammar.

Suppose that the following adjunction took place:

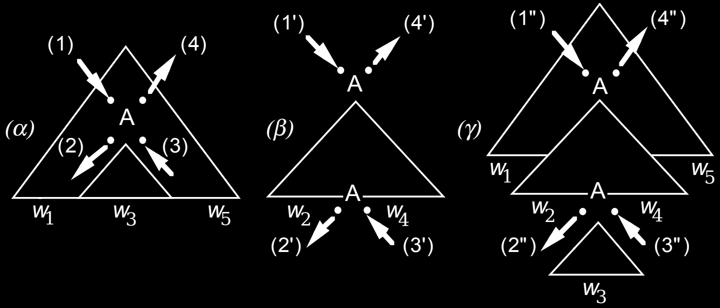


We need to traverse the derived tree γ but only have α and β at our disposal.

Recognizing Adjunction

If we could traverse γ, we would follow the path

$$\cdots 1'' \cdots 2'' \cdots 3'' \cdots 4'' \cdots$$



This can be simulated by traversing α and β such that the dots around the nodes labeled by A are visited in the following order:

$$\cdots 1\, 1' \cdots 2'\, 2 \cdots 3\, 3' \cdots 4' \cdots 4 \cdots$$

Dotted Tree

We introduce the notion of dotted tree.

It consists of:

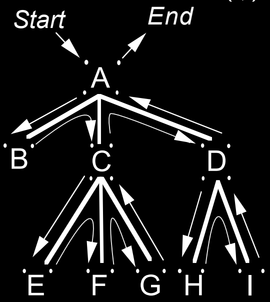
- a tree γ
- a dot location (adr, pos) where
  - adr is the Gorn address of a node in γ.
  - $pos \in \{la, lb, rb, ra\}$ is a relative position.

Definition (Gorn Address)

Given a node n in a tree γ, the Gorn address of n is:

- 0, if n is the root
- k, if n is the k^{th} child of the root
- $adr.k$, if n is the k^{th} child of the node at address adr, $adr \neq 0$

(γ)



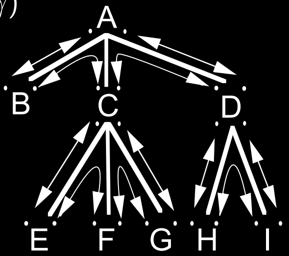
Example (Dotted trees)

- $\langle \gamma, 0, la \rangle$ $({}^{\bullet}\mathrm{A}\ )$
- $\langle \gamma, 3, rb \rangle$ $(\ \mathrm{D}_{\bullet})$
- $\langle \gamma, 2.1, ra \rangle$ $(\ \mathrm{E}^{\bullet})$

Equivalent Dot Positions

For the sake of convenience we will consider equivalent two successive dot positions (according to the tree traversal) that do not cross a node in the tree.

(γ)



Example (Equivalent dotted trees)

- $\langle \gamma, 0, lb \rangle \equiv \langle \gamma, 1, la \rangle$
- $\langle \gamma, 1, ra \rangle \equiv \langle \gamma, 2, la \rangle$
- $\langle \gamma, 2, lb \rangle \equiv \langle \gamma, 2.1, la \rangle$

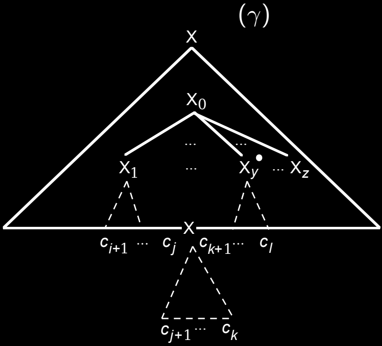
Chart Items

The algorithm stores intermediate results in a set of items called chart. Each item contains a dotted elementary tree and the corresponding range of the input string which has been recognized (by this item).

Definition (Chart Item)

An item is an 8-tuple $[\gamma, adr, pos, i, j, k, l, adj]$ where

- $\gamma \in I \cup A$ is an elementary tree.
- adr is the Gorn address of a node in γ.
- $pos \in \{la, lb, rb, ra\}$ is a relative position.
- i, j, k, l are indices on the input string. i, l delimit the range spanned by the dotted node and its left sibling nodes. j, k delimit the gap below the foot note if it exists. Otherwise their values are $-$.
- $adj \in \{\bot, \top\}$ is a boolean indicating whether an adjunction has been recognized at address adr in γ.



Outline of the Algorithm

- Initialize the chart $\mathcal{C}$ with items of the form $[\alpha, 0, la, 0, -, -, 0, \bot]$, where $\alpha \in I$, root label S.

- Then use 4 types of operations to add new items to $\mathcal{C}$:

  Scan, Predict, Complete, Adjoin

  Operations stated as inference rules:

$$\frac{\text{item}_1 \ \cdots \ \text{item}_m}{\text{item}_*} \quad \text{conditions}$$

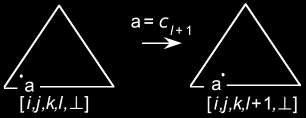
  Add item_* to $\mathcal{C}$ if $\text{item}_1, \cdots, \text{item}_m \in \mathcal{C}$ and conditions are met.

- Accept input string $c_1 \cdots c_n$ if $\mathcal{C}$ contains at least one item $[\alpha, 0, ra, 0, -, -, n, \bot]$, where $\alpha \in I$, root label S.

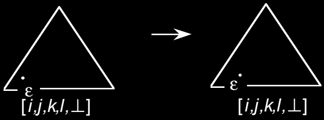
Scan Operations

Input string: $c_1 \cdots c_n$
Input TAG: $G = (N, T, I, A, S)$

1 $$\frac{[\gamma, adr, la, i, j, k, l, \bot]}{[\gamma, adr, ra, i, j, k, l+1, \bot]} \quad \gamma(adr) \in T, \ \gamma(adr) = c_{l+1}$$

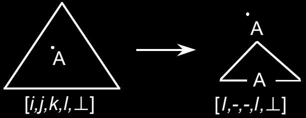


2 $$\frac{[\gamma, adr, la, i, j, k, l, \bot]}{[\gamma, adr, ra, i, j, k, l, \bot]} \quad \gamma(adr) = \varepsilon$$



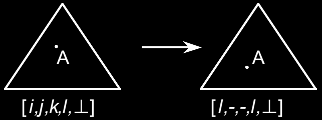
1

$$\frac{[\gamma, adr, la, i, j, k, l, \bot]}{[\beta, 0, la, l, -, -, l, \bot]} \quad \begin{array}{l} \gamma(adr) \in N, \\ \beta \in SA(\gamma, adr) \end{array}$$



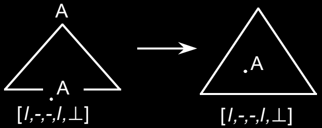
2

$$\frac{[\gamma, adr, la, i, j, k, l, \bot]}{[\gamma, adr, lb, l, -, -, l, \bot]} \quad \begin{array}{l} \gamma(adr) \in N, \\ OA(\gamma, adr) = \bot \end{array}$$



3

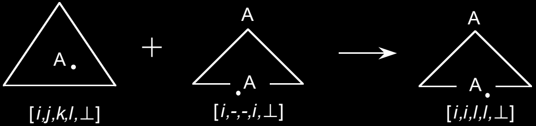
$$\frac{[\beta, adr, lb, l, -, -, l, \bot]}{[\gamma, adr', lb, l, -, -, l, \bot]} \quad \begin{array}{l} adr = \mathrm{foot}(\beta), \\ \beta \in SA(\gamma, adr') \end{array}$$



Complete Operations

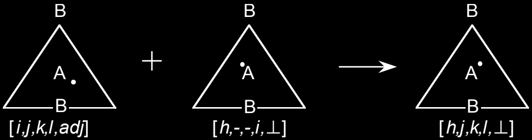
1

$$\frac{[\gamma, adr, rb, i, j, k, l, \bot] \qquad [\beta, adr', lb, i, -, -, i, \bot]}{[\beta, adr', rb, i, i, l, l, \bot]} \quad adr' = \mathrm{foot}(\beta),\ \beta \in SA(\gamma, adr)$$



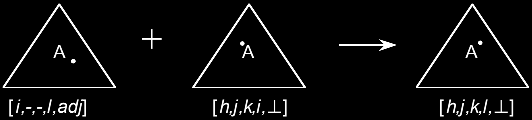
2

$$\frac{[\gamma, adr, rb, i, j, k, l, adj] \qquad [\gamma, adr, la, h, -, -, i, \bot]}{[\gamma, adr, ra, h, j, k, l, \bot]} \quad \gamma(adr) \in N$$



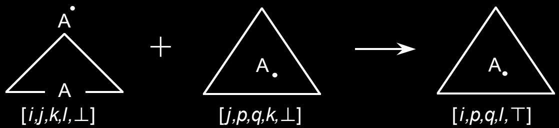
3

$$\frac{[\gamma, adr, rb, i, -, -, l, adj] \qquad [\gamma, adr, la, h, j, k, i, \bot]}{[\gamma, adr, ra, h, j, k, l, \bot]} \quad \gamma(adr) \in N$$



Adjoin Operation

- $$\frac{[\beta, 0, ra, i, j, k, l, \bot] \qquad [\gamma, adr, rb, j, p, q, k, \bot]}{[\gamma, adr, rb, i, p, q, l, \top]} \quad \beta \in SA(\gamma, adr)$$



RECOGNIZER Algorithm

Algorithm (RECOGNIZER; Joshi and Schabes, 1997)

Input: String $c_1 \cdots c_n$
TAG $G = (N, T, I, A, S)$ (that only allows adjunction)

- Initialize: $\mathcal{C} := \left\{ [\alpha, 0, la, 0, -, -, 0, \bot] \;\middle|\; \alpha \in I,\ \alpha(0) = S \right\}$
- While (new items can be added to $\mathcal{C}$)
  apply the following operations on each item in $\mathcal{C}$:

- $$\frac{[\gamma, adr, la, i, j, k, l, \bot]}{[\gamma, adr, ra, i, j, k, l+1, \bot]} \quad \begin{array}{l}\gamma(adr) \in T, \\ \gamma(adr) = c_{l+1}\end{array}$$
- $$\frac{[\gamma, adr, la, i, j, k, l, \bot]}{[\gamma, adr, ra, i, j, k, l, \bot]} \quad \gamma(adr) = \varepsilon$$
- $$\frac{[\gamma, adr, la, i, j, k, l, \bot]}{[\beta, 0, la, l, -, -, l, \bot]} \quad \begin{array}{l}\gamma(adr) \in N, \\ \beta \in SA(\gamma, adr)\end{array}$$
- $$\frac{[\gamma, adr, la, i, j, k, l, \bot]}{[\gamma, adr, lb, l, -, -, l, \bot]} \quad \begin{array}{l}\gamma(adr) \in N, \\ OA(\gamma, adr) = \bot\end{array}$$
- $$\frac{[\beta, adr, lb, l, -, -, l, \bot]}{[\gamma, adr', lb, l, -, -, l, \bot]} \quad \begin{array}{l}adr = \texttt{foot}(\beta), \\ \beta \in SA(\gamma, adr')\end{array}$$
- $$\frac{\begin{array}{c}[\gamma, adr, rb, i, j, k, l, \bot] \\ [\beta, adr', lb, i, -, -, i, \bot]\end{array}}{[\beta, adr', rb, i, i, l, l, \bot]} \quad \begin{array}{l}adr' = \texttt{foot}(\beta), \\ \beta \in SA(\gamma, adr)\end{array}$$
- $$\frac{\begin{array}{c}[\gamma, adr, rb, i, j, k, l, adj] \\ [\gamma, adr, la, h, -, -, i, \bot]\end{array}}{[\gamma, adr, ra, h, j, k, l, \bot]} \quad \gamma(adr) \in N$$
- $$\frac{\begin{array}{c}[\gamma, adr, rb, i, -, -, l, adj] \\ [\gamma, adr, la, h, j, k, i, \bot]\end{array}}{[\gamma, adr, ra, h, j, k, l, \bot]} \quad \gamma(adr) \in N$$
- $$\frac{\begin{array}{c}[\beta, 0, ra, i, j, k, l, \bot] \\ [\gamma, adr, rb, j, p, q, k, \bot]\end{array}}{[\gamma, adr, rb, i, p, q, l, \top]} \quad \beta \in SA(\gamma, adr)$$

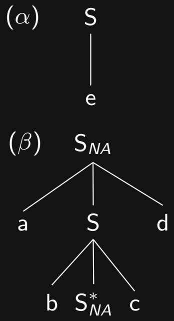
Output: If $\left(\exists\, [\alpha, 0, ra, 0, -, -, n, \bot] \in \mathcal{C} : \alpha \in I, \alpha(0) = S \right)$
then return acceptance else return rejection

Recognizer Algorithm

Example (execution of Recognizer)

Input string:
abecd

Input TAG:



Gen. language:
$\{a^n b^n e\, c^n d^n \mid n \geq 0\}$

Input read	#	Item added to chart $[\gamma, adr, pos, i, j, k, l, adj]$	Operation
	1.	$[\alpha, 0, la, 0, -, -, 0, \perp]$	initialization
	2.	$[\beta, 0, la, 0, -, -, 0, \perp]$	$\text{PRED}_1(1)$
	3.	$[\alpha, 1, la, 0, -, -, 0, \perp]$	$\text{PRED}_2(1)$
	4.	$[\beta, 1, la, 0, -, -, 0, \perp]$	$\text{PRED}_2(2)$
a	5.	$[\beta, 2, la, 0, -, -, 1, \perp]$	$\text{SCAN}_1(4)$
a	6.	$[\beta, 0, la, 1, -, -, 1, \perp]$	$\text{PRED}_1(5)$
a	7.	$[\beta, 2.1, la, 1, -, -, 1, \perp]$	$\text{PRED}_2(5)$
a	8.	$[\beta, 1, la, 1, -, -, 1, \perp]$	$\text{PRED}_2(6)$
ab	9.	$[\beta, 2.2, la, 1, -, -, 2, \perp]$	$\text{SCAN}_1(7)$
ab	10.	$[\beta, 2.2, lb, 2, -, -, 2, \perp]$	$\text{PRED}_2(9)$
ab	11.	$[\alpha, 1, la, 2, -, -, 2, \perp]$	$\text{PRED}_3(10)$
ab	12.	$[\beta, 2.1, la, 2, -, -, 2, \perp]$	$\text{PRED}_3(10)$
abe	13.	$[\alpha, 0, rb, 2, -, -, 3, \perp]$	$\text{SCAN}_1(11)$
abe	14.	$[\beta, 2.2, rb, 2, 2, 3, 3, \perp]$	$\text{COMP}_1(13, 10)$
abe	15.	$[\beta, 2.3, la, 1, 2, 3, 3, \perp]$	$\text{COMP}_2(14, 9)$
abec	16.	$[\beta, 2, rb, 1, 2, 3, 4, \perp]$	$\text{SCAN}_1(15)$
abec	17.	$[\beta, 3, la, 0, 2, 3, 4, \perp]$	$\text{COMP}_2(16, 5)$
abecd	18.	$[\beta, 0, rb, 0, 2, 3, 5, \perp]$	$\text{SCAN}_1(17)$
abecd	19.	$[\beta, 0, ra, 0, 2, 3, 5, \perp]$	$\text{COMP}_2(18, 2)$
abecd	20.	$[\alpha, 0, rb, 0, -, -, 5, \top]$	$\text{ADJ}(19, 13)$
abecd	21.	$[\alpha, 0, ra, 0, -, -, 5, \perp]$	$\text{COMP}_3(20, 1)$

Complexity of RECOGNIZER

Given:

- n: length of the input string
- $G = (N, T, I, A, S)$: input TAG
- m: maximal number of internal nodes per tree in $I \cup A$

Worst-case complexity can be reached by the ADJOIN operation:

$$\frac{[\beta, 0, ra, i, j, k, l, \bot] \qquad [\gamma, adr, rb, j, p, q, k, \bot]}{[\gamma, adr, rb, i, p, q, l, \top]} \quad \beta \in SA(\gamma, adr)$$

At most:

- $|A|$ possibilities for β
- $|I \cup A|$ possibilities for γ
- m possibilities for adr
- $n+2$ possibilities per index $\quad (i, \cdots, q \in \{0, \cdots, n\} \cup \{-\})$

$\Rightarrow$ ADJOIN can be applied at most $|A| \cdot |I \cup A| \cdot m \cdot (n+2)^6$ times.

$\Rightarrow$ Time complexity of RECOGNIZER: $\mathcal{O}(|A| \cdot |I \cup A| \cdot m \cdot n^6)$

$\Rightarrow$ For a specific grammar: $\mathcal{O}(n^6)$

Extending RECOGNIZER to a Parser

- RECOGNIZER can be easily extended to a parser by remembering why items were placed into the chart.
- We can use items of the form

  $$[\gamma, adr, pos, i, j, k, l, adj, P]$$

  where P is a set of pointers/pairs of pointers to items which caused the item to exist.
- Results in a graph of all possible derivations.
- Time complexity remains the same, i.e. $\mathcal{O}(n^6)$.

Recognizing Substitution

- $\textsc{Recognizer}$ can be extended by two rules for substitution:

$$\textsc{Predict}_{\textsc{Subst}}: \frac{[\gamma, adr, lb, i, -, -, i, \bot]}{[\alpha, 0, la, i, -, -, i, \bot]} \quad \alpha \in SS(\gamma, adr)$$

$$\textsc{Substitute}: \frac{[\alpha, 0, ra, i, -, -, l, \bot]}{[\gamma, adr, rb, i, -, -, l, \bot]} \quad \alpha \in SS(\gamma, adr)$$

$SS(\gamma, adr) \subseteq I$: set of trees substitutable at node (γ, adr), empty if (γ, adr) not a substitution node

- Time complexity remains the same, i.e. $\mathcal{O}(n^6)$.

Outline

LTAG-Spinal Parser

LTAG-spinal:
Roughly speaking, a subset of LTAG, where every elementary tree is in spinal form (no branching, except for footnodes).

We look at the left-to-right incremental LTAG-spinal parser by Shen and Joshi (2005), implemented in Java.

Input: POS-tagged sentences

```
Donald_NNP is_VBZ most_RBS famous_JJ for_IN his_PRP$
semi-intelligible_JJ speech_NN and_CC his_PRP$
explosive_JJ temper_NN ._.
```

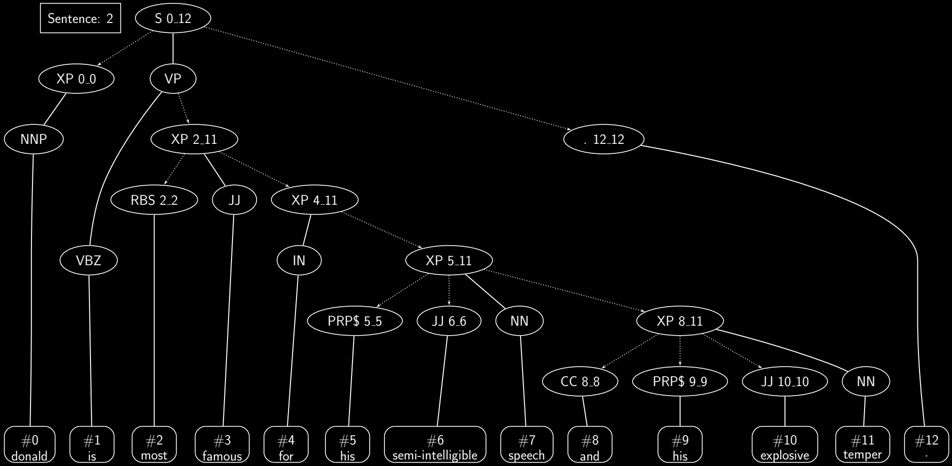
LTAG-Spinal Parser

Output:

```
2
root 1
#0 donald
 spine: a_( XP NNP^ )
#1 is
 spine: a_( S ( VP VBZ^ ) )
 att #0, on 0, slot 0, order 0
 att #3, on 0.0, slot 1, order 0
 att #12, on 0, slot 1, order 0
#2 most
 spine: a_RBS^
#3 famous
 spine: a_( XP JJ^ )
 att #2, on 0, slot 0, order 0
 att #4, on 0, slot 1, order 0
#4 for
 spine: a_( XP IN^ )
 att #7, on 0, slot 1, order 0
#5 his
 spine: a_PRP$^
#6 semi-intelligible
 spine: a_JJ^
#7 speech
 spine: a_( XP NN^ )
 att #5, on 0, slot 0, order 0
 att #6, on 0, slot 0, order 1
 att #11, on 0, slot 1, order 0
#8 and
 spine: a_CC^
#9 his
 spine: a_PRP$^
#10 explosive
 spine: a_JJ^
#11 temper
 spine: a_( XP NN^ )
 att #8, on 0, slot 0, order 0
 att #9, on 0, slot 0, order 1
 att #10, on 0, slot 0, order 2
#12 .
 spine: a_.^
```

LTAG-Spinal Parser

Graphical representation of the output:



LTAG-Spinal Parser - Tests

Test data: 2401 sentences from section 23 of the Penn Treebank

- Test system of Shen and Joshi (2005):
  2 × 1.13 GHz Pentium III, 2 GB RAM

  By varying some settings of their algorithm, they get:

sen/sec	f-score (%)
0.79	88.7
⋮	⋮
0.07	94.2

- Our test system (stromboli):
  16 × 2.80 GHz Xeon X5560, 35 GB RAM

  I performed two series of measurements:

  - default settings
  - settings closer to S&J ?

sen/sec	f-score (%)
10.20	?
3.22	?

Conclusion

- TAG: a grammar formalism related to CFG, but more powerful
- Very interesting from the theoretical point of view (mathematical and linguistical)
- Parsable in polynomial time, but with a high exponent: $\mathcal{O}(n^6)$
- Some recent research focuses on a subset, LTAG-spinal.

References

Joshi, A. K. and Schabes, Y. (1997)
Tree-Adjoining Grammars.
In Salomma, A. and Rosenberg, G., editors, *Handbook of Formal Languages and Automata*, volume 3, pages 69–124. Springer.

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

Abeillé, A. and Rambow, O. (2000)
Tree Adjoining Grammar: An Overview.
In Abeillé, A. and Rambow, O., editors, *Tree Adjoining Grammars: Formalisms, Linguistic Analyses and Processing*, volume 107 of CSLI Lecture Notes, pages 1–68. CSLI Publications, Stanford.

Shen, L. and Joshi, A. K. (2005)
Incremental LTAG Parsing.
In *Proceedings of the Human Language Technology Conference / Conference of Empirical Methods in Natural Language Processing (HLT/EMNLP)*.