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

Laura Kallmeyer
Heinrich-Heine-Universität Düsseldorf

slides partly done by Tatiana Bladier

Wintersemester 2021

Overview

1. Introduction
2. Tree Insertion Grammar (TIG)
3. Earley parsing for TIGs
4. Off-spine TAG (osTAG)

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]
    $A \rightarrow a$
    $A \rightarrow aA_1 \dots A_n$
- Very large output grammars $\Rightarrow$ awkward or impossible to use

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

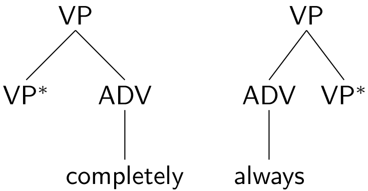
Tree Insertion Grammar (TIG): motivation

- Lexicalized CFGs allow parsing in cubic time
- Conversion to lexicalized CFGs ⇒ *weak* lexicalization
  - ⋆ the strings are preserved
  - ⋆ derived trees are not preserved ⇒ 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

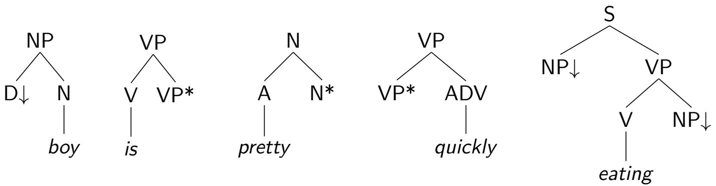
- A *right* (resp. *left*) auxiliary tree is an auxiliary tree with no leaves to the left (resp. right) of the foot node.



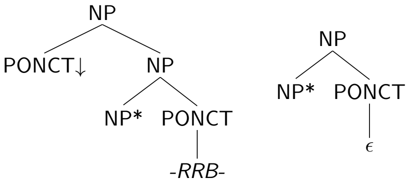
- A **Tree insertion grammar (TIG)** [SW95] is then defined as a TAG where all auxiliary trees are either right or left auxiliary trees and have at least one lexical node.
- Substitution is the same as in TAG.
- Adjunction is restricted compared to TAG.
  ⇒ TIGs derive only context-free languages

Tree Insertion Grammar: Restrictions on adjunction

- Only non-empty *right auxiliary trees* or *left auxiliary trees* are allowed (see above)

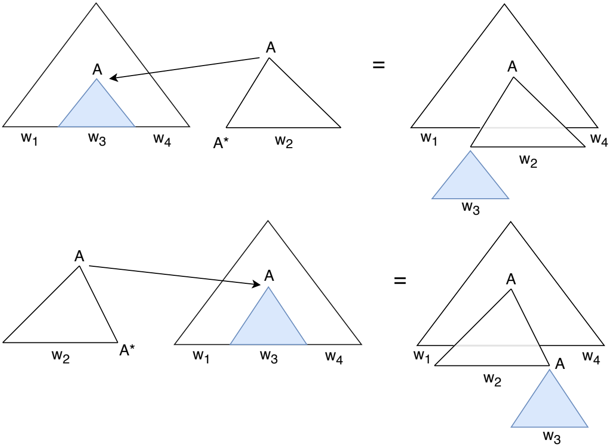


- *Wrapping auxiliary trees* and *empty auxiliary trees* are forbidden



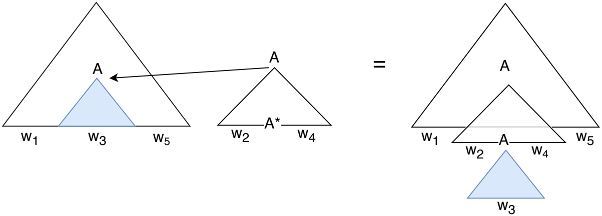
Right- and left-adjunction

Even with derived auxiliary trees, only right adjunction and left adjunction is allowed:



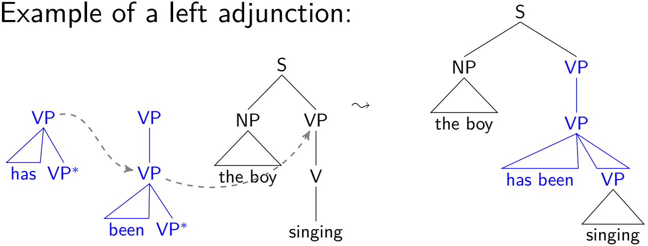
Wrapping adjunction

Wrapping adjunction is not allowed:

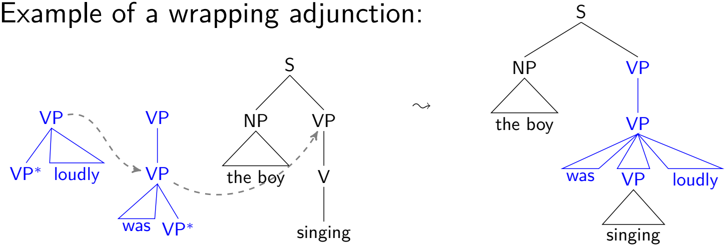


Right/left versus Wrapping adjunction

Example of a left adjunction:

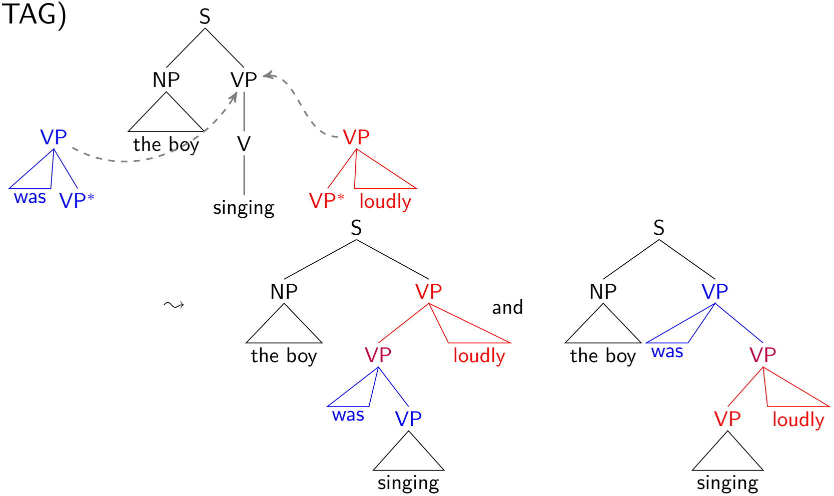


Example of a wrapping adjunction:



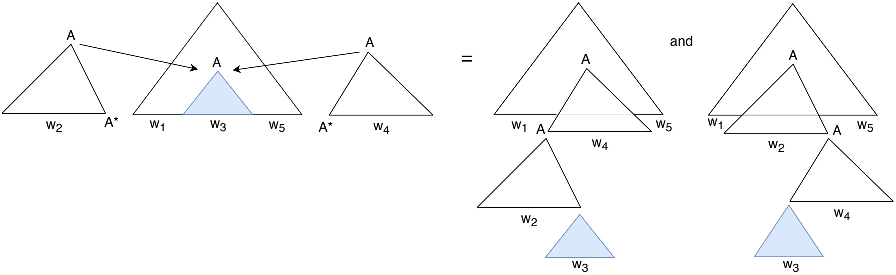
Tree Insertion Grammar: Restrictions on adjunction

- Simultaneous multiple adjunction is allowed (non-standard in TAG)



Tree Insertion Grammar: Restrictions on adjunction

- TIG allows multiple simultaneous adjunction on a single node
- Simultaneous adjunction $\neq$ wrapping adjunction; strings are adjoined independently

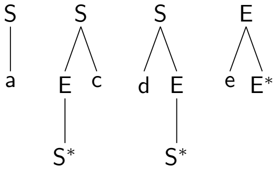


Tree Insertion Grammar: Restrictions on adjunction

- At the following nodes, adjunction is not allowed: substitution nodes, foot nodes, roots of auxiliary trees.
- A left (resp. right) auxiliary tree is not allowed to be adjoined to the nodes on the spine of a right (resp. left) auxiliary tree

Tree Insertion Grammar: Restrictions on adjunction

Example:



- Language generated if grammar is taken to be a TAG (assuming NA at foot nodes)?

$$L((de^*)^* ac^* | e^+ (de^*)^* ac^* c)$$

- Language generated if grammar is taken to be a TIG?

$$L((de^*)^* ac^*)$$

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 ⇒ 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

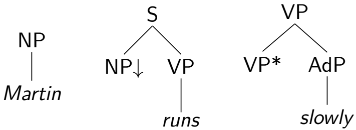
Conversion of TIG to CFG (1) [SW95]

- 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: $Y_i \to \epsilon$, $Z_i \to \epsilon$.
- 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.
- T' = T, S' = S

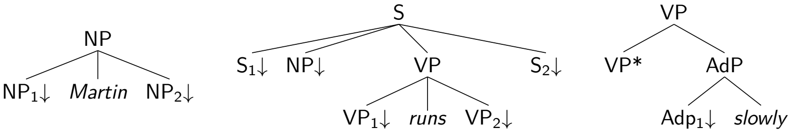
Conversion of TIG to CFG (2)

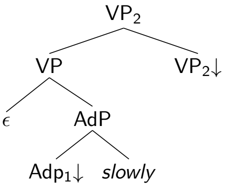
- Step 4: Convert every auxiliary tree t in A as follows:
  - let A_i be the label of the root μ of t.
  - If t is a left auxiliary tree, add a new root labeled Y_i with two children: μ on the left, and on the right, a node labeled Y_i and marked for substitution.
  - If t is a right auxiliary tree, 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 $\}$





$P = \{$ $NP_1 \to \epsilon$, $NP_2 \to \epsilon$, $S_1 \to \epsilon$, $S_2 \to \epsilon$, $VP_1 \to \epsilon$, $VP_2 \to \epsilon$, $AdP_1 \to \epsilon$, $AdP_2 \to \epsilon$,
$NP \to NP_1$ Martin NP_2, $S \to S_1$ NP VP_1 runs VP_2 S_2,
$VP_2 \to Adp_1$ slowly $VP_2$$\}$

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 $i \leq j$

  To this end, we do the following:
  - 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, via substitution of lexicalized trees.
- Step 4: Every unanchored auxiliary tree gets a lexical anchor (via substitution of the initial trees from the previous step)

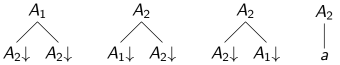
Conversion of CFG to LTIG (example)

CFG

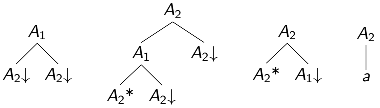
$A_1 \rightarrow A_2 A_2$

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

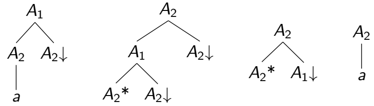
Step 1



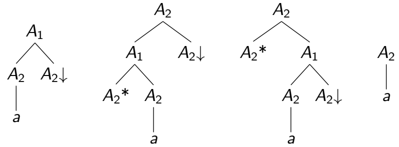
Step 2



Step 3



Step 4

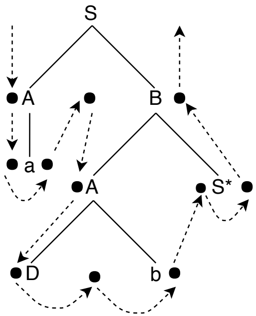


Earley parsing for TIGs

- We use dotted productions $v \to v_1 \dots v_i \bullet v_{i+1} \dots v_k$ as for TAG parsing.
- Let *Adjoin* be a boolean predicate; $Adjoin(v_r, v)$ indicates whether the auxiliary tree with root node v_r can be adjoined at node v.
- Let *LeftAux* be a boolean predicate; $LeftAux(v_r)$ indicates whether the tree with root v_r is a left auxiliary tree. *RightAux* is defined accordingly for right auxiliary trees.
- *Foot*, *Subst*, *Init* are boolean predicates indicating whether a node is a foot node, a substitution node or the root of an initial tree respectively.
- Parse items have the form $[v \to v_1 \dots v_i \bullet v_{i+1} \dots v_k, i, j]$ where: $v \to v_1 \dots v_i \bullet v_{i+1} \dots v_k$ is a dotted production, and $0 \leq i \leq j \leq n$ indicate the already recognized span.

Earley parsing for TIGs

The Earley parser traverses the trees as follows:



In contrast to TAG:

- no need for a top/bottom distinction; multiple adjunction is allowed
- only two indices needed, spans cannot be discontinuous

Earley parsing for TIGs

Initialization: $$\frac{}{[v \rightarrow \bullet\alpha, 0, 0]} \quad \mathit{Init}(v)$$ [1]

PredictLeftAdjunction: $$\frac{[v \rightarrow \bullet\alpha, i, j]}{[v_r \rightarrow \bullet\gamma, j, j]} \quad \mathit{LeftAux}(v_r), \mathit{Adjoin}(v_r, v)$$

LeftAdjunction: $$\frac{[v \rightarrow \bullet\alpha, i, j][v_r \rightarrow \gamma\bullet, j, k]}{[v \rightarrow \bullet\alpha, i, k]} \quad \mathit{LeftAux}(v_r), \mathit{Adjoin}(v_r, v)$$

PredictRightAdjunction: $$\frac{[v \rightarrow \alpha\bullet, i, j]}{[v_r \rightarrow \bullet\gamma, j, j]} \quad \mathit{RightAux}(v_r), \mathit{Adjoin}(v_r, v)$$

RightAdjunction: $$\frac{[v \rightarrow \alpha\bullet, i, j][v_r \rightarrow \gamma\bullet, j, k]}{[v \rightarrow \alpha\bullet, i, k]} \quad \mathit{RightAux}(v_r), \mathit{Adjoin}(v_r, v)$$

[1] We don't assume that the derived tree must have a root label S.

Earley parsing for TIGs

Scan: $$\frac{[v \to \alpha \bullet v'\beta, i, j]}{[v \to \alpha v' \bullet \beta, i, j+1]} \quad l(v') = w_{j+1}$$

EpsScan: $$\frac{[v \to \alpha \bullet v'\beta, i, j]}{[v \to \alpha v' \bullet \beta, i, j]} \quad l(v') = \varepsilon$$

ScanFoot: $$\frac{[v \to \alpha \bullet v'\beta, i, j]}{[v \to \alpha v' \bullet \beta, i, j]} \quad Foot(v')$$

Earley parsing for TIGs

PredictSubst: $$\frac{[v \to \alpha \bullet v'\beta, i, j]}{[v_r \to \bullet\gamma, j, j]} \quad Subst(v'), Init(v_r), \; l(v') = l(v_r)$$

Substitute: $$\frac{[v \to \alpha \bullet v'\beta, i, j][v_r \to \gamma\bullet, j, k]}{[v \to \alpha v' \bullet \beta, i, k]} \quad Subst(v'), Init(v_r), \; l(v') = l(v_r)$$

Earley parsing for TIGs

MoveDown: $\frac{[v \rightarrow \alpha \bullet v'\beta, i, j]}{[v' \rightarrow \bullet\gamma, j, j]}$ $v' \rightarrow \gamma$ is a rule, i.e., a subtree

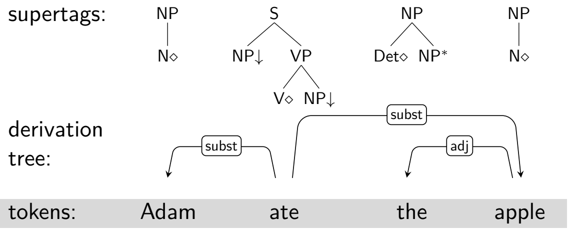
CompleteNode: $\frac{[v \rightarrow \alpha \bullet v'\beta, i, j][v' \rightarrow \gamma\bullet, j, k]}{[v \rightarrow \alpha v' \bullet \beta, i, k]}$

Goal Items: $[v \rightarrow \alpha\bullet, 0, n], Init(v)$

Data driven parsing for TAG: overall architecture

Data-driven TAG/TIG parsing works as follows [BBN+09, BWKJ19]:

- Training data: sentences with supertag information and derivation tree edges.



- Formally, the derivation tree is a dependency tree.
  ⇒ Data-driven TAG/TIG parsing consists of
  1. supertagging (a sequence labeling task);
  2. dependency parsing;
  3. computation of derived tree.

Reminder: PCFG

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

$$P(A \to \alpha | A) = \frac{count(A \to \alpha)}{count(A)}$$

- where $count(A \to \alpha)$ 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

Reminder: 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).
- For constituency trees, we usually compare for each parsed sentence the set of bracketings produced by the parser with the set of gold bracketings.
- 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 (e.g., $[\mathrm{NP}, 3, 5]$), otherwise (e.g., $[3, 5]$) it is called unlabeled.

Evaluation

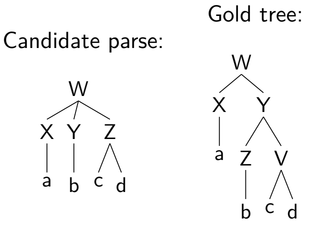
Commonly, bracket scoring is defined as follows. Let O be the set of bracketings from the parser output, and G the gold bracketings.

Evaluation metrics are precision P ("how many of the bracketings we found are correct?") recall R ("how many of the gold bracketings did we find?") and F-score $F1$:

$$P = \frac{|O \cap G|}{|O|} \qquad R = \frac{|O \cap G|}{|G|} \qquad F1 = \frac{2 \cdot P \cdot R}{P + R}$$

Evaluation

Example:



Cand. bracketings: [W, 0, 4], [X, 0, 1], [Y, 1, 2], [Z, 2, 4]
Gold bracketings: [W, 0, 4], [X, 0, 1], [Y, 1, 4], [Z, 1, 2], [V, 2, 4]

unlabelled $P = 4/4 = 1$ labelled $P = 2/4 = 0.5$
unlabelled $R = 4/5 = 0.8$ labelled $R = 2/5 = 0.4$
unlabelled $F1 = 0.89$ labelled $F1 = 0.44$

Evaluation

For TAG parsing (provided we have gold data), in addition, we can measure

- supertagging accuracy:

$$\frac{\text{number of correct cand. supertags}}{\text{number of gold supertags}}$$

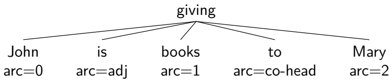
- dependency parsing (= derivation tree) accuracy:

$$\frac{\text{number of correct cand. derivation edges}}{\text{number of gold derivation edges}}$$

The latter can be measured labelled and unlabelled.

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% unlabeled dep. tree 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

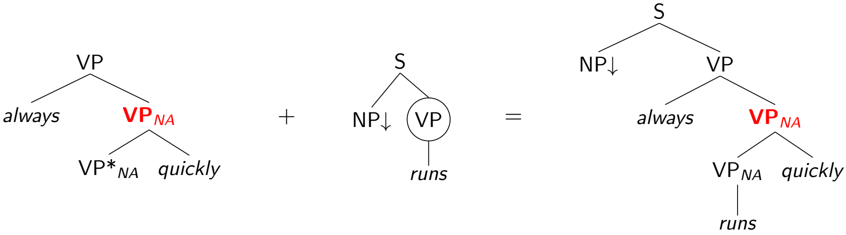
[BBN+09]

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, but restricted 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 initial tree τ and each interior node η in τ with children $\eta_1, \ldots, \eta_k$, add the following rule to the CFG:

  (1) $\eta \to \eta_1 \ldots \eta_k$

- If the interior node η is on the spine of an auxiliary tree 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) $\eta(\eta') \to \eta_1 \ldots \eta_s(\eta') \ldots \eta_k$

- To initiate adjunction at any node η' where a tree τ with root η is adjoinable, the following rule is used:

  (3) $\eta' \to \eta(\eta')$

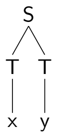
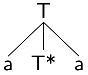
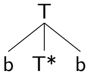
- For the foot node η_f of τ:

  (4) $\eta_f(\eta) \to \eta$

- To handle substitution, any frontier node η which allows substitution of a tree rooted with η'

  (5) $\eta(\eta) \to \eta'$

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:		$\beta_\epsilon(\alpha_1) \xrightarrow{2} \mathsf{a}\beta_2(\alpha_1)\mathsf{a}$ $\beta_\epsilon(\alpha_2) \xrightarrow{2} \mathsf{a}\beta_2(\alpha_2)\mathsf{a}$ $\beta_2(\alpha_1) \xrightarrow{4} \alpha_1$ $\beta_2(\alpha_2) \xrightarrow{4} \alpha_2$	$\alpha_1 \xrightarrow{2} \mathsf{a}\alpha_1\mathsf{a}$ $\alpha_2 \xrightarrow{2} \mathsf{a}\alpha_2\mathsf{a}$
γ:		$\gamma_\epsilon(\alpha_1) \xrightarrow{2} \mathsf{b}\gamma_2(\alpha_1)\mathsf{b}$ $\gamma_\epsilon(\alpha_2) \xrightarrow{2} \mathsf{b}\gamma_2(\alpha_2)\mathsf{b}$ $\gamma_2(\alpha_1) \xrightarrow{4} \alpha_1$ $\gamma_2(\alpha_2) \xrightarrow{4} \alpha_2$	$\alpha_1 \xrightarrow{2} \mathsf{b}\alpha_1\mathsf{b}$ $\alpha_2 \xrightarrow{2} \mathsf{b}\alpha_2\mathsf{b}$

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

[SYCS13]

- Coarse-to-fine: first use a PCFG for parsing, then feed all parses above some threshold probability into the osTAG chart. The three parsing models below take less (OSTAG^1) or more context (OSTAG^3) into account in the probabilistic model of adjunction at a node.
- TSG is a baseline Tree Substitution Grammar model (no adjunction, only substitution).
- Parsing F-Score for different models (full test set and sentences of length 40 or less)

	All	40	# adj. (all)	# Wrap. adj. (all)
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 Boullier, 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*, pages 185–188, Boulder, Colorado, June 2009. Association for Computational Linguistics.

[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.

[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.