Parsing Beyond Context-Free Grammars: Data-driven TAG parsing (TIG, osTAG)

Laura Kallmeyer & Tatiana Bladier
Heinrich-Heine-Universität Düsseldorf

Sommersemester 2018

Overview

Idea: TAG parsing in cubic time

- The main disadvantage of using TAGs for practical NLP applications are the (rather) high computation costs $\mathcal{O}(n^6)$
- With certain modifications and restrictions on the formalism, parsing with TAGs in cubic time ($\mathcal{O}(n^3)$) is possible
- Tree insertion grammar (TIG) - a compromise between CFG and TAG [SW95]
  - ⋆ Best of two worlds: efficiency of CFG parsing and lexicalizing power of TAG
  - ⋆ MICA parser (off-the-shelf, freely available) for TIGs [BBN+09]
- Off-spine TAG (OSTAG) – a variant of TAG with additional constraints for cubic time parsing [SYCS13]

Tree Insertion Grammar (TIG): motivation

- Lexicalizing of context-free grammars enables faster parsing
  - ⋆ Greibach normal form (1965) (without ϵ productions) [Gre65]
    $A \rightarrow a$
    $A \rightarrow aA_1 \ldots A_n$
- Very large output grammars $\Rightarrow$ awkward or impossible to use

CFG	**CFG in Greibach NF**
	$S \rightarrow aT_bST_b$
$S \rightarrow abSb$	$S \rightarrow aT_a$
$S \rightarrow aa$	$T_a \rightarrow a$
	$T_b \rightarrow b$

Tree Insertion Grammar (TIG): motivation

- Lexicalized CFGs allow parsing in cubic time
- Conversion to lexicalized CFGs $\Rightarrow$ *weak* lexicalization
  - ⋆ the strings are preserved
  - ⋆ derived trees are not preserved $\Rightarrow$ wrong trees
- *Strong* lexicalization is possible with context-sensitive formalisms
  - ⋆ TAG, Linear indexed grammar (LIG), Combinatory categorial grammar (CCG), linear context-free rewriting systems (LCFRS)
- Larger computation costs than CFGs ($\mathcal{O}(n^6)$) for TAG

- Tree Insertion Grammar is a compromise between CFG and TAG:
  - ⋆ Efficiency of CFG parsing and strong lexicalizing power of TAG
  - ⋆ TIGs can be parsed in cubic time
  - ⋆ Grammars are smaller compared to CFGs

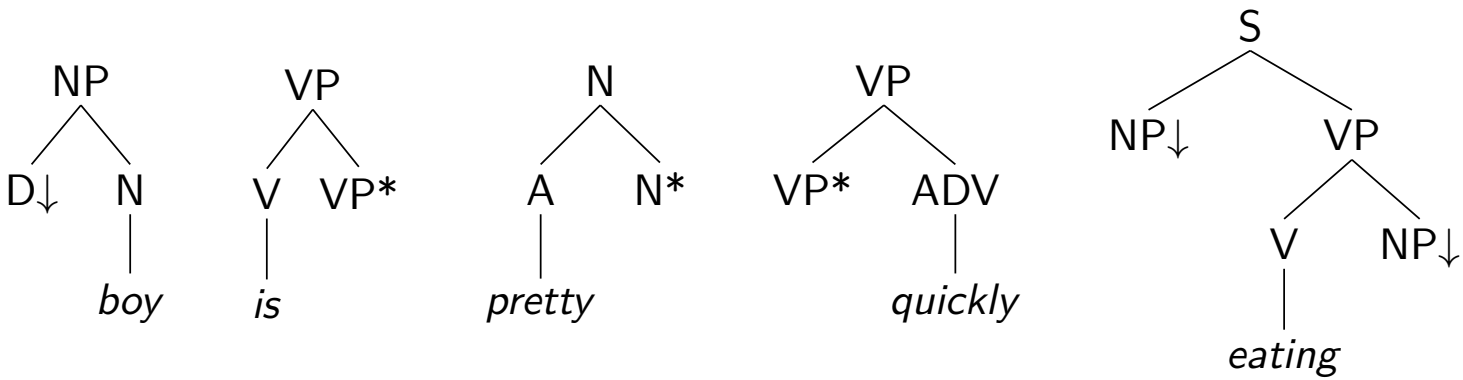
Tree Insertion Grammar

- Tree insertion grammar [SW95] is related to TAG
- A TIG is defined as a tuple $G = \langle N, T, S, I, A \rangle$ such that
  - T and N are disjoint alphabets, the terminals and nonterminals,
  - $S \in N$ is the start symbol,
  - I is a finite set of finite initial trees, and
  - A is a finite set of finite auxiliary trees.
- TIG also has substitution and adjunction operations
- TIGs can generate context-free languages

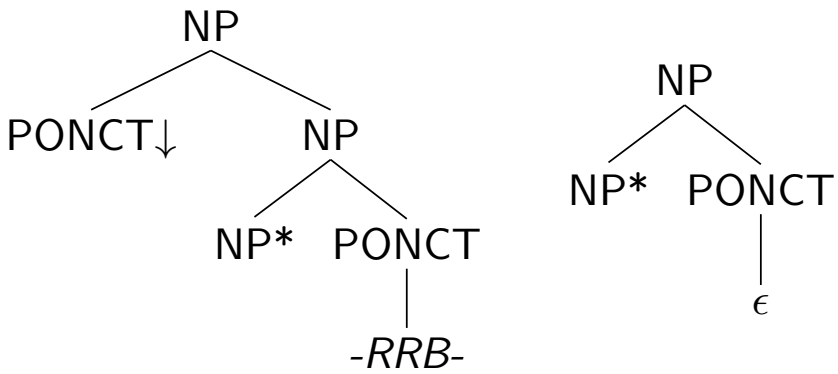
  $\Rightarrow$ TIG formalism is weaker than TAG, but might be sufficient (dependent on the goals of the NLP project)
- Substitution operation is the same as in TAG
- Adjunction operation is different:
  - Adjunction is restricted $\Rightarrow$ TIGs derive only context-free languages

Tree Insertion Grammar: Restrictions on adjunction

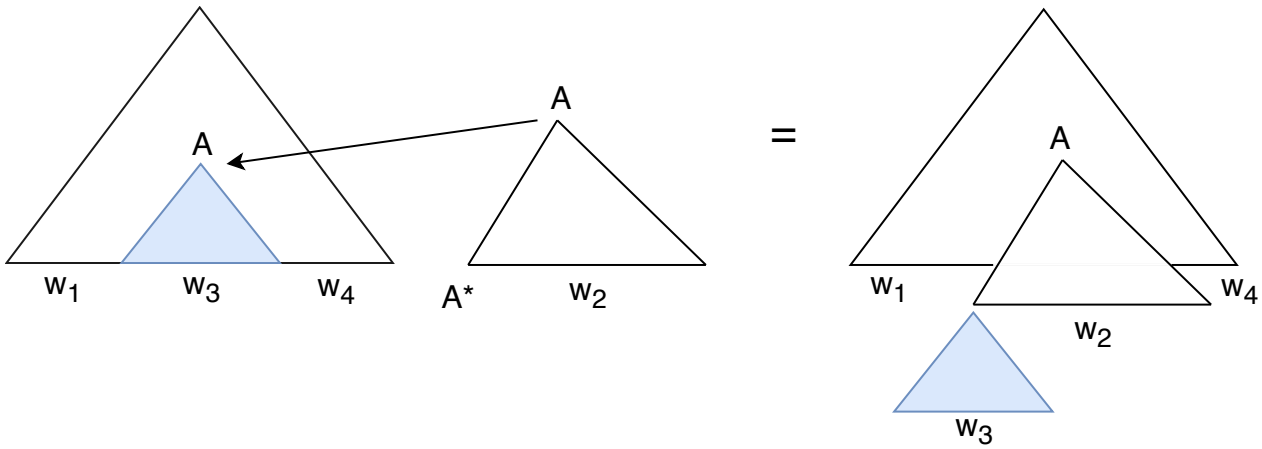
- Only non-empty *right auxiliary trees* or *left auxiliary trees* are in

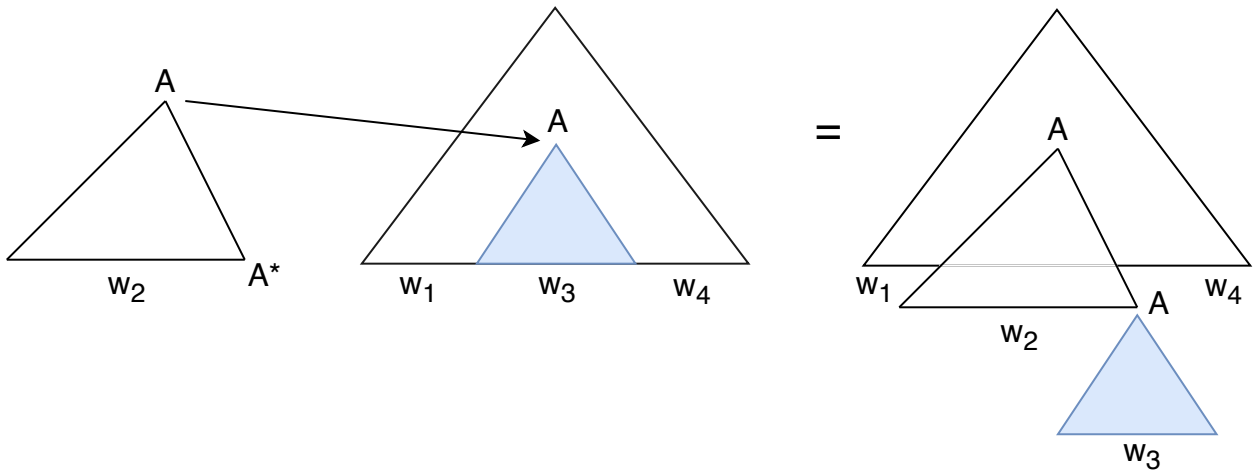


- *Wrapping auxiliary trees* and *empty auxiliary trees* are out

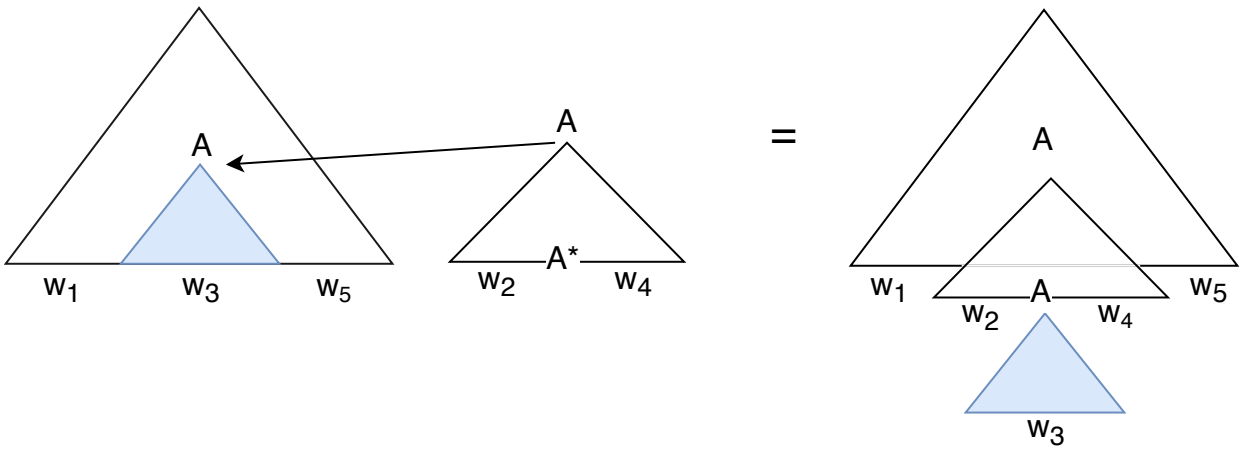


Right- and left-adjunction





Wrapping adjunction

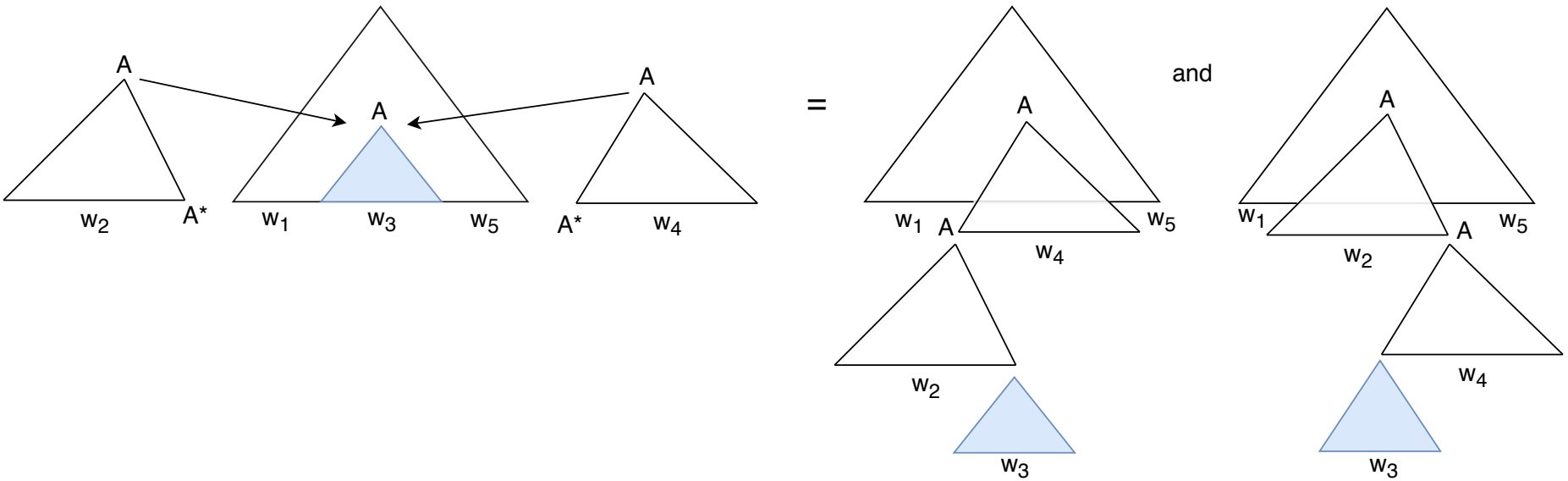


Tree Insertion Grammar: Restrictions on adjunction

- Simultaneous multiple adjunction is allowed
- TIG allows multiple simultaneous adjunction on a single node
- Simultaneous adjunction $\neq$ wrapping adjunction

  $\Rightarrow$ strings are adjoined independently

Simultaneous multiple adjunction



Tree Insertion Grammar: Restrictions on adjunction

- Adjunction is not allowed: at the substitution nodes, at the foot nodes, and at the root of an auxiliary tree.
- A left(right) auxiliary tree is not allowed to be adjoined to the nodes on the spine of the right(left) auxiliary tree
- No adjunction is permitted on nodes which are to the right(left) of the spine of an elementary left(right) auxiliary tree

TIG: possible extensions

- Adding adjunction constraints
- Limit or forbid simultaneous adjunction (e.g. at most one left and right auxiliary tree)
- Stochastic parameters to control the probabilities of substitution and adjunction
- Additional requirement for a TIG to be lexicalized (LTIG)
  - ⋆ *Left(right) anchored* LTIG $\Rightarrow$ if every elementary tree is *left(right) anchored*

Relations between CFG, TIG and TAG

- TIGs generate context-free languages
- Any CFG can be converted to TIG
- TIG without adjoining constraints can be easily converted to CFG
- TIG prohibits wrapping adjunction
  - ⋆ trivially a TAG without alterations
- TIG prohibits adjunction on the root nodes of auxiliary trees
  - ⋆ TAG allows such adjunction
- TIG allows multiple simultaneous adjunction
  - ⋆ Such adjunction is not allowed in TAG
- TIG without adjoining constraints can be converted to TAG deriving the same trees
  - ⋆ If TIG uses adjoining constraints, such a conversion can be difficult

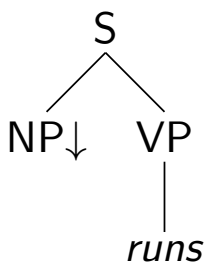
Conversion of TIG to CFG (1)

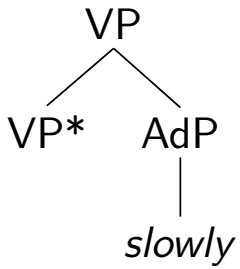
- TIG $G = \langle N, T, S, I, A \rangle$ and CFG $G' = \langle N', T', S', P \rangle$
- Step 1: For each nonterminal A_i in N, add two nonterminals Y_i and Z_i. This yields a new set N'.
- Step 2: For each nonterminal A_i in N, include the following rules in P: $Y_i \rightarrow \epsilon$, $Z_i \rightarrow \epsilon$.
- Step 3: Alter every node μ in every elementary tree in I and A as follows:
  - let A_i be the label of μ.
  - If left adjunction is possible at μ, add a new leftmost child of μ labeled Y_i and mark it for substitution.
  - If right adjunction is possible at μ, add a new rightmost child of μ labeled Z_i and mark it for substitution.

Conversion of TIG to CFG (2)

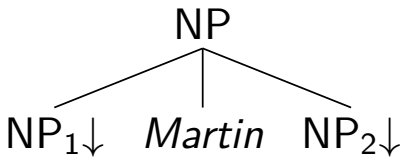
- Step 4: Convert every auxiliary tree t in A as follows:
  - let A_i be the label of the root μ of t.
  - If t is a left auxiliary tree, add a new root labeled Y_i with two children: μ on the left, and on the right, a node labeled Y_i and marked for substitution.
  - If t is a right auxiliary tree, add a new root labeled Z_i with two children: μ on the left, and on the right, a node labeled Z_i and marked for substitution.
  - Relabel the foot of t with ϵ, turning t into an initial tree.
- Step 5: Every elementary tree t is now initial tree.
  - Each t is converted into a rule in P as follows:
  - The label of the root of t becomes the left hand side of a rule.
  - The labels on the frontier of t with any instances of ϵ omitted become the right hand side of the rule.

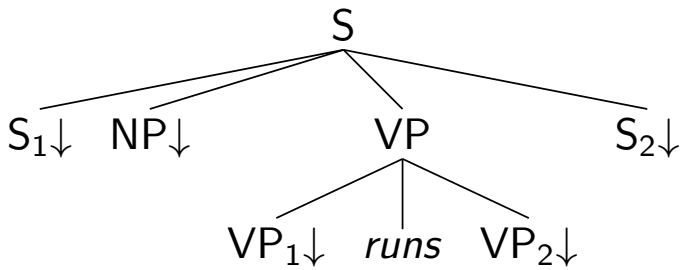
Conversion of TIG to CFG (3)

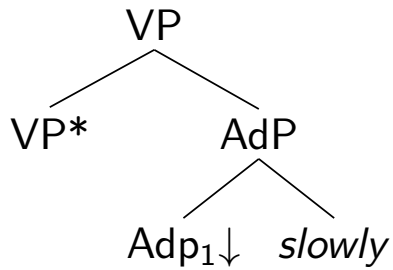


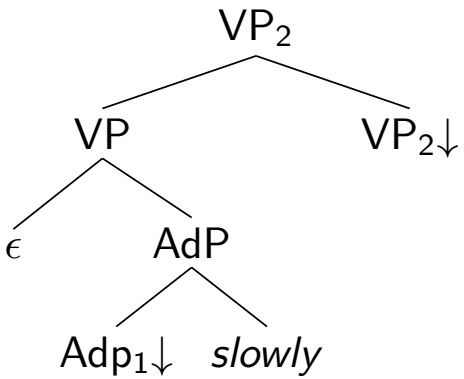


N = { NP, S, VP, AdP }
N' = { NP, NP_1, NP_2, S, S_1, S_2,
VP, VP_1, VP_2, AdP, AdP_1, AdP_2 }









P = { $NP_1 \rightarrow \epsilon$, $NP_2 \rightarrow \epsilon$, $S_1 \rightarrow \epsilon$, $S_2 \rightarrow \epsilon$, $VP_1 \rightarrow \epsilon$,
$VP_2 \rightarrow \epsilon$, $AdP_1 \rightarrow \epsilon$, $AdP_2 \rightarrow \epsilon$,
NP $\rightarrow$ NP_1 Martin NP_2, S $\rightarrow$ S_1 NP VP S_2,
VP $\rightarrow$ VP_1 runs VP_2, $VP_2 \rightarrow VPVP_2$,
VP $\rightarrow$ AdP, $Adp \rightarrow AdP_1$ slowly }

Conversion of CFG to LTIG

- Step 1: Create the set of initial trees
- Step 2: Let the label of the root be A_i Modify the grammar of Step 1, so that every tree t is either:
  - left-anchored (i.e. has a terminal as its first nonempty node)
  - has a first nonempty frontier node labeled A_j where $i \leq j$
  - if $i > j$, substitute initial trees such that the first nonempty frontier node is labeled A_i
  - if $i = j$, convert the tree to an auxiliary tree
- Step 3: Modify the set of initial trees until every tree is left anchored.
- Step 4: Every unanchored auxiliary tree gets a lexical anchor (via substitution of the initial trees from the previuos step)

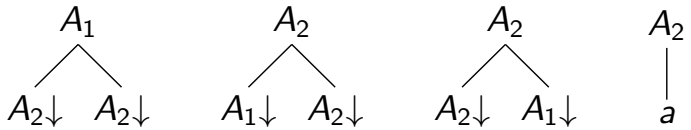
Conversion of CFG to LTIG (example)

CFG

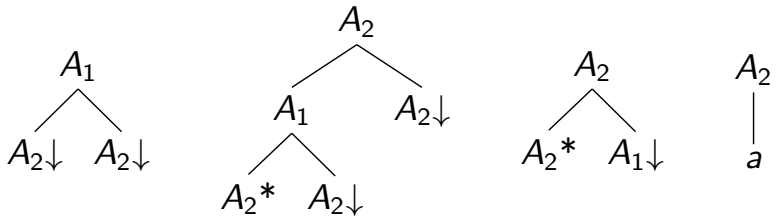
$A_1 \rightarrow A_2 A_2$

$A_2 \rightarrow A_1 A_2 | A_2 A_1 | a$

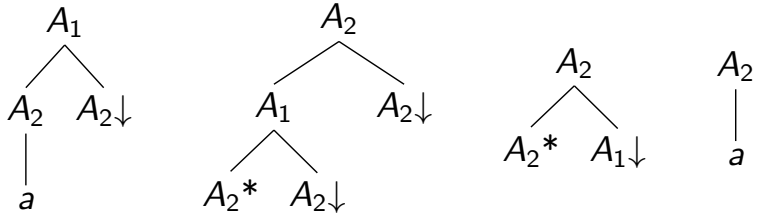
Step 1



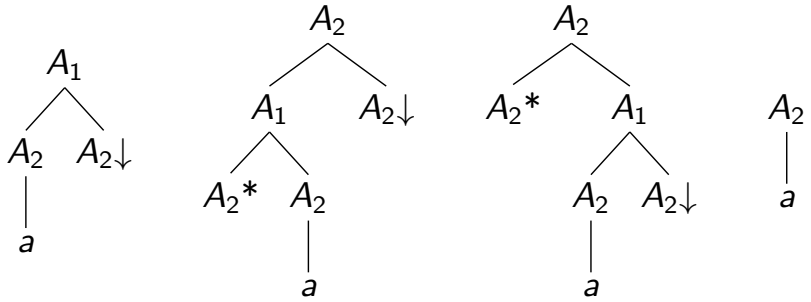
Step 2



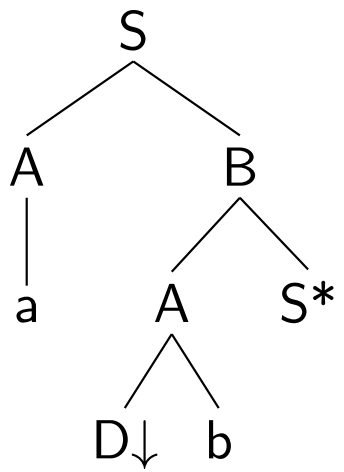
Step 3



Step 4



An auxiliary tree and its textual representation



$\mu_S^1 \rightarrow \mu_A^2 \mu_B^4$

$\mu_A^2 \rightarrow \mu_a^3$

$\mu_B^4 \rightarrow \mu_A^5 \mu_S^8$

$\mu_A^5 \rightarrow \mu_D^6 \mu_b^7$

$\text{LeftAux}(\mu_S^1)$

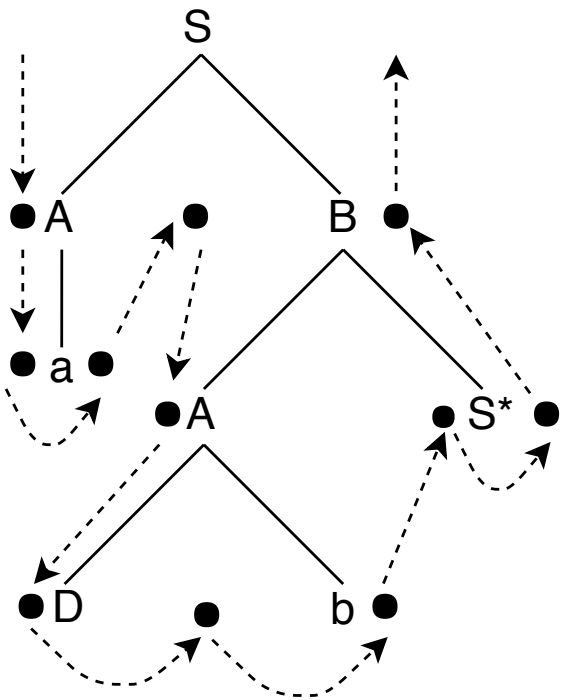
$\text{Subst}(\mu_D^6)$

$\text{Foot}(\mu_S^8)$

Earley parsing for TIGs

- The Earley-style TIG parser collects states into the chart C. A state is a 3-tuple, $[p, i, j]$ where:

  $\Rightarrow$ p is a position in an elementary tree and $0 \leq i \leq j \leq n$ are integers indicating a span of the input string.



Earley parsing for TIGs

Initialization: $\dfrac{}{[\mu_S \rightarrow \bullet\alpha, 0, 0]}$

PredictLeftAdjunction: $\dfrac{[\mu_A \rightarrow \bullet\alpha, i, j]}{[\rho_A \rightarrow \bullet\gamma, j, j]}$ $\mathit{LeftAux}(\rho_A)$, $\mathit{Adjoin}(\rho_A, \mu_A)$

LeftAdjunction: $\dfrac{[\mu_A \rightarrow \bullet\alpha, i, j][\rho_A \rightarrow \gamma\bullet, j, k]}{[\mu_A \rightarrow \bullet\alpha, i, k]}$ $\mathit{LeftAux}(\rho_A)$, $\mathit{Adjoin}(\rho_A, \mu_A)$

PredictRightAdjunction: $\dfrac{[\mu_A \rightarrow \alpha\bullet, i, j]}{[\rho_A \rightarrow \bullet\gamma, j, j]}$ $\mathit{RightAux}(\rho_A)$, $\mathit{Adjoin}(\rho_A, \mu_A)$

RightAdjunction: $\dfrac{[\mu_A \rightarrow \alpha\bullet, i, j][\rho_A \rightarrow \gamma\bullet, j, k]}{[\mu_A \rightarrow \alpha\bullet, i, k]}$ $\mathit{RightAux}(\rho_A)$, $\mathit{Adjoin}(\rho_A, \mu_A)$

Earley parsing for TIGs

Scan: $$\frac{[\mu_A \rightarrow \alpha \bullet v_a \beta, i, j]}{[\mu_A \rightarrow \alpha v_a \bullet \beta, i, j+1]} \quad a = a_{j+1}$$

EpsScan: $$\frac{[\mu_A \rightarrow \alpha \bullet v_a \beta, i, j]}{[\mu_A \rightarrow \alpha v_a \bullet \beta, i, j]} \quad a = \epsilon$$

ScanFoot: $$\frac{[\mu_A \rightarrow \alpha \bullet v_B \beta, i, j]}{[\mu_A \rightarrow \alpha v_B \bullet \beta, i, j]} \quad Foot(v_B)$$

Earley parsing for TIGs

PredictSubst: $$\frac{[\mu_A \rightarrow \alpha \bullet v_B \beta, i, j]}{[\rho_B \rightarrow \bullet \gamma, j, j]} \quad \begin{array}{l} Subst(v_B), \\ Init(\rho_B) \end{array}$$

Substitute: $$\frac{[\mu_A \rightarrow \alpha \bullet v_B \beta, i, j][\rho_B \rightarrow \gamma \bullet, j, k]}{[\mu_A \rightarrow \alpha v_B \bullet \beta, i, k]} \quad \begin{array}{l} Subst(v_B), \\ Init(\rho_B) \end{array}$$

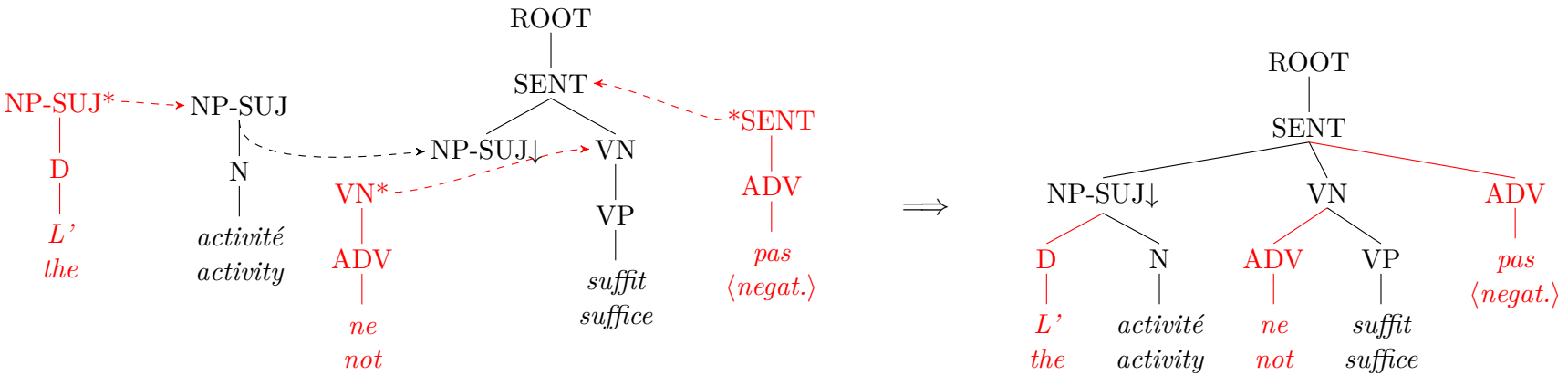
Earley parsing for TIGs

MoveDown: $$\frac{[\mu_A \rightarrow \alpha \bullet v_B \beta, i, j]}{[v_B \rightarrow \bullet \gamma, j, j]}$$

CompleteNode: $$\frac{[\mu_A \rightarrow \alpha \bullet v_B \beta, i, j][v_B \rightarrow \gamma \bullet, j, k]}{[\mu_A \rightarrow \alpha v_B \bullet \beta, i, k]}$$

Goal Item: $[\mu_S \rightarrow \alpha \bullet, 0, n], \mathit{Init}(\mu_S)$

Sister Adjunction



Data driven parsing: PCFG

- Grammar induction (with some preprocessing) from a treebank
- For all $A \rightarrow \alpha \in P$, the estimated probability $p(A \rightarrow \alpha)$ is

$$A \rightarrow \alpha \in P = \frac{count(A \rightarrow \alpha)}{count(A)}$$

- where $count(A \rightarrow \alpha)$ is the number of occurences of the production in the treebank and $count(A)$ the number of A-nodes in the treebank
- $\Rightarrow$ Maximum Likelihood Estimator

PCFG

$S \rightarrow NP\ VP\ 1.0$
$VP \rightarrow V\ 0.1$
$VP \rightarrow V\ NP\ 0.7$
$VP \rightarrow V\ NP\ NP\ 0.2$
$NP \rightarrow Det\ N\ 0.6$

$NP \rightarrow N\ 0.4$
$Det \rightarrow the\ 0.5$
$Det \rightarrow a\ 0.5$
$N \rightarrow cat\ 0.2$
$N \rightarrow dog\ 0.2$

$N \rightarrow man\ 0.3$
$N \rightarrow woman\ 0.3$
$V \rightarrow chased\ 0.8$
$V \rightarrow kissed\ 0.2$

A = 'the cat chased a dog'

$$P(A) = P(S) \times P(S \rightarrow NP\ VP|S) \times P(NP \rightarrow Det\ N\ |NP) \times P(VP \rightarrow V\ NP|VP) \times P(NP \rightarrow Det\ N\ |NP) \times P(Det \rightarrow the|Det) \times P(N \rightarrow cat|N\ ) \times P(V \rightarrow chased|V\ ) \times P(Det \rightarrow a|Det) \times P(N \rightarrow dog|N\ )$$
$$= 1.0 \times 1.0 \times 0.6 \times 0.7 \times 0.6 \times 0.5 \times 0.2 \times 0.8 \times 0.5 \times 0.2 = 0.002016$$

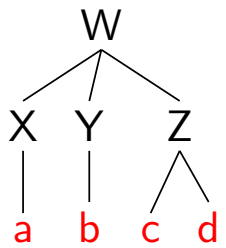
Evaluation

- In order to judge the performance of a parser, one must be able to assess the quality of its output (the parsed test data) with respect to the desired output (the gold data).
- The most widely used technique for this task consists of comparing for each parsed sentence the set of bracketings produced by the parser with the set of gold bracketings from the manual treebank annotation.
- A bracketing is a pair of indices on the input string denoting the start and the end of the span dominated by a certain non-terminal. The bracketing is called labeled if the label is included; if it is just the index pair, it is called unlabeled.

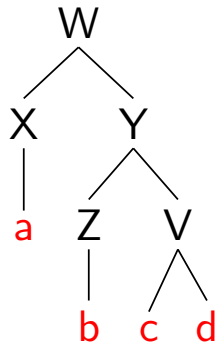
Evaluation

- Commonly, bracket scoring is defined as follows. Let O be the set of bracketings from the parser output, and let the set of bracketings from the treebank annotation be G.
  - ⋆ Precision (accuracy) is then computed as $\frac{|O \cap G|}{|O|}$,
  - ⋆ Recall (coverage) as $\frac{|O \cap G|}{|G|}$, and
  - ⋆ F-score F1 as $\frac{2 \times \mathit{precision} \times \mathit{recall}}{\mathit{precision} + \mathit{recall}}$.

Candidate parse:



Golden standard:



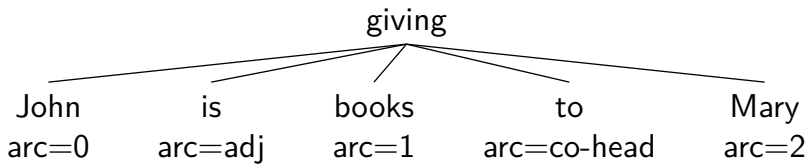
Candidate	Gold
$X \rightarrow^* a$	$X \rightarrow^* a$
$Y \rightarrow^* b$	$Z \rightarrow^* b$
$Z \rightarrow^* cd$	$V \rightarrow^* cd$
–	$Y \rightarrow^* bcd$
$W \rightarrow^* abcd$	$W \rightarrow^* abcd$

precision $= 4/4 = 1$
recall $= 4/5 = 0.8$
f-score $= 0.89$

labelled precision $= 2/4 = 0.5$
labelled recall $= 2/5 = 0.4$
labelled f-score $= 0.44$

MICA parser

- MICA (Marseille-INRIA-Columbia-AT&T) [BBN+09]
  http://mica.lif.univ-mrs.fr/
- Probabilistic dependency parser based on TIG
- Off-the-shelf parser: freely available, easy to install under Linux
- Earley-like parser (several optimizations are applied)
- Returns deep dependency parses



- MICA is based on LTIG extracted from the Penn Treebank
  - ⋆ ⇒ rich linguistic information is available (e.g voice, empty subjects, wh-movement, relative clauses etc.)
- state-of-the art performance (87,6% accuracy)

MICA parser

- Two processes:
  - supertagging (assignment of a sequence of elementary trees to the input word sequence), 4727 supertags from Penn Treebank
  - actual parser (derives syntactic structure from the n-best chosen supertags)
- MICA returns n-best parses for arbitrary n: parse trees are associated with probabilities
- MICA grammars are extracted in three steps:
  - ⋆ TIG extracted from Penn Treebank, along with a table of counts
  - ⋆ TIG and the table of counts are used to build a PCFG
  - ⋆ PCFG is "specialized" in order to model more finely some lexico-syntactic phenomena

[BBN+09]

MICA parser: output

ID	word	POS	Parent-ID	parent word	parent POS	supertag	parent-supertag	attributes
1	Time	NNP	2	flies	VBZ	t3	t81	cat:N dsubcat:nil ssubcat:nil pred:- comp:n root:NP lfront:nil rfront:nil intern:nil adjnodes:NP substnodes:nil DSub:nil SSub:nil DRIO:- SRIO:- DRole:0 SRole:0
2	flies	VBZ	2	flies	VBZ	t81	t81	cat:V dsubcat:NP0 ssubcat:NP0 voice:act comp:n root:S lfront:NPs0 rfront:nil intern:VP adjnodes:S VP substnodes:nil DSub:0 SSub:0 DRIO:0 SRIO:0 DRole:Root SRole:Root

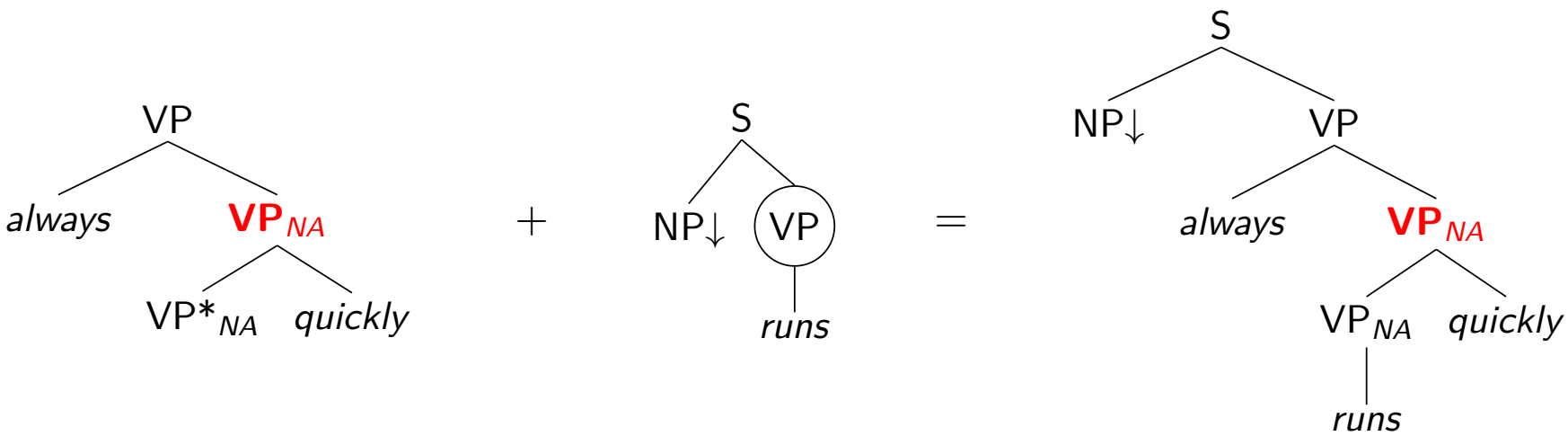
lp=-51.379112

Off-spine TAG (osTAG)

- Off-spine TAG (osTAG) is a context-free TAG variant [SYCS13]
- Linguistically motivated
- Generates context-free languages (as TIG)
- Normal TAG, just only one restriction with regard to adjunction:
  - ⋆ Adjunction is disallowed at any node on the spine of an auxiliary tree below the root
  - ⋆ Although relaxing this constraint is still possible (at the expense of complexity)

Off-spine TAG (osTAG)

⋆ In osTAG, adjunction is disallowed at the highlighted node:



osTAG to CFG

- For the pair of nodes η and η', the target nonterminal is noted as $\eta(\eta')$
- For each tree τ and each interior node η in τ that is not on the spine of τ, with children $\eta_1,\ldots,\eta_k$, add the following rule to the CFG:

  (1) $\eta \rightarrow \eta_1 \ldots \eta_k$
- If the interior node η is on the spine of τ (i.e. dominating the foot of τ) and η' is a node in a any tree where τ is adjoinable, and η_s is a child on the spine of the tree, add the following rule to the CFG:

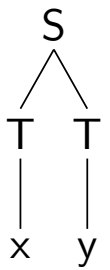
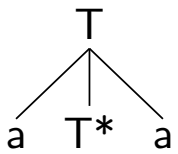
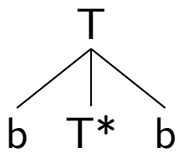
  (2) $\eta(\eta') \rightarrow \eta_1 \ldots \eta_s(\eta') \ldots \eta_k$
- To initiate adjunction at any node η' where a tree τ with root η is adjoinable, the following rule is used:

  (3) $\eta' \rightarrow \eta(\eta')$
- For the foot node η_f of τ:

  (4) $\eta_f(\eta) \rightarrow \eta$
- To handle substitution, any frontier node η which allows substitution of a tree rooted with η'

  (5) $\eta(\eta) \rightarrow \eta'$

osTAG to CFG

	osTAG	CFG	simplified CFG
α:		$\alpha_\epsilon \xrightarrow{1} \alpha_1\alpha_2$ $\alpha_1 \xrightarrow{1} x$ $\alpha_2 \xrightarrow{1} y$ $\alpha_1 \xrightarrow{3} \beta_\epsilon(\alpha_1)$ $\alpha_1 \xrightarrow{3} \gamma_\epsilon(\alpha_1)$ $\alpha_2 \xrightarrow{3} \beta_\epsilon(\alpha_2)$ $\alpha_2 \xrightarrow{3} \gamma_\epsilon(\alpha_2)$	$\alpha_\epsilon \xrightarrow{1} \alpha_1\alpha_2$ $\alpha_1 \xrightarrow{1} x$ $\alpha_2 \xrightarrow{1} y$
β:		$\beta_\epsilon(\alpha_1) \xrightarrow{2} a\beta_2(\alpha_1)a$ $\beta_\epsilon(\alpha_2) \xrightarrow{2} a\beta_2(\alpha_2)a$ $\beta_2(\alpha_1) \xrightarrow{4} \alpha_1$ $\beta_2(\alpha_2) \xrightarrow{4} \alpha_2$	$\alpha_1 \xrightarrow{2} a\alpha_1a$ $\alpha_2 \xrightarrow{2} a\alpha_2a$
γ:		$\gamma_\epsilon(\alpha_1) \xrightarrow{2} b\gamma_2(\alpha_1)b$ $\gamma_\epsilon(\alpha_2) \xrightarrow{2} b\gamma_2(\alpha_2)b$ $\gamma_2(\alpha_1) \xrightarrow{4} \alpha_1$ $\gamma_2(\alpha_2) \xrightarrow{4} \alpha_2$	$\alpha_1 \xrightarrow{2} b\alpha_1b$ $\alpha_2 \xrightarrow{2} b\alpha_2b$

Possible extensions of osTAG

- Instead of allowing zero adjunction on the spine of auxiliary trees, any non-zero bound would limit generative capacity
  - ⋆ Tradeoff: higher complexity ($\mathcal{O}(n^{k+2})$ for every k level of spine adjunction)
- Make OSTAG consistent with TIG constraints (no increasing in complexity)

Experiments with osTAG: evaluation

- Parsing F-Score for different models (full test set and sentences of length 40 or less)

	All	40	Adjunctions	Wrap. adjunctions
TSG	85.00	86.08	-	-
osTAG^1	85.42	86.43	1336	52
osTAG^2	85.54	86.56	1952	44
osTAG^3	85.86	86.84	3585	41

References I

[BBN+09] Srinivas Bangalore, Pierre Boulllier, Alexis Nasr, Owen Rambow, and Benoît Sagot.
Mica: A probabilistic dependency parser based on tree insertion grammars application note.
In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, NAACL-Short '09, pages 185–188, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[Chi00] David Chiang.
Statistical parsing with an automatically-extracted tree adjoining grammar.
In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 456–463. Association for Computational Linguistics, 2000.

[Gre65] Sheila A Greibach.
A new normal-form theorem for context-free phrase structure grammars.
Journal of the ACM (JACM), 12(1):42–52, 1965.

[SW95] Yves Schabes and Richard C Waters.
Tree insertion grammar: a cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced.
Computational Linguistics, 21(4), 1995.

[SYCS13] Ben Swanson, Elif Yamangil, Eugene Charniak, and Stuart Shieber.
A context free tag variant.
In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 302–310, 2013.