Parsing Beyond Context-Free Grammars: TAG: Grammar induction

Laura Kallmeyer & Tatiana Bladier
Heinrich-Heine-Universität Düsseldorf

Sommersemester 2018

Overview

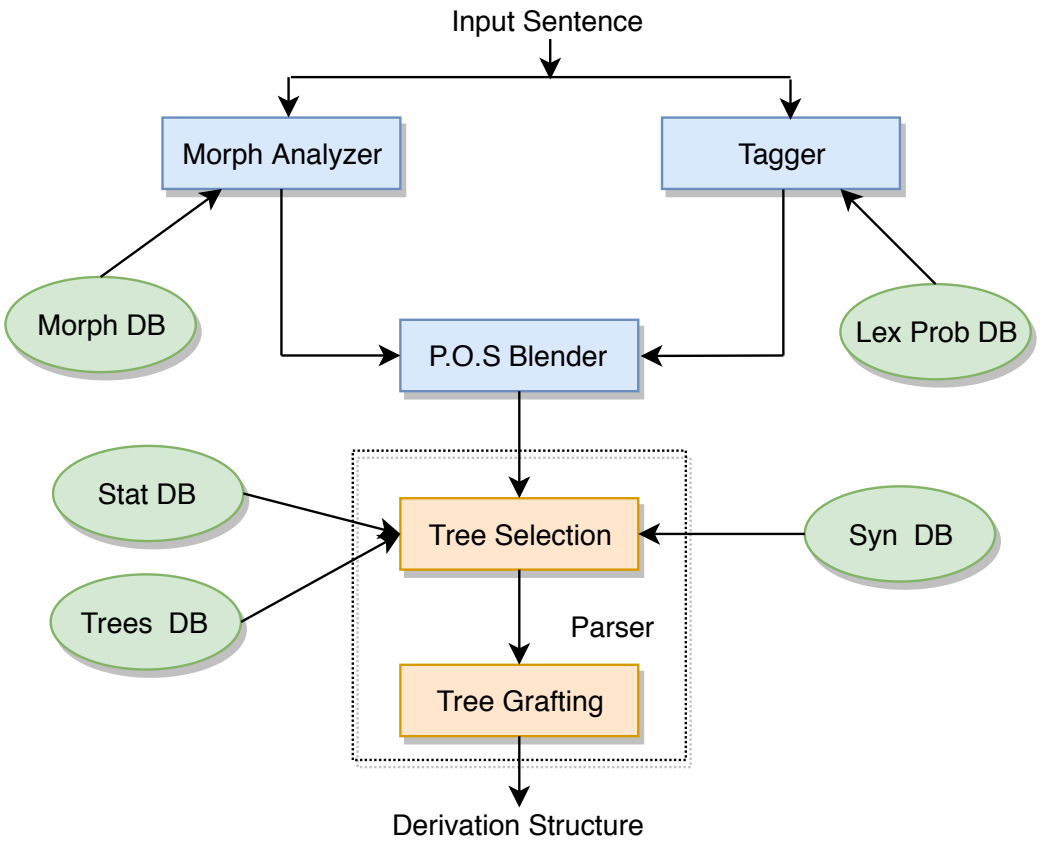
1. Hand-crafted TAGs
2. Annotated text corpora
3. TAG induction: Top-down approach
4. TAG induction: Bottom-up approach

TAG induction: motivation

- Extracting TAGs for probabilistic parsing
- TAGs for natural languages have been used for various tasks:
  - Parsing (Srinivas 1997, Lichte 2015, Waszczuk 2017)
  - Machine translation (Palmer 1998, Kurariya 2015)
  - Discourse Parsing (Forbes 2003)
  - Text generation (Stone and Doran 1997, Coran 1992, Danlos 2000)
- A wide-coverage TAG for a natural language contains several thousands of trees and takes a lot of effort to build
- Examples: hand-crafted TAGs for English (XTAG project [G$^+$98]) or French (FTAG)
- Growing amount of large-scale constituency treebanks for different languages (English, German, French, Spanish, Vietnamese, Korean etc.)
- Idea: automatic induction of linguistically plausible TAGs from syntactically annotated corpora
- Orientation on already existing hand-crafted LTAGs

XTAG: LTAG implementation for English

- large coverage LTAG for English
- implemented (2001) by the XTAG Research Group at the University of Pennsylvania
- architecture [G+98]:



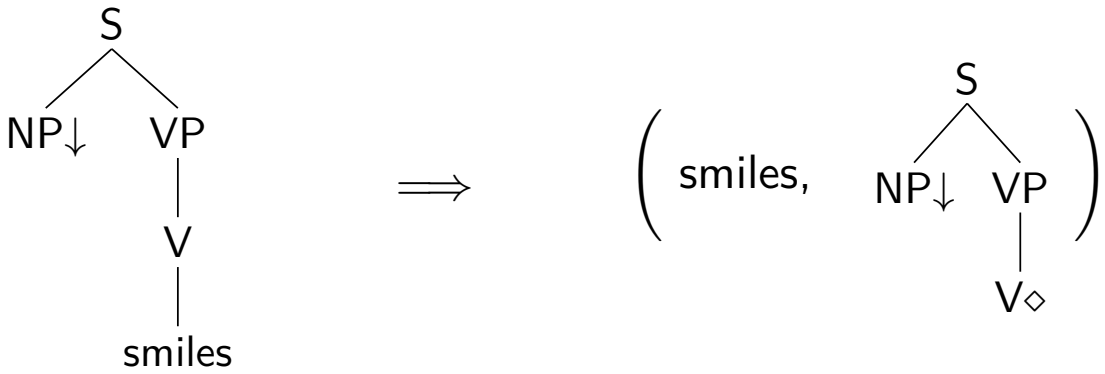
XTAG: facts

- Morphological Analyzer and MorphDB
  - ≈317000 inflected items derived from over 90000 stems
  - returns: root form + POS + inflectional information
- POS Tagger and Lex Prob DB
  - Wall-Street Journal trained trigram tagger
  - extended to output N-best POS sequences
- Syntactic DB
  - more than 30.000 entries, each consisting of:
    - ⋆ uninflected form of the word + POS
    - ⋆ list of trees or tree-families associated with the word
    - ⋆ list of feature equations that capture lexical idiosyncrasies
- Tree DB
  - 1004 trees divided into 53 families and 221 individual trees
- Tree Selection
  - for each word, tree templates are chosen from the Tree Database and the anchor position is filled with the word
- Tree Grafting
  - once a particular lexicalized tree set is schosen for a sentence, parsing is done $\Longrightarrow$ output: derived (parse) tree and derivation tree

[G+98]

XTAG

- XTAG: grammar implementation with TAG
  https://www.cis.upenn.edu/ xtag/
- XTAG *tree families*: grouping together trees that belong to the same subcategorization frame
- in each tree family:
  - base tree template
  - tree templates obtained from the base tree by transformations (meta-rules)
  - semantic interpretation of the arguments remain constant



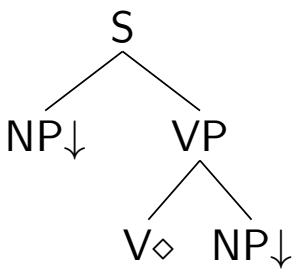
XTAG: tree families

- tree families for verb classes:
  - transitive verbs, e.g. *buy [_NP]*
    Joe bought a book.
  - ditransitive verbs, e.g. *buy [_NP _NP]*
    Joe bought Rose a book.
  - ditransitive verbs with PP, e.g. *put [_NP _PP]*
    Joe put the book on the shelf.
  - verbs with sentential complement, e.g. *tell [_NP _S]*
    Joe told Rose that Tom lives in Amsterdam.
  - light verbs, e.g. 'make comment'
    Joe made a comment on my book.
  - ...

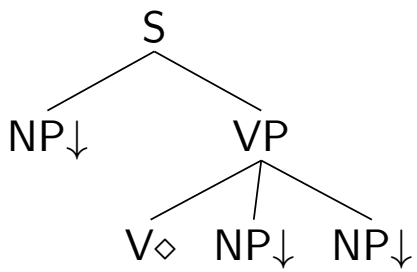
XTAG: tree families

- transitive / ditransitive / light verbs

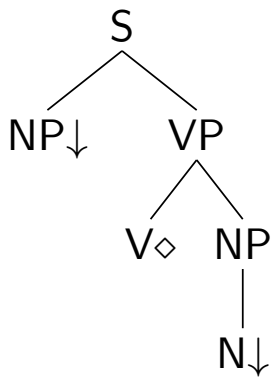
αNX0VNX1



αNX0VNX2NX1



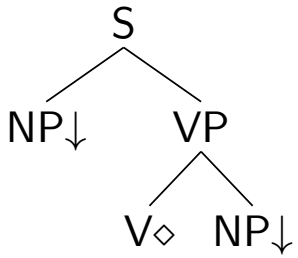
αNX0LVN1



- X◇: lexical anchor
- lexical insertion from the lexicon

XTAG: tree families

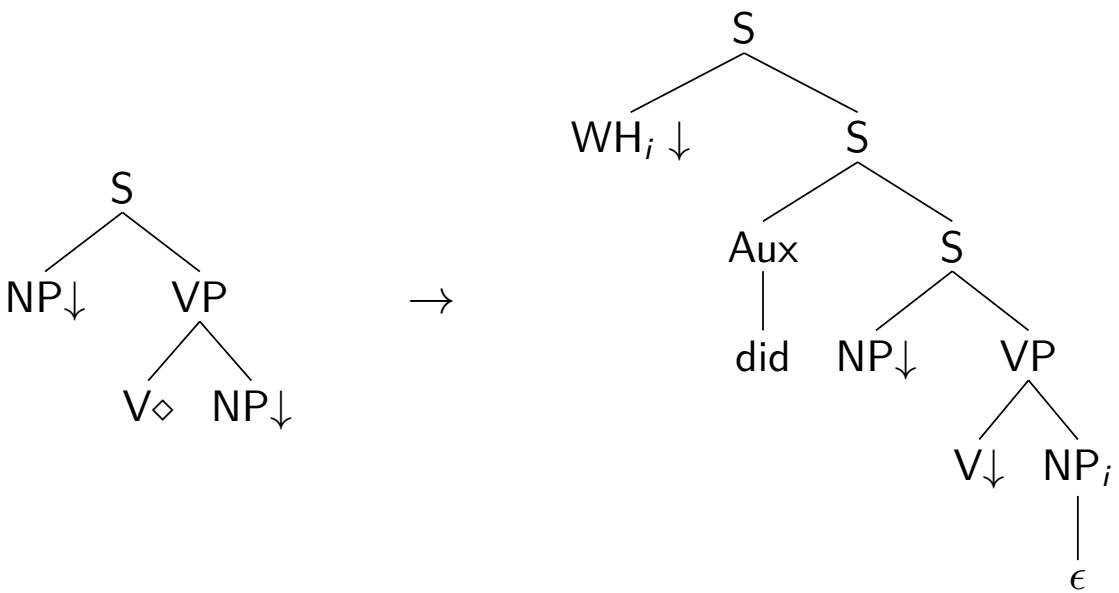
- tree family of *buy [_NP]* for
  - John bought a book.
    the 'base tree' (αNX0VNX1)



  - *Who bought a book? / What does John buy?* (wh-extraction)
  - *A book was bought by John.* (passivization)
  - *A book, Joe bought.* (topicalization)
- elementary trees derived from the base tree by transformation
- deriving *tree templates*

XTAG: tree families

- elementary trees derived from the base tree by transformation
- e.g. base tree $\rightarrow$ object extraction (wh-extraction)



XTAG: tree families

- transform base trees while changing the properties of particular words $\rightarrow$ derives a different tree family
- for example: ergative verbs as *melt*

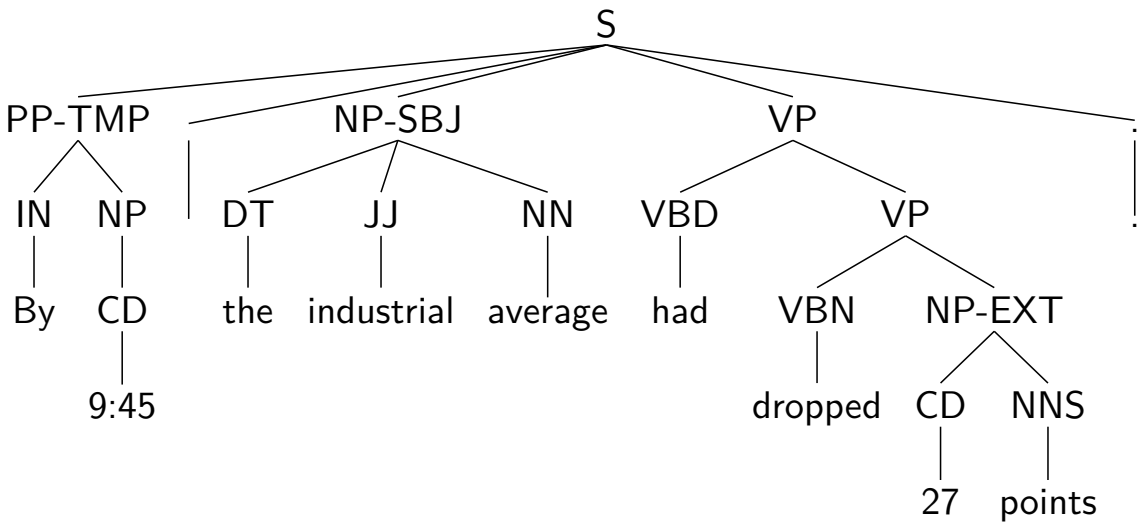
  The sun melted the ice. $\longrightarrow$ *The ice melted.*
- XTAG covers the following phenomena:
  - wh-extraction, topicalization
  - relative clauses, raising and small clause constructions
  - clausal adjuncts and imperatives
  - it-clefts, wh-clefts
  - auxiliaries, copula, infinitives, gerunds, passives, adjuncts
  - PRO constructions, noun-noun modifications, determiner sequences
  - genitives, negation, noun-verb contractions

Syntactically annotated text corpora: treebanks

- Hand-annotated (i.e. parsed and manually corrected) corpora are used for supervised and semi-supervised training of algorithms
- Constituency based corpora: Penn Treebank (English), NeGra German Corpus, TiGer treebank, French Treebank, Korean Treebank etc.
- Different formats: bracketed sentences, XML, NeGra

Penn Treebank

- Syntactically annotated corpus, constituency treebank
- Created between 1989 and 1996
- Several genres, including IBM computer manuals, nursing notes, Wall Street Journal articles, and transcribed telephone conversations.



- Bracketed notation:

  (S (PP-TMP (IN By) (NP (CD 9:45))) (, ,) (NP-SBJ (DT the) (JJ industrial) (NN average)) (VP (VBD had) (VP (VBN dropped) (NP-EXT (CD 27) (NNS points)))))

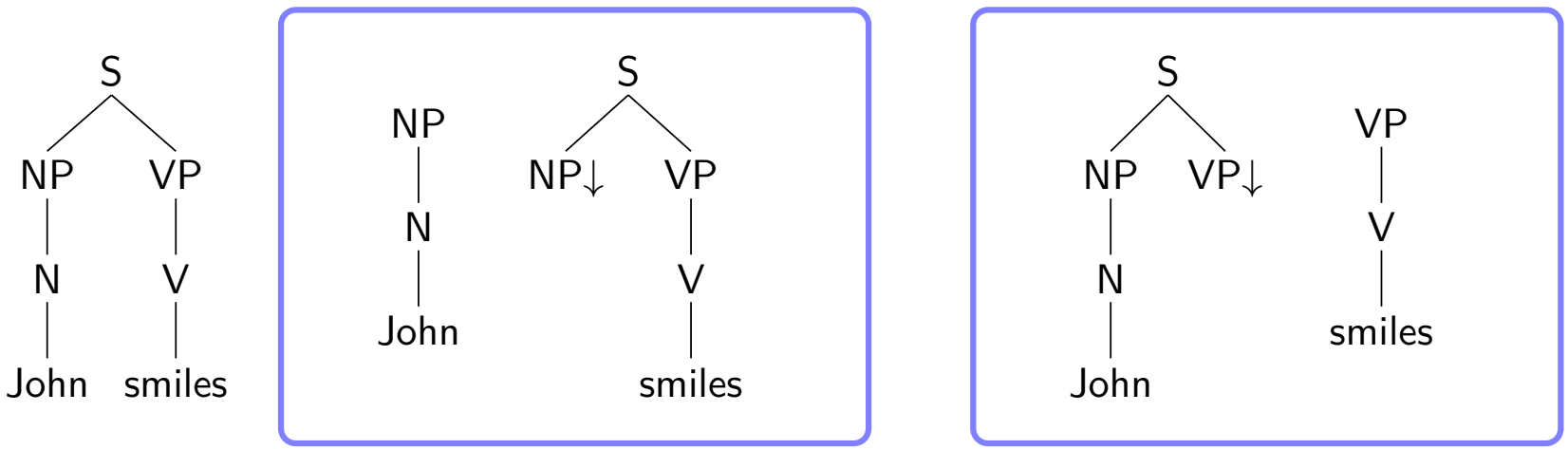
TAG induction: idea

- Sentences in treebanks are seen as derived TAG trees
- These derived TAG trees are cut in smaller pieces which correspond to elementary trees in TAG
- Treebanks provide additional information which can be used as features
- While extracting elementary trees we can also learn probabilities about trees (for example, statistics about frequencies of elementary trees)
- Probabilistic TAGs
- **Reversibility**

[XPJ00]

TAG induction: idea

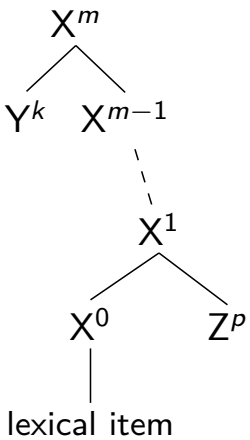
- Different kinds of TAGs can be induced from the same Treebank
- Annotated trees can be cut to elementary trees in different ways
- The choice of how the elementary trees should look like depends on the grammar. If there is a hand-crafted TAG, it can be used as orientation.
- However, there are certain linguistic principles to follow:
  - Dependencies (including long-distance dependencies) should be localized in the same elementary tree (i.e. head and its complements are included into elementary tree)
  - Recursion is factored into separate auxiliary trees (i.e. modifiers are excluded)
  - No elementary tree is anchored only by an empty element

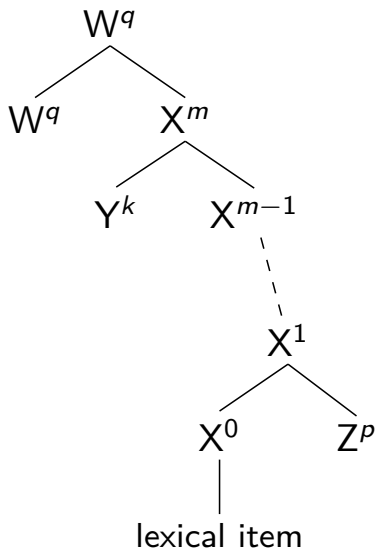


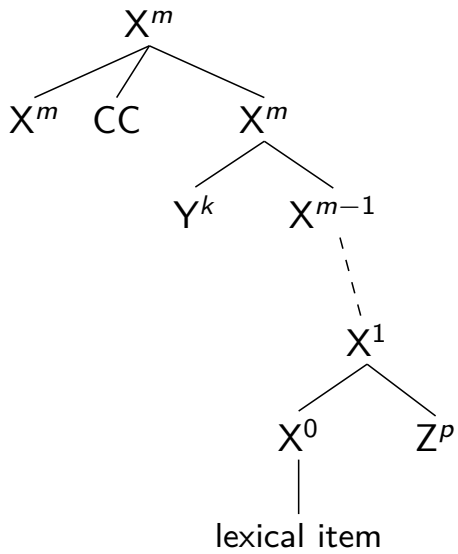
[XPJ00]

Prototypes of elementary trees

- Predicate-argument relation (head and its arguments)
- Modification relation (modifiers of the head)
- Coordinated relation (constructions with conjunctions)







LTAG extraction algorithm

- Convert a treebank tree into a derived tree in LTAG
  - labels come from the treebank, but also can be added or modified
- Mark children in the tree as being head, complement or modifier using heuristic percolation tables
- Decompose converted tree from the treebank into elementary trees
  - elementary trees contain the corresponding lexical anchor and the branches represent a particular syntactic context of a construction with slots for its complements
  - such elementary trees can be seen as supertags and be used for supertagging
- (optional step) create derivation trees based on the addresses of the children in the tree
  - these derivation trees show how extracted elementary trees can be combined to form a derived tree

[XP06]

Percolation tables for heads and modifiers

- Percolation tables are used to mark the roles of the children in the tree
  - ⋆ Head child
  - ⋆ Syntactically obligatory child (a complement)
  - ⋆ Modifier
- Needed to extract linguistically plausible LTAGs
- Deterministic extraction procedure (only one LTAG is possible)
- Language dependent and treebank dependent
- A head can have several arguments, and its projections can be modified by other components.
  - ⋆ e.g. a verbal phrase (VP) can have a verb (V) as its head and nominal phrase (NP) as its argument
  - ⋆ a VP can be modified by adverbial phrases (APs), prepositional phrases (PPs) etc.

[XP06]

Percolation tables for heads and modifiers

- Head rules for PTB (WSJ) based on Collins (1999 p. 240) (the list is not exhaustive)

parent node	search direction	head candidates
adjp	left-to-right	nns qp nn $ advp jj vbn vbg adjp jjr np jjs dt
conjp	right-to-left	cc rb in
nac	left-to-right	nn nns nnp nnps np nac ex $ cd qp prp vbg jj jjs jjr
vp	left-to-right	to vbd vbn md vbz vb vbg vbp vp adjp nn nns np

- Modifier rules for PTB (WSJ) (excerpt)

parent node	modifier nodes
s	pp pp-loc adv advp s-mnr
np	cc vbg jj jjs adjp nn np
vp	np jj jjs adjp pp-loc s-mnr

- All other nodes (non-leafs!) are marked as complements

TAG induction (top down): algorithm, [Xia01]

1. Step 1: Fully bracket the trees from the treebank
2. Step 2: Extract elementary trees
3. Step 3: Filter out invalid trees

The following slides present a slightly simplified algorithm presented in [Xia01] (i.e. simplification concerns no special treatment for coordination consituents, which are instead treated as regular modifiers)

Step 1: Fully bracketing the treebank trees, [Xia01]

→ We start from the root R of the tree in the treebank and choose the head-child hc for each node according to a head percolation table:

a) **Input:** a node N, the head percolation table HeadTb
Output: head-child hc of N
Definition: syn-tag(N) is the syntactic category of the node N
Algorithm: ttree node* FindHeadChild(N, HeadTb)
/* choose the head by the function tag */
(A) if (one child of N has a function tag that always marks a head)
then choose that child as the head-child hc ; return hc ;
/* choose the head by the head percolation table */
(B) x = syn-tag(N);
(C) Find the entry (x dir $y_1/y_2/\ldots/y_n$) in HeadTb ;
(D) for (each child ch of N , starting from the leftmost child or rightmost child according to dir)
if (syn-tag(ch) $\in \{y_1, y_2, y_n\}$)
then hc = ch ;
return hc ;
/* choose the head-child by default */
(E) if (dir == LEFT)
then hc = leftmost-child(N)
else hc = rightmost-child(N)
return hc ;

→ Mark sisters of hc as either arguments or modifiers

b) **Input:** a head-child hc, a sister sist of hc, the position pos of sist with respect to hc, the modifier table ModTb

Output: mark sist as either an argument or an adjunct of hc

Algorithm: void MarkSisterOfHead(sist, hc, pos, ModTb)

/* mark sist according to its function tags */

(A) if (sist has a function tag that always marks argument)
then mark sist as an argument ; return;

(B) if (sist has a function tag that always marks modifier)
then mark sist as a modifier ; return;

/* mark sist according to the modifier table */

(C) head tag = syn-tag(hc);

(D) Find the entry (head tag , mod num , $y_1/y_2/\ldots/y_n$) in ModTb

(E) if (((pos == LEFT) and (mod num == 0)) or
((pos == RIGHT) and (mod num == 0)))
then /* hc does not have left (right, resp.) modifiers */
mark sist as an argument;
return;

(F) if (the tag of sist matches any y_i)
then mark sist as an modifier ; return;

/* mark sist using the default

(G) mark sist as an argument .

→ After marking the nodes as being heads, arguments or modifiers, fully bracket the tree:

c) **Input:** a ttree T from a Treebank, the head percolation table HeadTb , the modifier table ModTb
Output: T becomes a derived tree
Algorithm: void BuildDerivedTree(T, HeadTb , ModTb)
(A) TargetList = $\{Root\}$, Root is the root of T ;
(B) while (TargetList is not empty)
Let R be the first node in TargetList;
TargetList = TargetList - $\{R\}$;
if (R is a leaf node)
then continue;
hc = FindHeadChild(R, HeadTb);
/* R 's children consist of a head, 0 or more arguments, 0 or more adjuncts */
add each child of R to TargetList;
mark each child other than hc as an argument or an adjunct
if (at least one child is marked as an adjunct)
then insert new nodes $\{R_i\}$ between R and hc so that at each level between R and hc exactly one of the following holds:
{ there are exactly one new node R i and one adjunct, where R_i has the same syntactic tag as its parent, or
there are a node (hc or a new node R i) with the label syn-tag(hc) and zero or a group of arguments of hc }

Step 2: Extract elementary trees from the tree

Input: a derived tree T, head percolation table HeadTb, modifier table ModTb
Output: a set of elementary trees EtreeSet
Notations: Given a node x in T , x.top and x.bot are the top and bottom part of x. f(x.top) (f(x.bot), resp.) is the etree node copied from x.top (x.bot, resp.).

Algorithm: etree-list BuildEtrees(T, HeadTb , ModTb)

(A) EtreeSet = {} ; R = Root(T);
(B) hc = FindHeadChild(R, HeadTb);
if (the head-child does not dominate any lexical word)
then /* This will ensure that no spine etree is anchored by empty categories */
choose its sibling to be hc;
(C) Based on the relation between hc and its sisters, go to one of the following (see next slide):

Step 2: Extract elementary trees from the tree

(C) Based on the relation between hc and its sisters, go to one of the following:

(C1) predicate-argument relation:

/* build a spine-etree Ts , which is formed by a predicate and its arguments */
find a head-path p from R to a leaf node A in the T .
for (each non-link node x on p) {
  /* a non-link node is a node whose head-child and other children
  form a head-argument relation */
    copy x.bot to Ts ;
    for (each child y_i of x)
      copy each y_i.top to Ts , as f(x.bot)'s child.
      if (y_i does not dominate any lexical words)
        then copy the whole subtree rooted at y_i to Ts
}
mark f(A.top) as the anchor of Ts.

EtreeSet = EtreeSet $\cup$ {*Ts*};

Step 2: Extract elementary trees from the tree

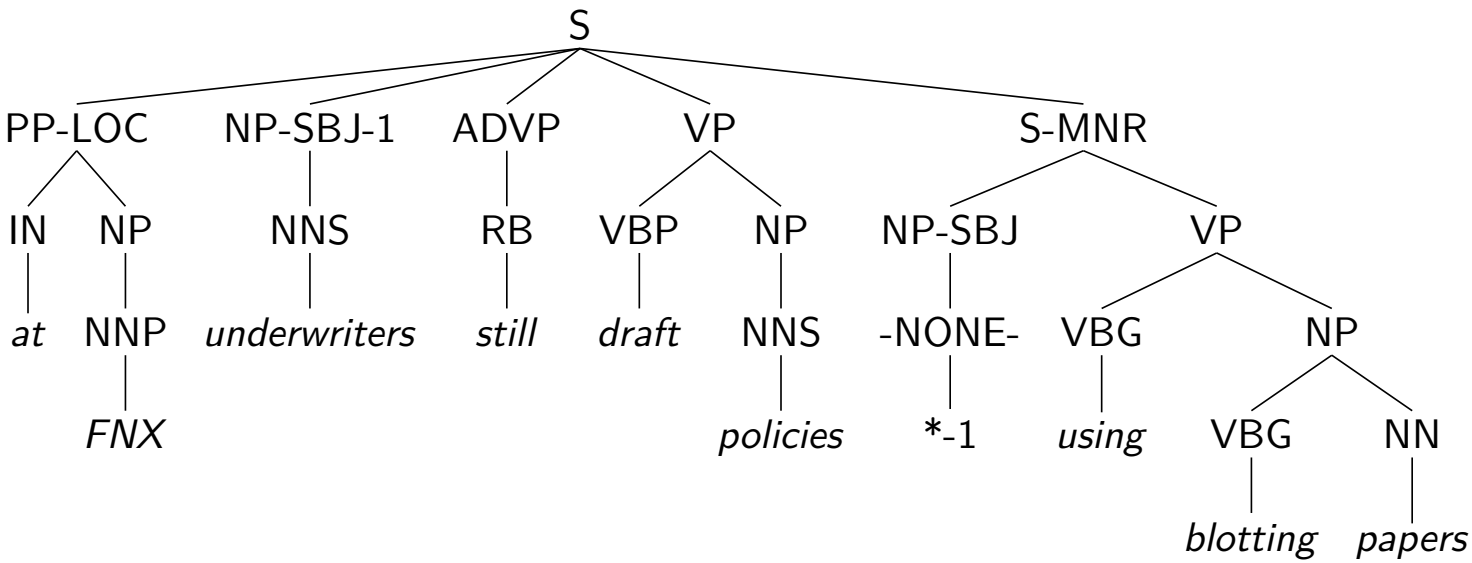
(C2) modification relation:

/* build a mod-etree Tm , which is formed by a modifier-modifiee pair and a spine-etree */
At this stage, hc should have only one sister, call it mod ;
Find a head-path p from mod to leaf node A in the derived tree;
Build an etree Ts from the path p as stated in (C1);
Copy R.bot, hc.top, mod.top to Tm ;
Create a mod-etree Tm in which f(R.bot) is the root and has two children:
f(hc.top) and f(mod.top); f(hc.top) is the foot node;
Make Ts a subtree of Tm whose root is f(mod.top);
EtreeSet = EtreeSet $\cup \{Tm\}$;

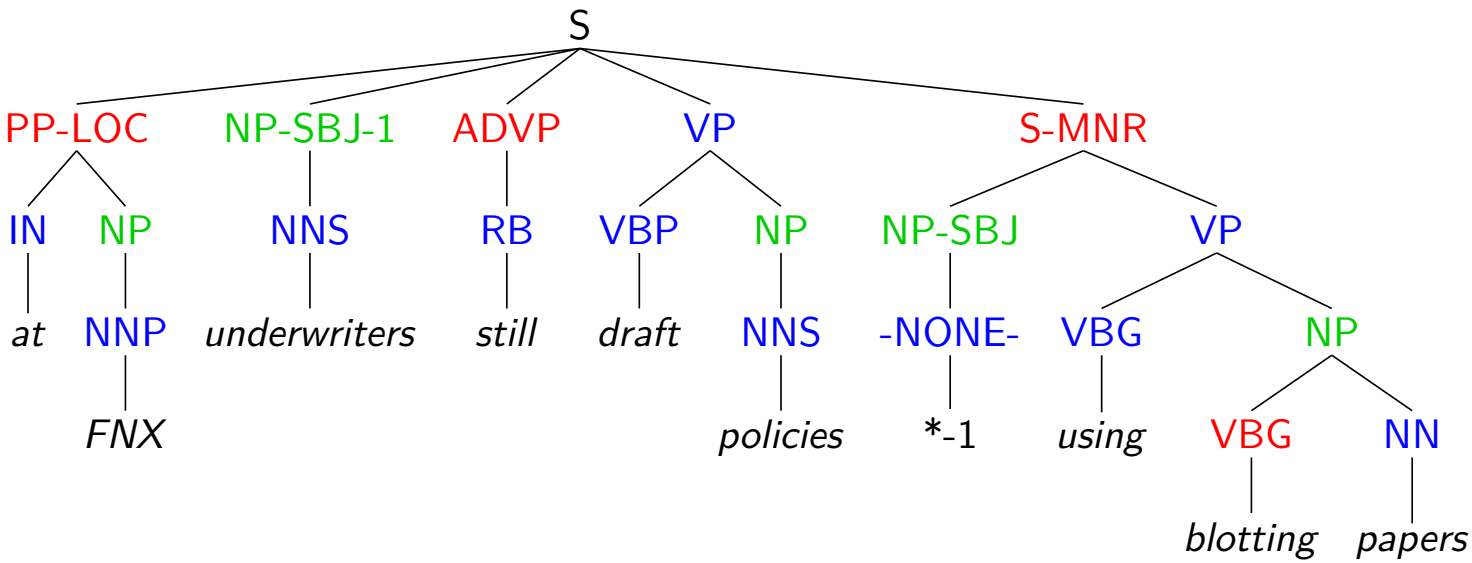
(D) Repeat step (B)-(C) for each child ch of R if ch.bot has not been copied to any etree;

(E) return EtreeSet;

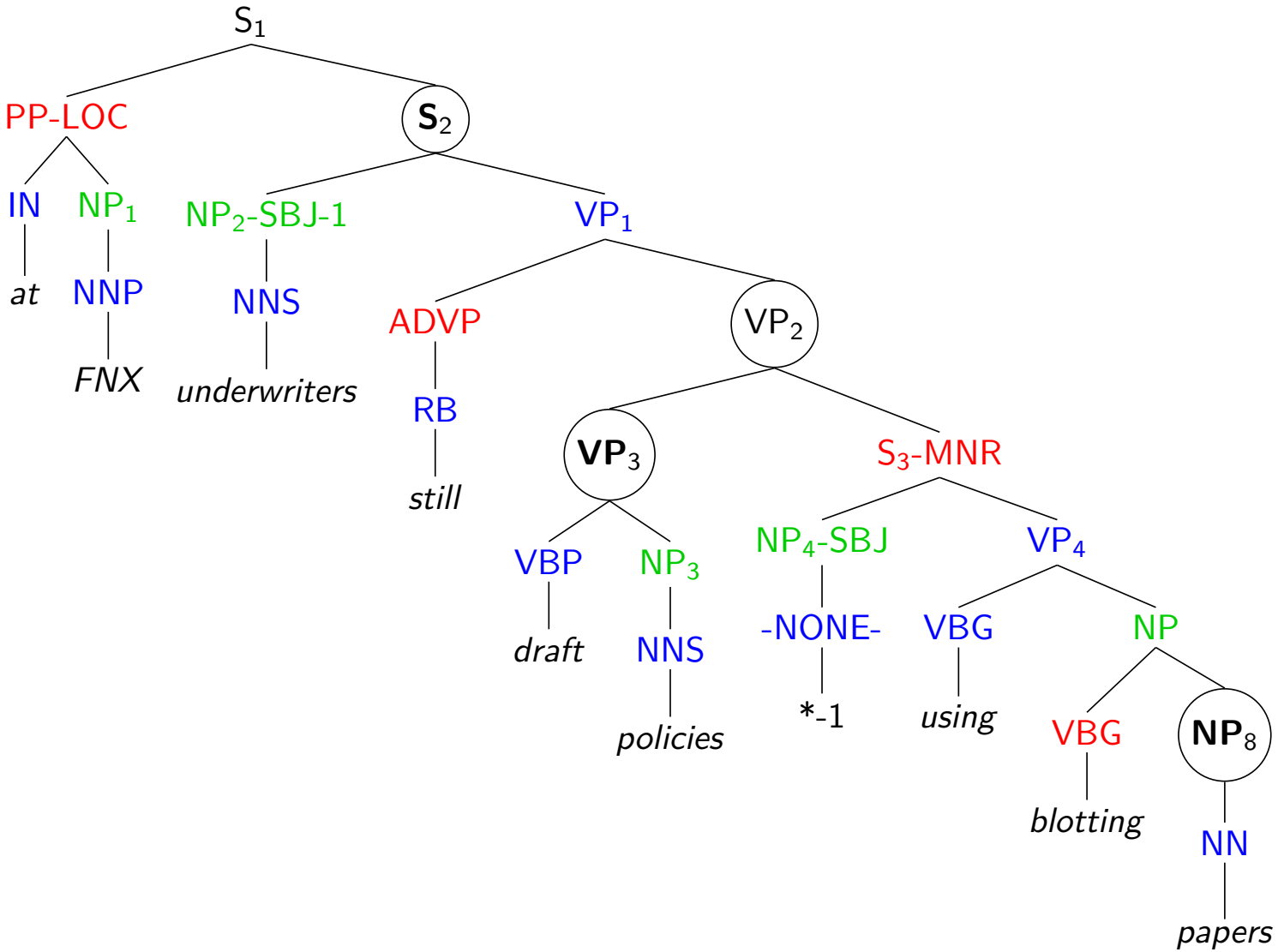
Step 1: Fully bracketing the treebank trees



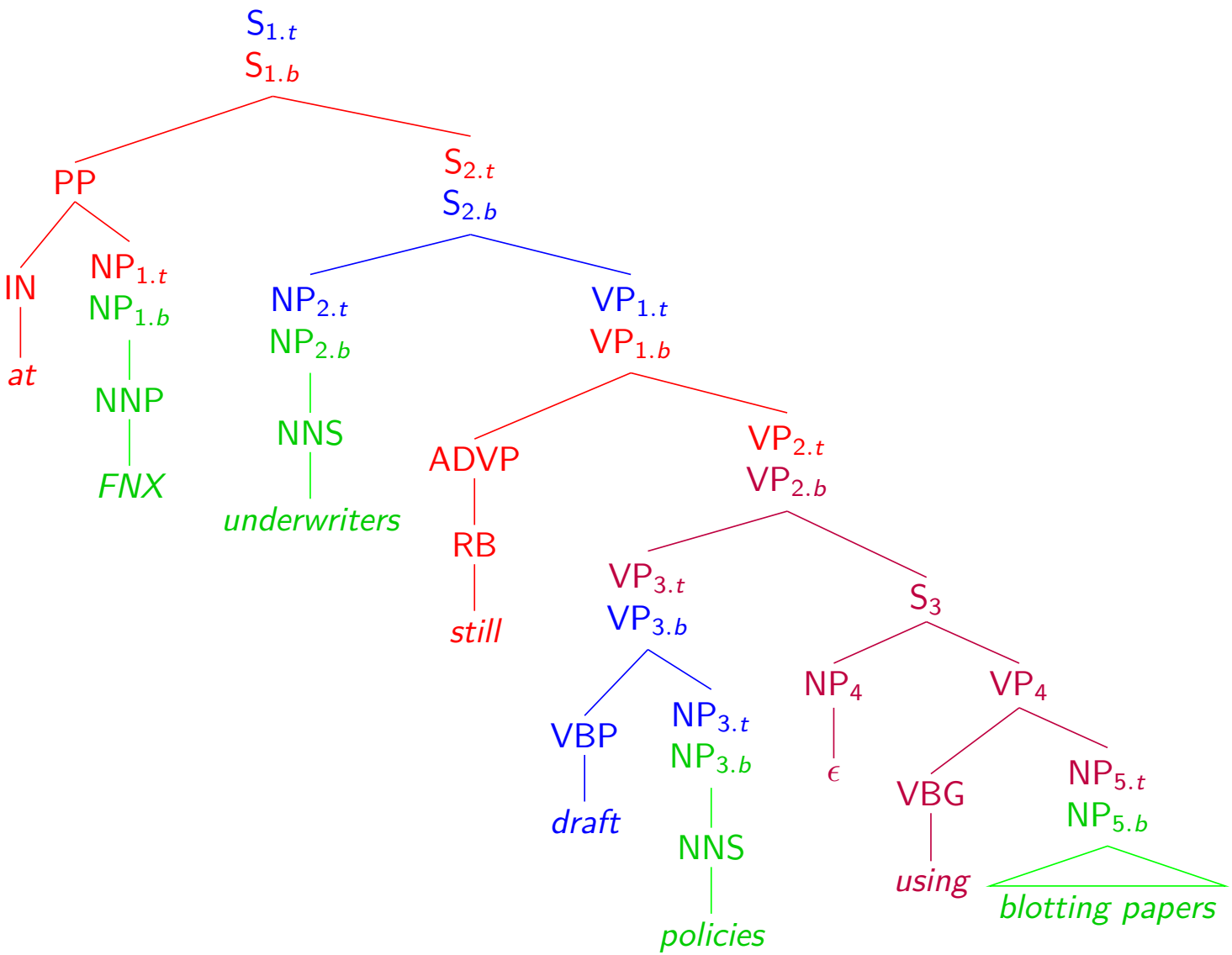
Step 1: Fully bracketing the treebank trees



Step 1: Fully bracketing the treebank trees

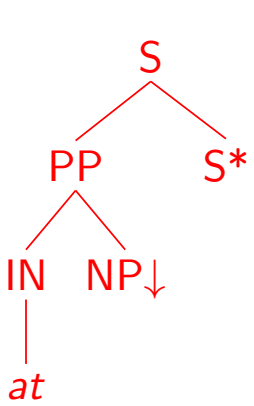


Step 2: Extracting elementary trees



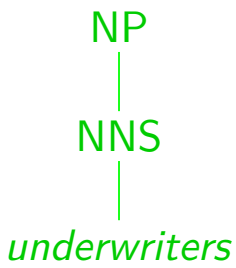
Step 2: Extracting elementary trees

#1

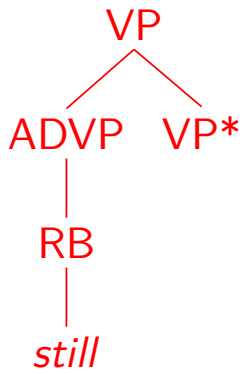


#2

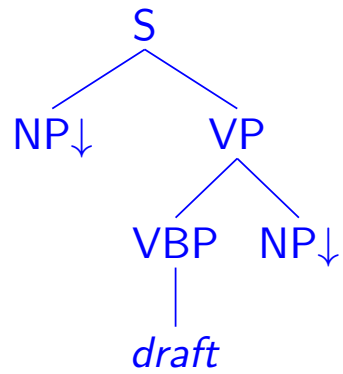
#3



#4

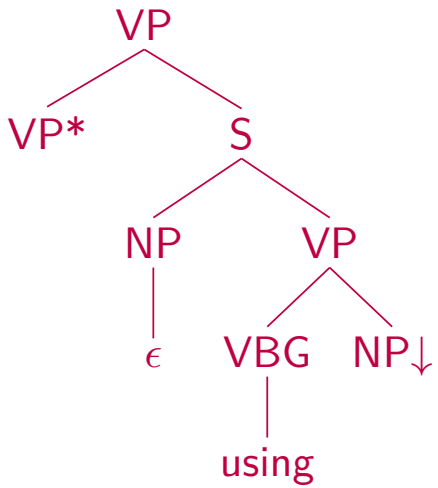


#5

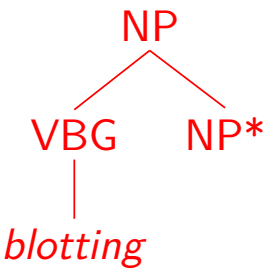


#6

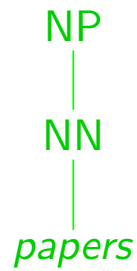
#7



#8



#9



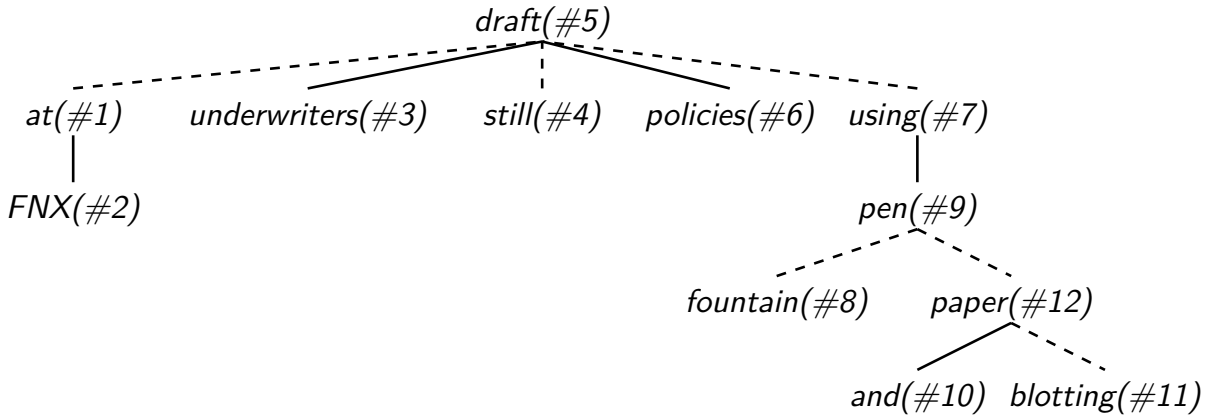
Step 3: filter out linguistically implausible trees

- Linguistically implausible trees are :
  - trees with erroneous labels (e.g. an adjective erroneously marked as a preposition) ,
  - trees with an erroneously marked head child (e.g. an adjective marked as a head of an NP)
  - other trees which do not meet the criteria of the design for extracted TAG
- Filtering out implausible trees is not a trivial task
- One possibility to rule out implausible trees is to first decompose a template into a set of sub-templates then mark the template as plausible if and only if every sub-template is plausible.
- Another possibility is to manually delete the trees from the output (if possible)

Step 4: Creating derivation trees (optional)

- no-multi-adjunction constraints

- **without** no-multi-adjunction constraints



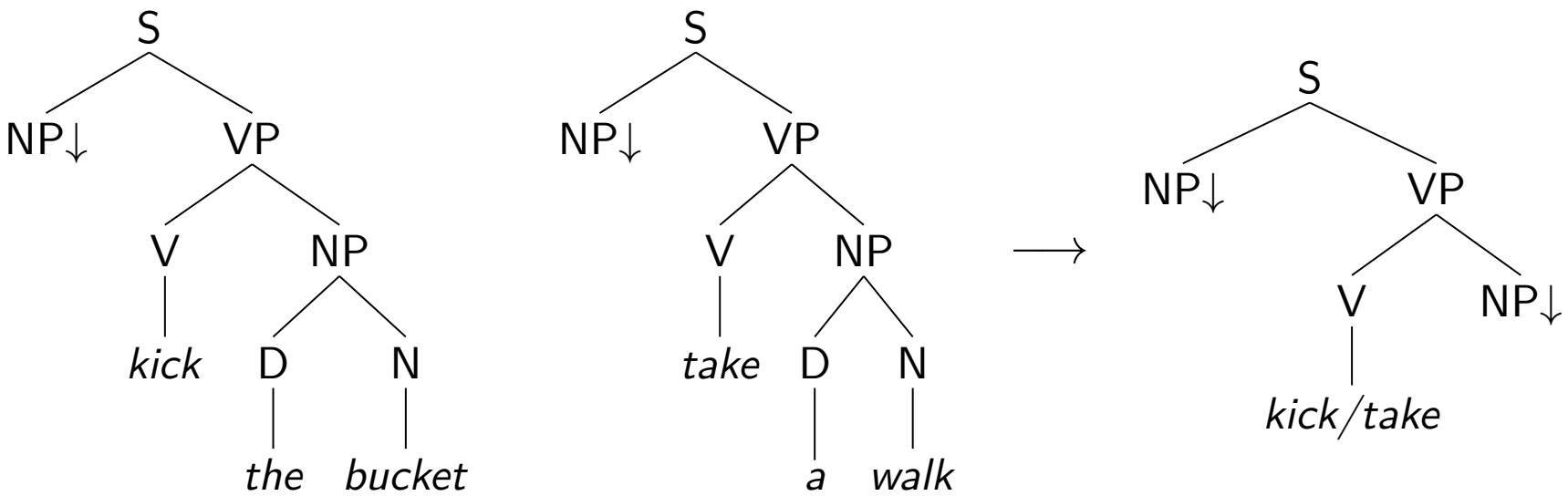
Some considerations

- It might be useful to reduce the set of the part-of-speech tags and functional tags in the treebank in order to reduce the number of elementary trees in the extracted LTAG

	tags in the PTB	**merged tags**
adjectives	JJ/JJR/JJS	A
adverbs	RB/RBR/RBS/WRB	Adv
determiners	DT/PDT/WDT/PRP$/WP$	D
nouns	CD/NN/NNS/NNP/NNPS/PRP/WP/EX/$/#	N
verbs	V/MD/VB/VBP/VBZ/VBN/VBD/VBG/TO	V
clauses	S/SQ/SBAR/SBARQ/SINV	S
noun phrases	NAC/NP/NX/QP/WHNP	NP
adjective phrases	ADJP/WHADJP	AP
adverbial phrases	ADVP/WHADVP	AdvP
preposition phrases	PP/WHPP	PP

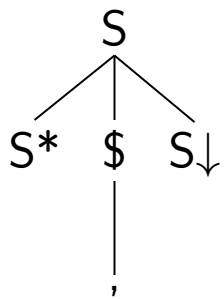
Some considerations: Co-anchors

- Although TAG elementary trees typically have one anchor, there are situations where more than one anchor could be desirable (for example, idiomatic expressions or multi-word expressions)



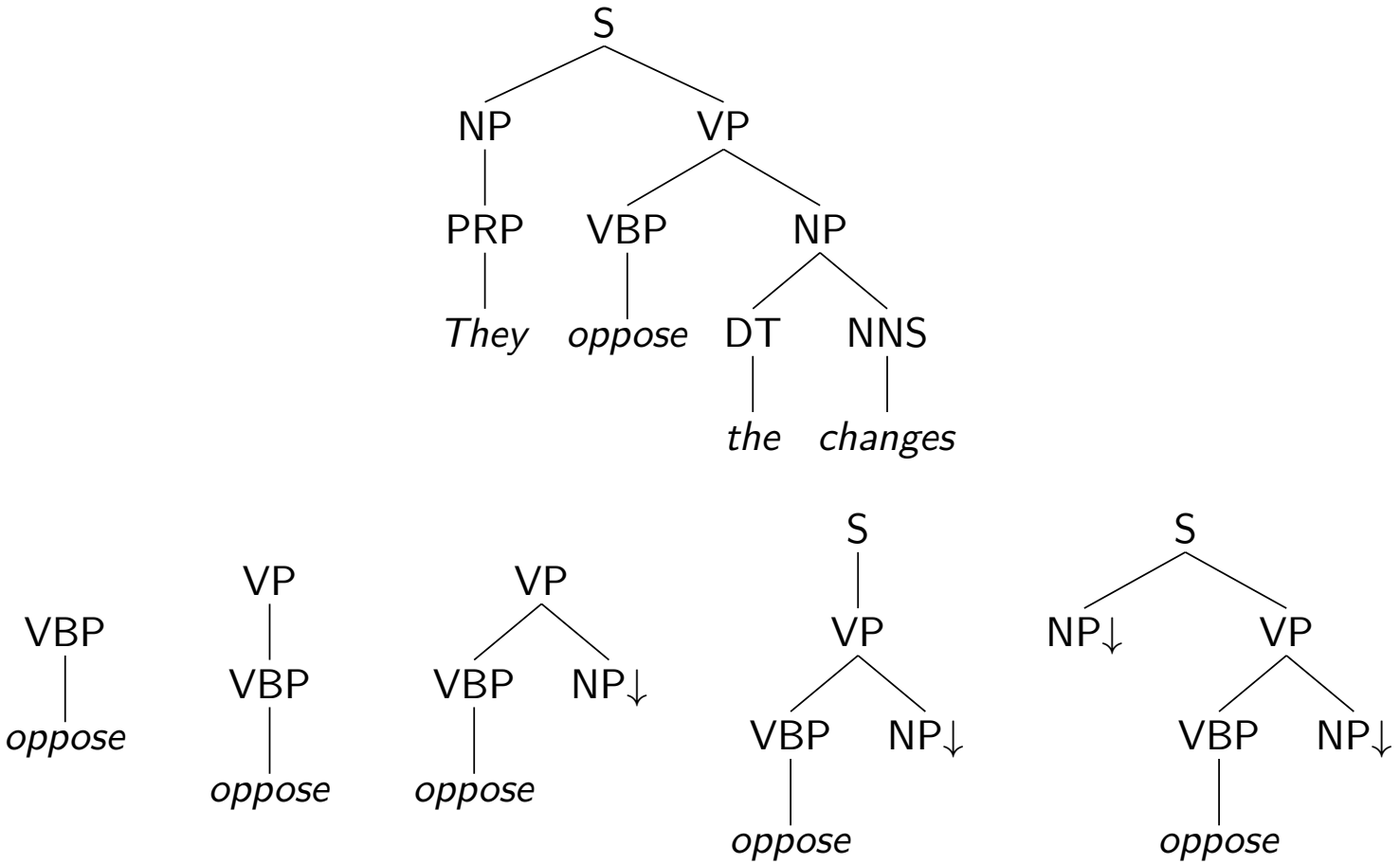
Some considerations: Punctuation

- Paired punctuation marks (e.g. quotation marks) vs. not-paired
- Punctuation marks are sometimes left out in hand-crafted LTAGs
- Paired punctuation marks are frequently beyond the TAG domain of locality.
- One way of punctuation treatment is by marking each punctuation symbol as an adjunct.



Bottom-up approach (1) [CBVS06]

- Tree-exctraction procedure is applied recursively and bottom-up
- During its application, there are multiple elementary trees in various stages are generated. [CBVS06]



Bottom-up approach (2) [CBVS06]

- One starts at the leave node and constructs the elementary tree in the following manner:
  - ⋆ extending the trunk of the partial tree by one level
  - ⋆ adding complements as substitution nodes to the root of the resulting tree
- elementary trees are extracted in parallel
- one tree is selected to be the *head partial tree* based on the roots of the partial trees and their common parent.
- a partial tree whose root is labeled as adjunct is factored into a modifier auxiliary tree that adjoines onto the head partial tree.

[CBVS06]

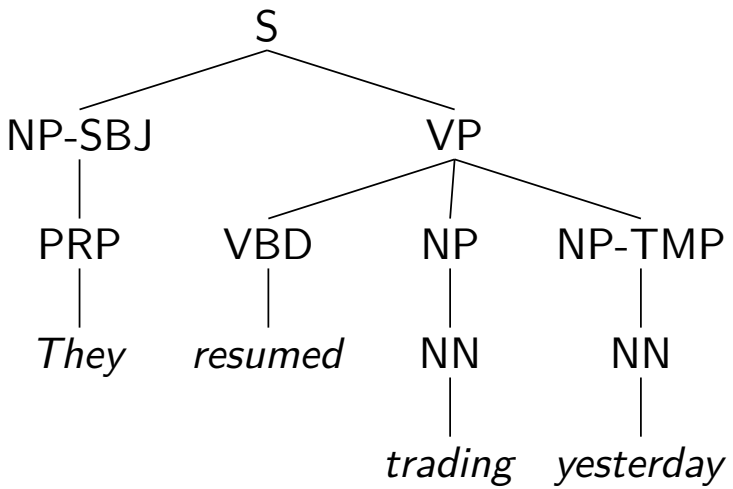
Bottom-up approach (algorithm) [CBVS06]

1. The procedure begins by associating an anchor (leave) and its corresponding preterminal with partial tree γ_0 . A *partial tree* is an elementary tree in a possibly completed or still incomplete stage of generation.
2. The full elementary tree is constructed bottom up from γ_0 , alternating between two steps: growing the trunk of the partial tree by one level, and adding substitution nodes as daughters of the root of the partial tree.
3. (for continuous steps, see next page)

Bottom-up approach (algorithm) [CBVS06]

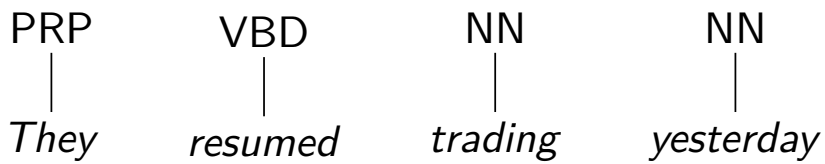
1. Let $\mu_1, \mu_2, \ldots, \mu_n$ be the root nodes of sibling subtrees $\gamma_1, \gamma_2, \ldots, \gamma_n$ respectively. Let μ_p, be an immediately dominating node corresponding to these siblings' parent:
   - One partial tree μ_i is selected to be the *head partial tree* based on the roots of the partial trees and their common parent μ_p
   - Roots of the remaining partial trees are marked as complements or adjuncts
   - The head partial tree is then extended by one level
   - Duplicates are made of complement nodes such that they become substitution nodes in the head partial tree.
   - A partial tree whose root is labeled as an adjunct is factored into a modifier auxiliary tree that adjoins onto the head partial tree
   - Repeat until reaching the partial tree whose root is labeled with a start symbol and which has marked substitution sites for the arguments of the head child.

Bottom-up approach (example)

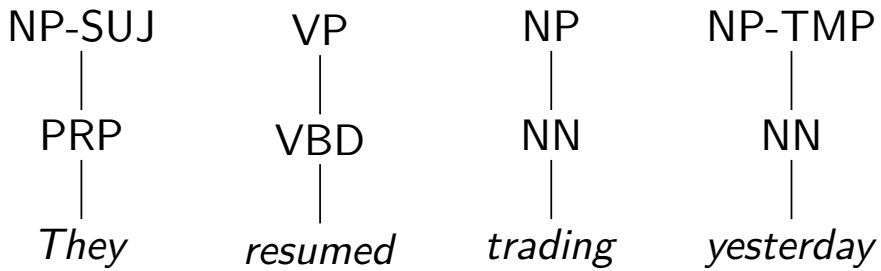


Bottom-up approach (example)

- First round of the algorithm:



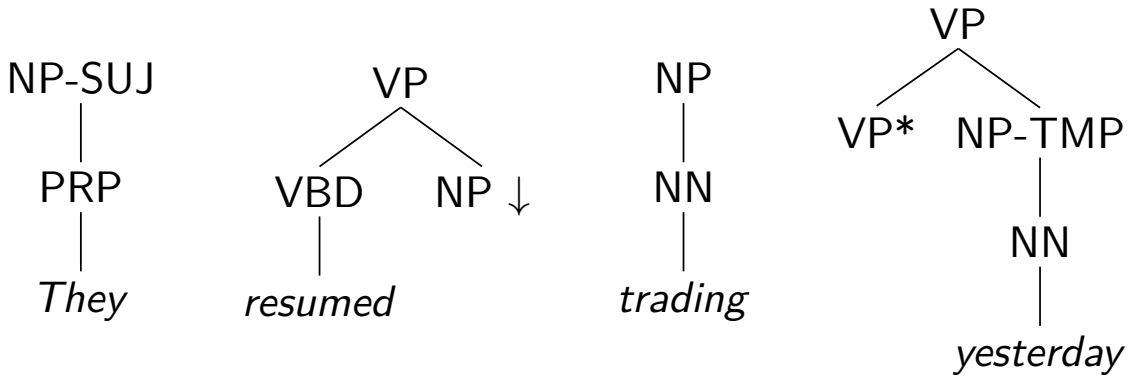
- Second round of the algorithm:



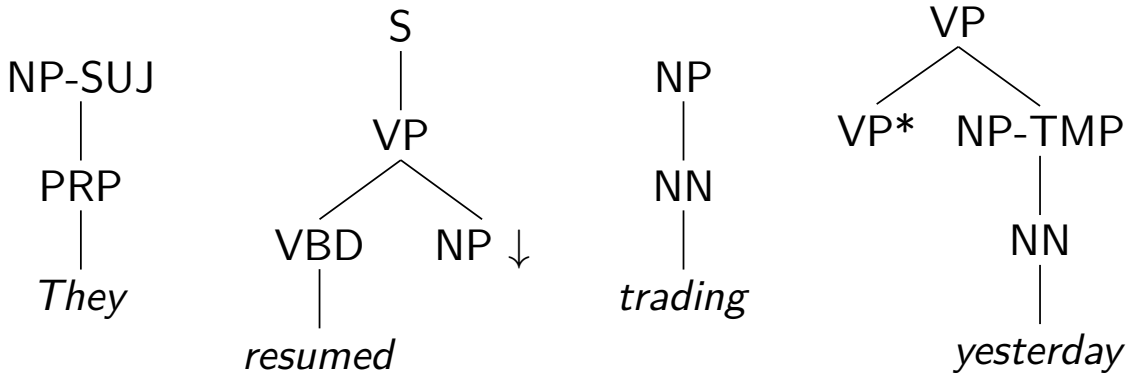
[CBVS06]

Bottom-up approach (example)

- Third round of the algorithm:



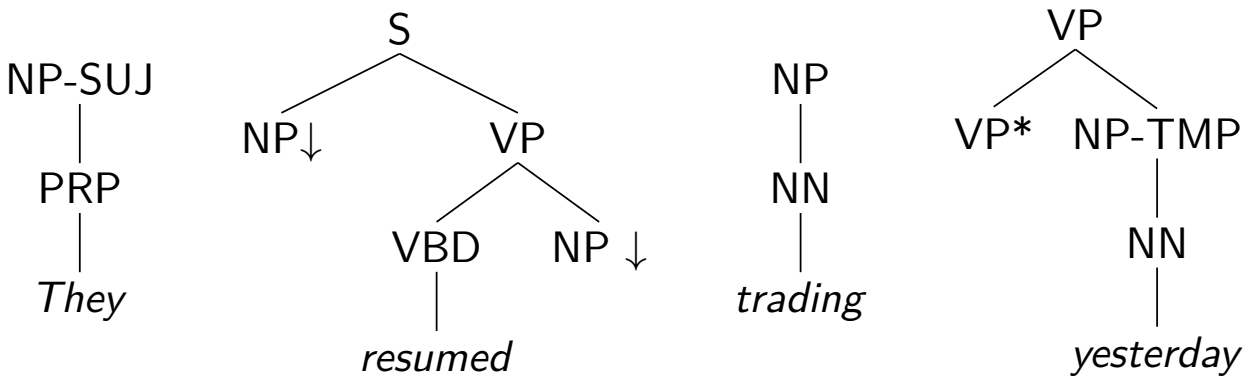
- Fourth round of the algorithm:



[CBVS06]

Bottom-up approach (example)

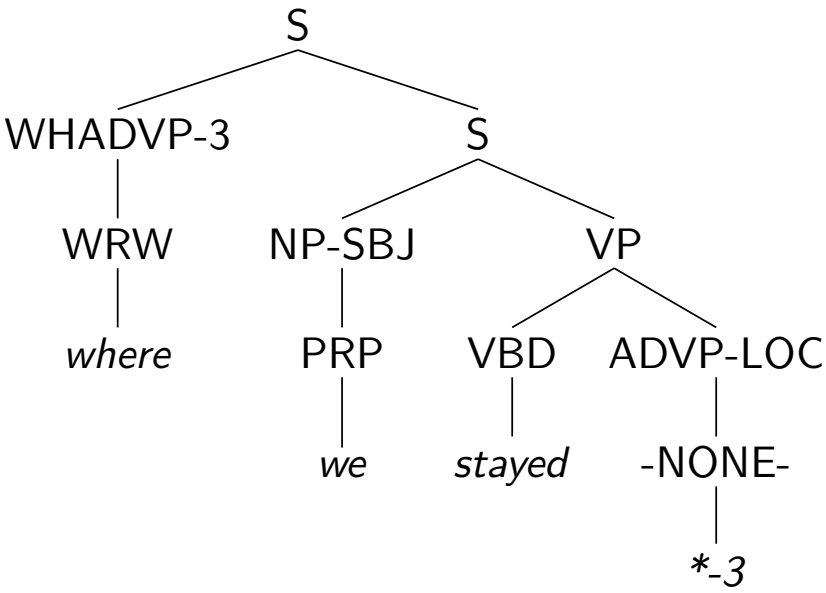
- Last round of the algorithm:

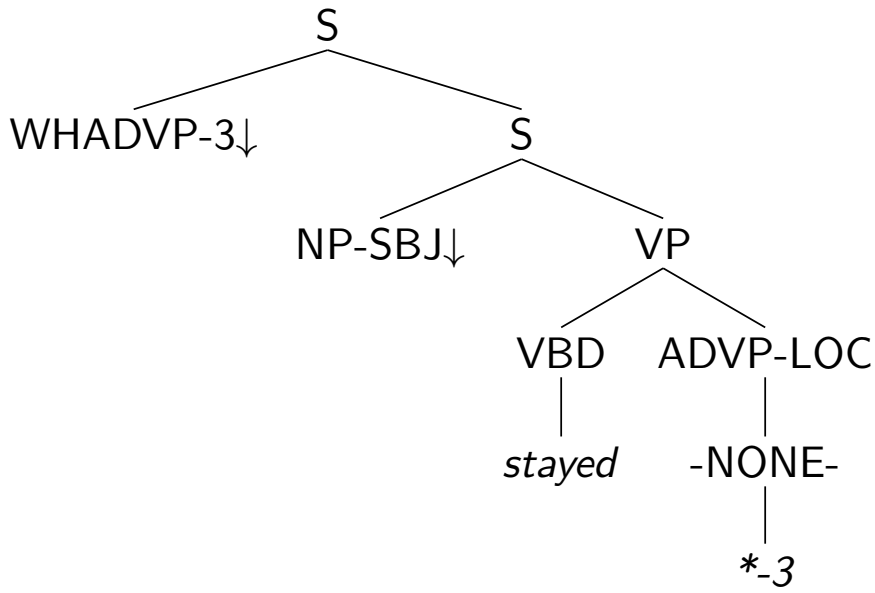


[CBVS06]

Some considerations: Empty elements

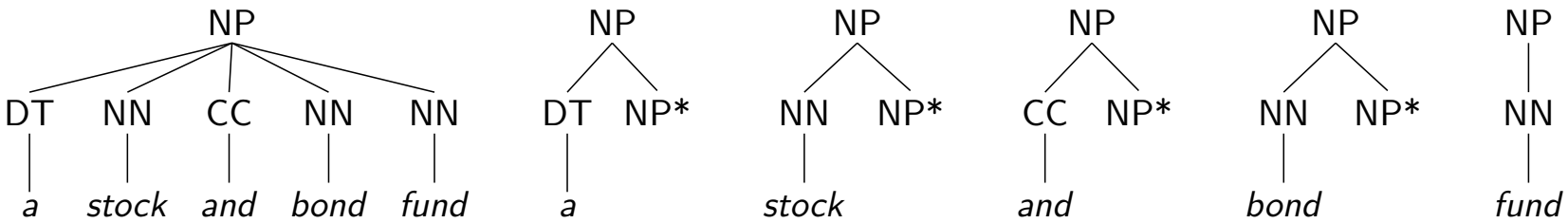
- Empty elements are terminal nodes in bracketings that do not represent lexical items
- Present in Penn Treebank and in the hand-crafted TAGs $\longrightarrow$ desirable to keep in the extracted LTAG?
- Modification of extraction algorithm can be made which grafts partial trees anchored by empty elements onto partial trees anchored by non-empty elements.





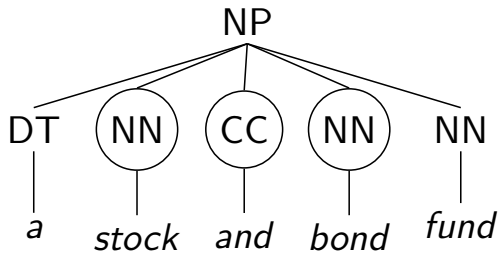
Some considerations: Conjunctions

- According to the proposed algorithm, constructions with conjunctions are treated like this:

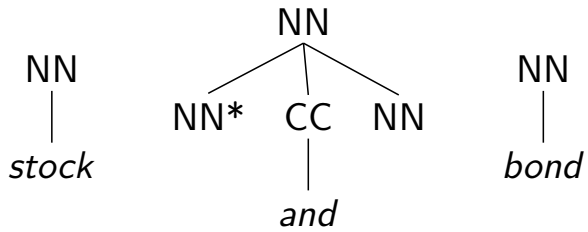


Some considerations: Conjunctions

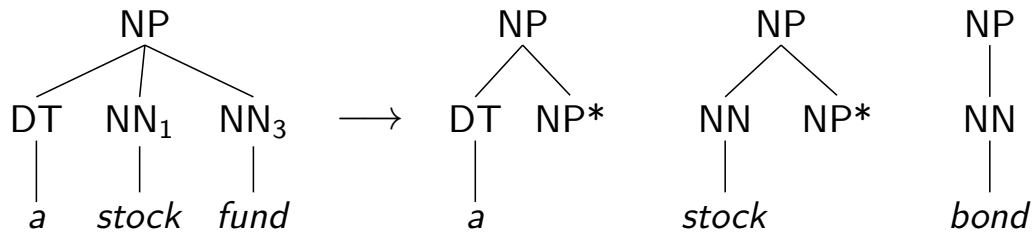
- Identify a sequence of daughters $n_1, \ldots n_j$, such that $1 \leq i < j \leq m$ representing conjuncts and associated conjunctions. n_i is the leftmost conjunct, n_j the rightmost conjunct.



- From the daughters $n_{i+1}, \ldots, n_j$ factor the conjunction into its own auxiliary tree. Auxiliary tree is anchored with conjunction. It adjoins onto n_i. Conjuncts become initial trees that substitute into the auxiliary tree.



- Default processing resumes on nodes $n_1, \ldots, n_i, n_{j+1}, \ldots, n_m$



Extracted TAGs: statistics

Parameters	**French LTAG** [BvCSK18]	**German LTAG** [Kae12]	**English LTAG** [KFM+17]
Elementary trees	5145	3426	4727
Elementary trees once	2693	1562	2165
POS tags	13 / 26	53	36
Sentences	21550	50000	44168
Avg. sentence length	31.34	17.71	appr. 20

References I

[BvCSK18] Tatiana Bladier, Andreas van Cranenburgh, Younes Samih, and Laura Kallmeyer.
German and french neural supertagging experiments for ltag parsing (to appear).
ACL 2018 Student Research Workshop, Melbourne, Australia, 2018.

[CBVS06] John Chen, Srinivas Bangalore, and K Vijay-Shanker.
Automated extraction of tree-adjoining grammars from treebanks.
Natural Language Engineering, 12(3):251–299, 2006.

[G+98] XTAG Research Group et al.
A lexicalized tree adjoining grammar for english.
arXiv preprint cs/9809024, 1998.

[Kae12] Miriam Kaeshammer.
A German treebank and lexicon for tree-adjoining grammars.
PhD thesis, Master's thesis, Universitat des Saarlandes, Saarlandes, Germany, 2012.

[KFM+17] Jungo Kasai, Robert Frank, Tom McCoy, Owen Rambow, and Alexis Nasr.
Tag parsing with neural networks and vector representations of supertags.
In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1713–1723, 2017.

[Xia99] Fei Xia.
Extracting tree adjoining grammars from bracketed corpora.
In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, pages 398–403, 1999.

[Xia01] Fei Xia.
Automatic grammar generation from two different perspectives.
University of pennsylvania, 2001.

References II

[XP06] Fei Xia and Martha Palmer.
From treebanks to tree-adjoining grammars.
Supertagging: using complex lexical descriptions in natural language processing, pages 1–39, 2006.

[XPJ00] Fei Xia, Martha Palmer, and Aravind Joshi.
A uniform method of grammar extraction and its applications.
In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 53–62. Association for Computational Linguistics, 2000.