Language modeling with tree-adjoining grammars

Day 2

Kata Balogh & Simon Petitjean

(*Heinrich-Heine-Universität Düsseldorf*)

ESSLLI 2023

University of Ljubljana, 31 July – 4 August, 2023

erc TreeGraSP

Funded by DFG Deutsche Forschungsgemeinschaft German Research Foundation

hhu Heinrich Heine Universität Düsseldorf

Recall: definition of TAG

Tree Adjoining Grammar (TAG)

A Tree Adjoining Grammar is a tuple $G = \langle N, T, I, A, O, C \rangle$:

T and N are disjoint alphabets of terminals (T) and non-terminals (N),

I is a finite set of **intial trees**, and

A is a finite set of **auxiliary trees**.

$O : \{v \mid v \text{ is a node in a tree in } I \cup A\} \rightarrow \{1, 0\}$ is a function, and

$C : \{v \mid v \text{ is a node in a tree in } I \cup A\} \rightarrow \mathcal{P}(A)$ is a function.

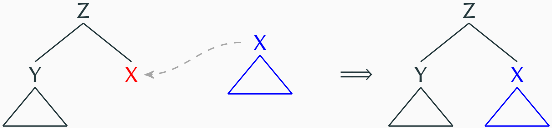
The trees in $I \cup A$ are called **elementary trees**.

Let v be a node in $I \cup A$:

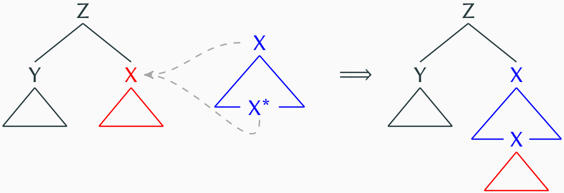
- **obligatory adjunction (OA)**: $O(v) = 1$
- **null adjunction (NA)**: $O(v) = 0$ and $C(v) = \emptyset$
- **selective adjunction (SA)**: $O(v) = 0$ and $C(v) \neq \emptyset$ and $C(v) \neq A$

Recall: operations in TAG

Substitution: replace a non-terminal leaf node with another tree



Adjunction: replace a non-terminal node with an auxiliary tree



Recall: the ideal grammar formalism

TAG is **mildly context-sensitive**

- generates the context-free languages
- generates cross-serial dependencies (i.e. ww)
- constant growth (or semi linear, no a^{2^n})
- polynomial time parsing ($\mathcal{O}(n^6)$)

[Joshi 1985, Schabes 1990, Joshi & Schabes 1997, Kallmeyer 2010]

TAG can **strongly lexicalize** finitely ambiguous CFG.

[Schabes 1990, Joshi & Schabes 1997]

TAG is linguistically, computationally and psycholinguistically **adequate**.

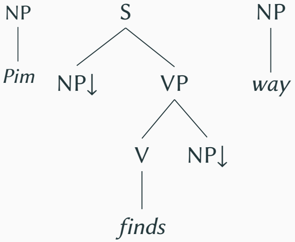
Example TAG

$G_{TAG} = \langle N, T, I, A \rangle$, where

$N = \{\text{S, NP, VP, V, Adv, Det}\}$

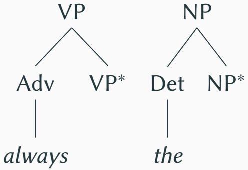
$T = \{\textit{finds, the, pim, always, way}\}$

$I = \{$



$\}$

$A = \{$



$\}$

XP↓: substitution node
XP*: footnote

Derivation in TAG

- a derivation in TAG begins with an initial tree

Derivation in TAG

- a derivation in TAG begins with an initial tree
- in a final tree all leaves have terminal symbols $\rightsquigarrow$ **derived tree**

Derivation in TAG

- a derivation in TAG begins with an initial tree
- in a final tree all leaves have terminal symbols $\rightsquigarrow$ **derived tree**
- **derived tree** in TAG
  - the tree obtained by derivation
  - final phrase structure tree
  - equivalent with the derivation tree of a CFG

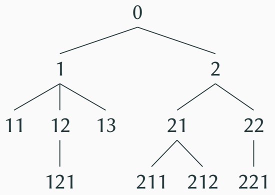
Derivation in TAG

- a derivation in TAG begins with an initial tree
- in a final tree all leaves have terminal symbols ⇝ **derived tree**
- **derived tree** in TAG
  - the tree obtained by derivation
  - final phrase structure tree
  - equivalent with the derivation tree of a CFG
- **derivation tree** in TAG
  - uniquely describes a TAG derivation
  - the derivation tree contains:
    - **nodes** for all elementary trees used in the derivation,
    - **edges** for all adjunctions and substitutions performed throughout the derivation,
    - **edge labels** indicating the target node of the rewriting operation

Derivation trees

For the node addresses of elementary trees, **Gorn addresses** are used:

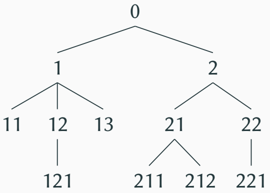
- the root has address ϵ (or 0)
- the n^{th} daughter of the node with address p has address pn.



Derivation trees

For the node addresses of elementary trees, **Gorn addresses** are used:

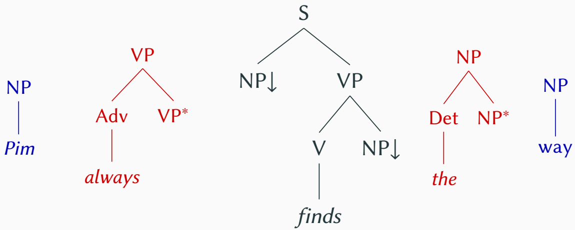
- the root has address ϵ (or 0)
- the n^{th} daughter of the node with address p has address pn.



Derivation tree

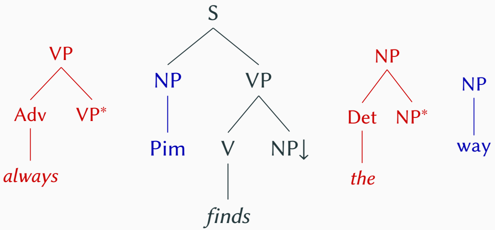
Whenever an elementary tree γ rewrites the node at Gorn address p in the elementary tree γ', there is an edge from γ' to γ labeled with p.

Example derivation

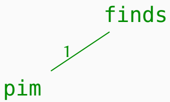


Derivation tree:

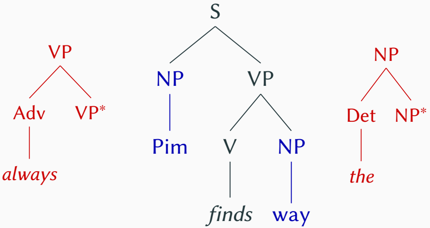
Example derivation



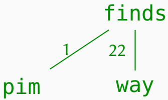
Derivation tree:



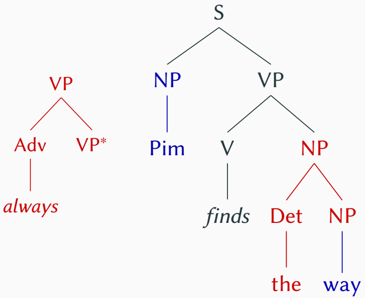
Example derivation



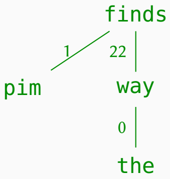
Derivation tree:



Example derivation

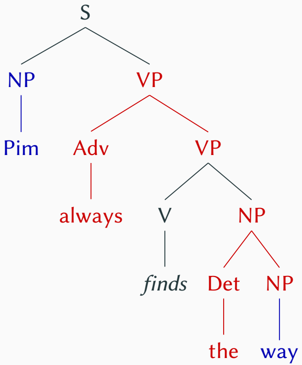


Derivation tree:

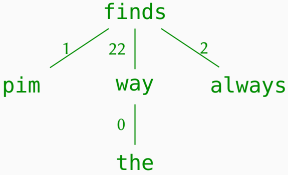


Example derivation

Derived tree:



Derivation tree:



What is an elementary tree, and what is its shape?

Linguistic analyses with LTAG

What is an elementary tree, and what is its shape?

syntactic/semantic properties of linguistic objects	$\stackrel{?}{\Longrightarrow}$	elementary trees

Linguistic analyses with LTAG

What is an elementary tree, and what is its shape?

syntactic/semantic properties of linguistic objects $\stackrel{?}{\Longrightarrow}$ elementary trees

$\Rightarrow$ Syntactic design principles [Frank 2002]

- Lexicalization
- Fundamental TAG Hypothesis (FTH)
- Condition on Elementary Tree Minimality (CETM)
- θ-Criterion for TAG

Linguistic analyses with LTAG

What is an elementary tree, and what is its shape?

syntactic/semantic properties of linguistic objects	$\overset{?}{\Longrightarrow}$	elementary trees

⇒ Syntactic design principles [Frank 2002]

- Lexicalization
- Fundamental TAG Hypothesis (FTH)
- Condition on Elementary Tree Minimality (CETM)
- θ-Criterion for TAG

⇒ Semantic design principles [Abeille & Rambow 2000]

Linguistic analyses with LTAG

What is an elementary tree, and what is its shape?

syntactic/semantic properties of linguistic objects	$\overset{?}{\Longrightarrow}$	elementary trees

⇒ Syntactic design principles [Frank 2002]

- Lexicalization
- Fundamental TAG Hypothesis (FTH)
- Condition on Elementary Tree Minimality (CETM)
- θ-Criterion for TAG

⇒ Semantic design principles [Abeille & Rambow 2000]

⇒ Design principle of economy

Syntactic design principles (1): Lexicalization

Each elementary tree has at least one non-empty lexical item, its **lexical anchor**.

Syntactic design principles (1): Lexicalization

Each elementary tree has at least one non-empty lexical item, its **lexical anchor**.

⇒ All widely used grammar formalisms support some kind of lexicalization!

Syntactic design principles (1): Lexicalization

Each elementary tree has at least one non-empty lexical item, its **lexical anchor**.

⇒ All widely used grammar formalisms support some kind of lexicalization!

⇒ TAG → LTAG: Lexicalized Tree-Adjoining Grammar

[Schabes & Joshi 1990, Joshi & Schabes 1991]

(Recall: reasons for lexicalization!)

Syntactic design principles (2): FTH

Fundamental TAG Hypothesis (FTH)

Every syntactic dependency is expressed locally within an elementary tree.

[Frank 2002]

Syntactic design principles (2): FTH

Fundamental TAG Hypothesis (FTH)

Every syntactic dependency is expressed locally within an elementary tree.

[Frank 2002]

"syntactic dependency"

- valency/subcategorization
- binding
- filler-gap constructions (extraction)
- …

Syntactic design principles (2): FTH

Fundamental TAG Hypothesis (FTH)

Every syntactic dependency is expressed locally within an elementary tree.

[Frank 2002]

“syntactic dependency”

- valency/subcategorization
- binding
- filler-gap constructions (extraction)
- …

“expressed within an elementary tree”

- terminal leaf (i.e. lexical anchor)
- nonterminal leaf (substitution node and footnode)
- marking an inner node for obligatory adjunction

⇒ extended domain of locality

Complex primitives

Complicate locally, simplify globally.

"[...] start with complex (more complicated) primitives, which capture directly some crucial linguistic properties and then introduce some general operations for composing these complex structures (primitive or derived). What is the nature of these complex primitives? In the conventional approach the primitive structures (or rules) are kept as simple as possible. This has the consequence that information (e.g., syntactic and semantic) about a lexical item (word) is distributed over more than one primitive structure. Therefore, the information associated with a lexical item is not captured locally, i.e., within the domain of a primitive structure."

[Joshi 2004]

Syntactic design principles (3): CETM

Condition on Elementary Tree Minimality (CETM)

The syntactic heads in an elementary tree and their projections must form the extended projection of a single lexical head.

[Frank 2002]

Note: We only use simple, non-extended projections!

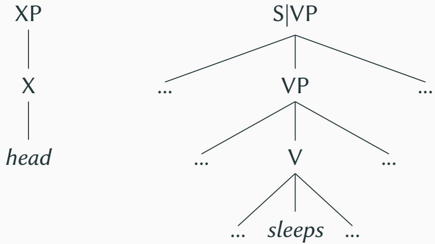
Syntactic design principles (3): CETM

Condition on Elementary Tree Minimality (CETM)

The syntactic heads in an elementary tree and their projections must form the extended projection of a single lexical head.

[Frank 2002]

Note: We only use simple, non-extended projections!



Syntactic design principles (4): θ-Criterion for TAG

Thematic role (θ-role)

The semantic relationship of an argument with its predicate is expressed through the assignment of a role by the predicate to the argument.

Syntactic design principles (4): θ-Criterion for TAG

Thematic role (θ-role)

The semantic relationship of an argument with its predicate is expressed through the assignment of a role by the predicate to the argument.

- Different theta-roles have different labels, such as AGENT, THEME, PATIENT, GOAL, SOURCE, EXPERIENCER etc.
- *Bart kicked the ball.*
  - *kicked* $\rightsquigarrow$ predicate
  - *Bart* $\rightsquigarrow$ AGENT
  - *ball* $\rightsquigarrow$ THEME/PATIENT

Syntactic design principles (4): θ-Criterion for TAG

Thematic role (θ-role)

The semantic relationship of an argument with its predicate is expressed through the assignment of a role by the predicate to the argument.

- Different theta-roles have different labels, such as AGENT, THEME, PATIENT, GOAL, SOURCE, EXPERIENCER etc.
- *Bart kicked the ball.*
  - *kicked* $\rightsquigarrow$ predicate
  - *Bart* $\rightsquigarrow$ AGENT
  - *ball* $\rightsquigarrow$ THEME/PATIENT
- *The ball was kicked by Bart.*
  - *kicked* $\rightsquigarrow$ predicate
  - *Bart* $\rightsquigarrow$ AGENT
  - *ball* $\rightsquigarrow$ THEME/PATIENT

Syntactic design principles (4): θ-Criterion for TAG

θ-Criterion (TAG version) **[Frank 2002]**

a. If H is the lexical head of an elementary tree T,
H assigns all of its θ-roles in T.

b. If A is a frontier non-terminal of elementary tree T,
A must be assigned a θ-role in T.

$\Longrightarrow$ Valency/subcategorization is expressed only with non-terminal leaves!

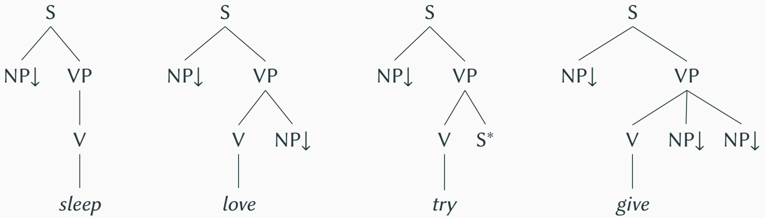
Syntactic design principles (4): θ-Criterion for TAG

θ-Criterion (TAG version) **[Frank 2002]**

a. If H is the lexical head of an elementary tree T,
   H assigns all of its θ-roles in T.

b. If A is a frontier non-terminal of elementary tree T,
   A must be assigned a θ-role in T.

$\Longrightarrow$ Valency/subcategorization is expressed only with non-terminal leaves!



Further design principles

Semantic design principles

Predicate-argument co-occurrence:

Each elementary tree associated with a predicate contains a non-terminal leaf for each of its arguments.

Further design principles

Semantic design principles

Predicate-argument co-occurrence:

Each elementary tree associated with a predicate contains a non-terminal leaf for each of its arguments.

Semantic anchoring:

Elementary trees are not semantically void (to, that.)

Further design principles

Semantic design principles

Predicate-argument co-occurrence:

Each elementary tree associated with a predicate contains a non-terminal leaf for each of its arguments.

Semantic anchoring:

Elementary trees are not semantically void (to, that.)

Compositional principle:

An elementary tree corresponds to a single semantic unit.

Further design principles

Semantic design principles

Predicate-argument co-occurrence:

Each elementary tree associated with a predicate contains a non-terminal leaf for each of its arguments.

Semantic anchoring:

Elementary trees are not semantically void (to, that.)

Compositional principle:

An elementary tree corresponds to a single semantic unit.

Design principle of economy

The elementary trees are shaped in such a way, that the size of the elementary trees and the size of the grammar is minimal.

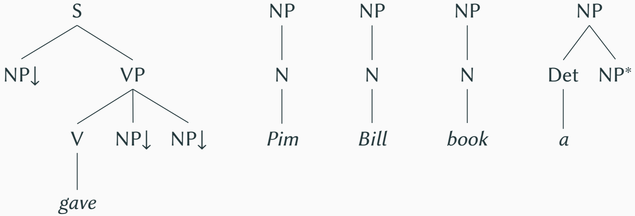
Sample derivations: NP and PP complements

(1) Pim gave Bill a book.

Sample derivations: NP and PP complements

(1) Pim gave Bill a book.

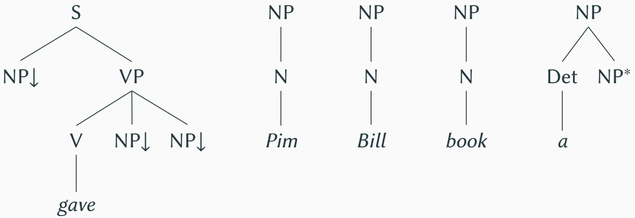
Elementary trees:



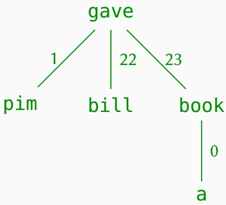
Sample derivations: NP and PP complements

(1) Pim gave Bill a book.

Elementary trees:



Derivation tree:



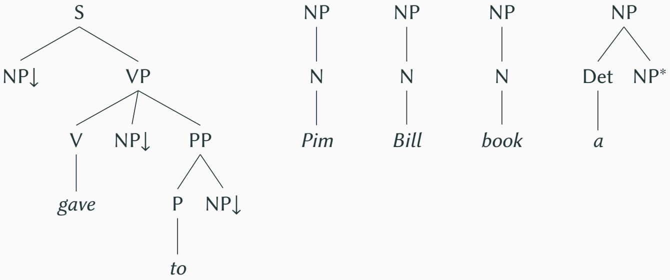
Sample derivations: NP and PP complements

(2) Pim gave a book to Bill.

Sample derivations: NP and PP complements

(2) Pim gave a book to Bill.

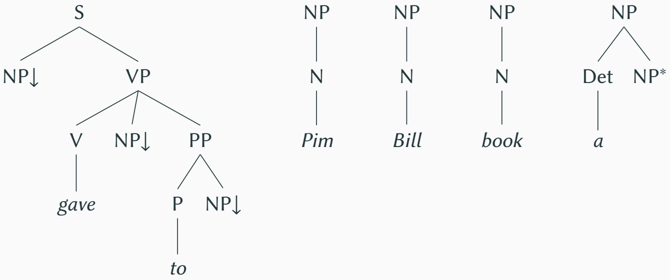
Elementary trees:



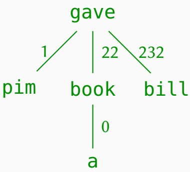
Sample derivations: NP and PP complements

(2) Pim gave a book to Bill.

Elementary trees:



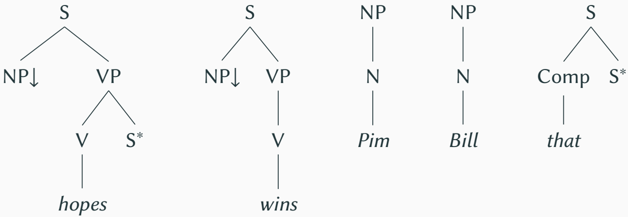
Derivation tree:



Sample derivations: Sentential complements

(3) Pim hopes that Bill wins.

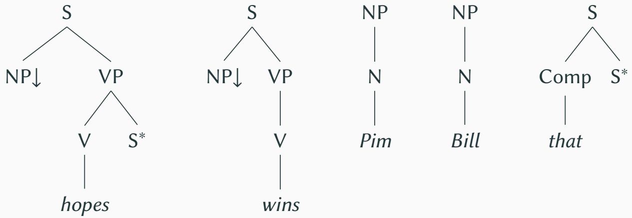
Elementary trees:



Sample derivations: Sentential complements

(3) Pim hopes that Bill wins.

Elementary trees:



Derivation tree:

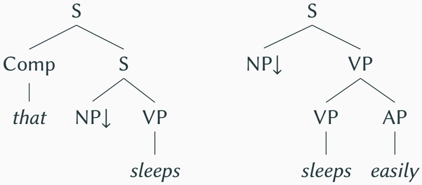
Modification and functional elements

How to insert **modifiers** (e.g. *easily*) and **functional elements** (complementizers, determiners, do-auxiliaries, ...)?

Modification and functional elements

How to insert **modifiers** (e.g. *easily*) and **functional elements** (complementizers, determiners, do-auxiliaries, ...)?

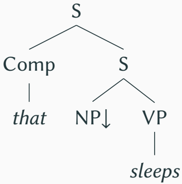
- either as co-anchor in the elementary tree of the lexical item they are associated with

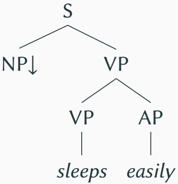


Modification and functional elements

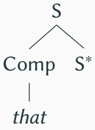
How to insert **modifiers** (e.g. *easily*) and **functional elements** (complementizers, determiners, do-auxiliaries, ...)?

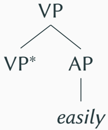
- either as co-anchor in the elementary tree of the lexical item they are associated with





- or by separate auxiliary trees (e.g., XTAG grammar)





⇒ Footnodes/Adjunctions indicate both complementation and modification.

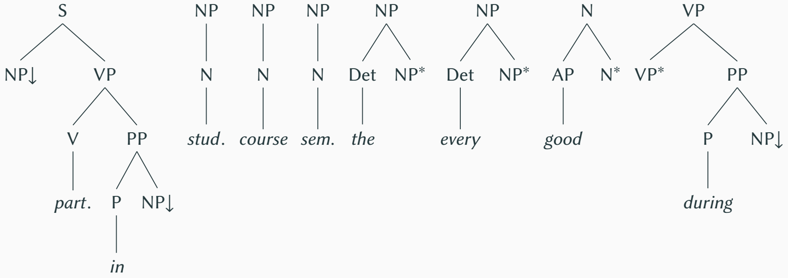
⇒ Enhancement of the CETM

[see Abeille & Rambow 2000]

Sample derivations: Modifiers

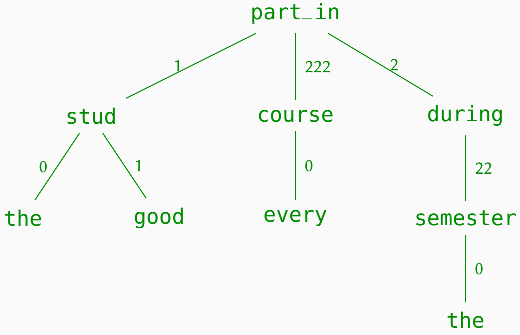
(4) The good student participated in every course during the semester.

Elementary trees:



Sample derivations: Modifiers

Derivation tree:



Feature structures

- generalizing agreement and case marking
- modelling adjunction constraints (TAG specific)

$\Rightarrow$ use **feature structures**

Feature structures

- generalizing agreement and case marking
- modelling adjunction constraints (TAG specific)

$\Rightarrow$ use **feature structures**

$\Rightarrow$ smaller grammars that are easier to maintain

Feature structures

- generalizing agreement and case marking
- modelling adjunction constraints (TAG specific)

⇒ use **feature structures**

⇒ smaller grammars that are easier to maintain

- case assignment:

  *Joe saw her. / *Joe saw she.*

  Joe expected her to come. (ECM) / **Joe expected she to come.*

Feature structures

- generalizing agreement and case marking
- modelling adjunction constraints (TAG specific)

⇒ use **feature structures**

⇒ smaller grammars that are easier to maintain

- case assignment:

  *Joe saw her. / *Joe saw she.*

  Joe expected her to come. (ECM) / **Joe expected she to come.*

- person/number agreement:

  *You sing. / *You sings.*

  *She sings. / *She sing.*

  *This woman sings. / *This woman sing.*

  *These women sing. / *These women sings.*

Feature structures

- generalizing agreement and case marking
- modelling adjunction constraints (TAG specific)

⇒ use **feature structures**

⇒ smaller grammars that are easier to maintain

- case assignment:

  Joe saw her. / **Joe saw she.*

  Joe expected her to come. (ECM) / **Joe expected she to come.*

- person/number agreement:

  You sing. / **You sings.*

  She sings. / **She sing.*

  This woman sings. / **This woman sing.*

  These women sing. / **These women sings.*

- also: definiteness agreement (Hungarian), ...

Feature structures

- a list of features (e.g., CASE) and values (e.g., nom)

Feature structures

- a list of features (e.g., CASE) and values (e.g., nom)
- represented as **attribute-value matrices (AVM)**

Feature structures

- a list of features (e.g., CASE) and values (e.g., nom)
- represented as **attribute-value matrices (AVM)**

sings:

$$\begin{bmatrix} \textsc{cat} & \text{v} \\ \textsc{vform} & \text{finite} \\ \textsc{agr} & \begin{bmatrix} \textsc{num} & \text{sg} \\ \textsc{pers} & 3 \end{bmatrix} \end{bmatrix}$$

Feature structures

- a list of features (e.g., CASE) and values (e.g., nom)
- represented as **attribute-value matrices (AVM)**

  sings:

$$\begin{bmatrix} \textsc{cat} & \text{v} \\ \textsc{vform} & \text{finite} \\ \textsc{agr} & \begin{bmatrix} \textsc{num} & \text{sg} \\ \textsc{pers} & 3 \end{bmatrix} \end{bmatrix}$$

- feature values:
  - atomic (e.g., the value of CAT)
  - feature structures (e.g., the value of AGR)

Feature structures

- a list of features (e.g., CASE) and values (e.g., nom)
- represented as **attribute-value matrices (AVM)**

  sings:

  $$\begin{bmatrix} \textsc{cat} & \text{v} \\ \textsc{vform} & \text{finite} \\ \textsc{agr} & \begin{bmatrix} \textsc{num} & \text{sg} \\ \textsc{pers} & 3 \end{bmatrix} \end{bmatrix}$$

- feature values:
  - atomic (e.g., the value of CAT)
  - feature structures (e.g., the value of AGR)
- combining constituents ⇒ **unify** feature structures

Unification

- unification ($\sqcup$) is a partial operation on feature structures

Unification

- unification ($\sqcup$) is a partial operation on feature structures
- intuitively: unification is the operation of combining two feature structures such that the new feature structure contains all the information of the original two, and nothing more

$$\begin{bmatrix} \textsc{cat} & \text{vp} \\ \textsc{agr} & \begin{bmatrix} \textsc{num} & \text{pl} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \textsc{cat} & \text{vp} \\ \textsc{agr} & \begin{bmatrix} \textsc{pers} & 3 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \textsc{cat} & \text{vp} \\ \textsc{agr} & \begin{bmatrix} \textsc{num} & \text{pl} \\ \textsc{pers} & 3 \end{bmatrix} \end{bmatrix}$$

Unification

- unification ($\sqcup$) is a partial operation on feature structures
- intuitively: unification is the operation of combining two feature structures such that the new feature structure contains all the information of the original two, and nothing more

$$\begin{bmatrix} \textsc{cat} & \text{vp} \\ \textsc{agr} & \begin{bmatrix} \textsc{num} & \text{pl} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \textsc{cat} & \text{vp} \\ \textsc{agr} & \begin{bmatrix} \textsc{pers} & 3 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \textsc{cat} & \text{vp} \\ \textsc{agr} & \begin{bmatrix} \textsc{num} & \text{pl} \\ \textsc{pers} & 3 \end{bmatrix} \end{bmatrix}$$

- partial operation $\Rightarrow$ unification can fail

$$\begin{bmatrix} \textsc{cat} & \text{np} \\ \textsc{num} & \text{sg} \end{bmatrix} \sqcup \begin{bmatrix} \textsc{cat} & \text{np} \\ \textsc{num} & \text{pl} \end{bmatrix} = \text{FAIL}$$

Unification

- unification ($\sqcup$) is a partial operation on feature structures
- intuitively: unification is the operation of combining two feature structures such that the new feature structure contains all the information of the original two, and nothing more

$$\begin{bmatrix} \textsc{cat} & \text{vp} \\ \textsc{agr} & \begin{bmatrix} \textsc{num} & \text{pl} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \textsc{cat} & \text{vp} \\ \textsc{agr} & \begin{bmatrix} \textsc{pers} & 3 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \textsc{cat} & \text{vp} \\ \textsc{agr} & \begin{bmatrix} \textsc{num} & \text{pl} \\ \textsc{pers} & 3 \end{bmatrix} \end{bmatrix}$$

- partial operation $\Rightarrow$ unification can fail

$$\begin{bmatrix} \textsc{cat} & \text{np} \\ \textsc{num} & \text{sg} \end{bmatrix} \sqcup \begin{bmatrix} \textsc{cat} & \text{np} \\ \textsc{num} & \text{pl} \end{bmatrix} = \text{FAIL}$$

- underspecified feature values

$$\begin{bmatrix} \textsc{cat} & \text{np} \\ \textsc{case} & \text{nom} \mid \text{acc} \end{bmatrix} \sqcup \begin{bmatrix} \textsc{cat} & \text{np} \\ \textsc{case} & \text{acc} \end{bmatrix} = \begin{bmatrix} \textsc{cat} & \text{np} \\ \textsc{case} & \text{acc} \end{bmatrix}$$

Unification: definition

Unification ($F \sqcup G$)

The unification of two feature structures F and G is (if it exists) the smallest feature structure that is subsumed by both F and G.

Unification: definition

Unification ($F \sqcup G$)

The unification of two feature structures F and G is (if it exists) the smallest feature structure that is subsumed by both F and G.

Subsumption ($F_1 \sqsubseteq F_2$)

A feature structure F_1 subsumes ($\sqsubseteq$) another feature structure F_2, iff all the information that is contained in F_1 is also contained in F_2.

Unification: definition

Unification ($F \sqcup G$)

The unification of two feature structures F and G is (if it exists) the smallest feature structure that is subsumed by both F and G.

Subsumption ($F_1 \sqsubseteq F_2$)

A feature structure F_1 subsumes ($\sqsubseteq$) another feature structure F_2, iff all the information that is contained in F_1 is also contained in F_2.

That is, (if it exists) $F \sqcup G$ is the feature structure with the following three properties:

Unification: definition

Unification ($F \sqcup G$)

The unification of two feature structures F and G is (if it exists) the smallest feature structure that is subsumed by both F and G.

Subsumption ($F_1 \sqsubseteq F_2$)

A feature structure F_1 subsumes ($\sqsubseteq$) another feature structure F_2, iff all the information that is contained in F_1 is also contained in F_2.

That is, (if it exists) $F \sqcup G$ is the feature structure with the following three properties:

(1) $F \sqsubseteq (F \sqcup G)$

Unification: definition

Unification ($F \sqcup G$)

The unification of two feature structures F and G is (if it exists) the smallest feature structure that is subsumed by both F and G.

Subsumption ($F_1 \sqsubseteq F_2$)

A feature structure F_1 subsumes ($\sqsubseteq$) another feature structure F_2, iff all the information that is contained in F_1 is also contained in F_2.

That is, (if it exists) $F \sqcup G$ is the feature structure with the following three properties:

(1) $F \sqsubseteq (F \sqcup G)$

(2) $G \sqsubseteq (F \sqcup G)$

Unification: definition

Unification ($F \sqcup G$)

The unification of two feature structures F and G is (if it exists) the smallest feature structure that is subsumed by both F and G.

Subsumption ($F_1 \sqsubseteq F_2$)

A feature structure F_1 subsumes ($\sqsubseteq$) another feature structure F_2, iff all the information that is contained in F_1 is also contained in F_2.

That is, (if it exists) $F \sqcup G$ is the feature structure with the following three properties:

(1) $F \sqsubseteq (F \sqcup G)$

(2) $G \sqsubseteq (F \sqcup G)$

(3) If H is a feature structure such that $F \sqsubseteq H$ and $G \sqsubseteq H$, then $(F \sqcup G) \sqsubseteq H$. If there is no smallest feature structure that is subsumed by both F and G, then we say that $F \sqcup G$ is undefined.

Unification: definition

Unification ($F \sqcup G$)

The unification of two feature structures F and G is (if it exists) the smallest feature structure that is subsumed by both F and G.

Subsumption ($F_1 \sqsubseteq F_2$)

A feature structure F_1 subsumes ($\sqsubseteq$) another feature structure F_2, iff all the information that is contained in F_1 is also contained in F_2.

That is, (if it exists) $F \sqcup G$ is the feature structure with the following three properties:

(1) $F \sqsubseteq (F \sqcup G)$

(2) $G \sqsubseteq (F \sqcup G)$

(3) If H is a feature structure such that $F \sqsubseteq H$ and $G \sqsubseteq H$, then $(F \sqcup G) \sqsubseteq H$. If there is no smallest feature structure that is subsumed by both F and G, then we say that $F \sqcup G$ is undefined.

For any feature structure F: $F \sqcup [\,] = [\,] \sqcup F = F$

⇒ The empty feature structure is the **identity element** for unification

Reentrancies

- the paths that both lead to the same node $\Rightarrow$ to the same value

Reentrancies

- the paths that both lead to the same node $\Rightarrow$ to the same value
  - $\Rightarrow$ hence, they share that value

Reentrancies

- the paths that both lead to the same node ⇒ to the same value
  - ⇒ hence, they share that value
- this property of sharing value(s) is called **reentrancy**

Reentrancies

- the paths that both lead to the same node ⇒ to the same value
  - ⇒ hence, they share that value
- this property of sharing value(s) is called **reentrancy**
- in AVMs: expressed by coindexing the shared values (boxed numbers)

Reentrancies

- the paths that both lead to the same node $\Rightarrow$ to the same value
  - $\Rightarrow$ hence, they share that value
- this property of sharing value(s) is called **reentrancy**
- in AVMs: expressed by coindexing the shared values (boxed numbers)
- within feature structures:

$$\begin{bmatrix} \textsc{attr}_1 & \boxed{1} \\ \textsc{attr}_2 & \boxed{1} \end{bmatrix} \qquad \begin{bmatrix} \textsc{attr}_1 & \boxed{1}\text{val}_1 \\ \textsc{attr}_2 & \boxed{1} \end{bmatrix} \qquad \begin{bmatrix} \textsc{attr}_1 & \boxed{1}\text{val}_1 \\ \textsc{attr}_2 & \begin{bmatrix} \textsc{attr}_3 & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Reentrancies

- the paths that both lead to the same node ⇒ to the same value
  - ⇒ hence, they share that value
- this property of sharing value(s) is called **reentrancy**
- in AVMs: expressed by coindexing the shared values (boxed numbers)
- within feature structures:

$$\begin{bmatrix} \text{ATTR}_1 & \boxed{1} \\ \text{ATTR}_2 & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{ATTR}_1 & \boxed{1}\text{val}_1 \\ \text{ATTR}_2 & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{ATTR}_1 & \boxed{1}\text{val}_1 \\ \text{ATTR}_2 & \begin{bmatrix} \text{ATTR}_3 & \boxed{1} \end{bmatrix} \end{bmatrix}$$

- between feature structures (in a tree):

Reentrancies

- the paths that both lead to the same node ⇒ to the same value
  - ⇒ hence, they share that value
- this property of sharing value(s) is called **reentrancy**
- in AVMs: expressed by coindexing the shared values (boxed numbers)
- within feature structures:

$$\begin{bmatrix} \text{ATTR}_1 & \boxed{1} \\ \text{ATTR}_2 & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{ATTR}_1 & \boxed{1}\text{val}_1 \\ \text{ATTR}_2 & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{ATTR}_1 & \boxed{1}\text{val}_1 \\ \text{ATTR}_2 & \begin{bmatrix} \text{ATTR}_3 & \boxed{1} \end{bmatrix} \end{bmatrix}$$

- between feature structures (in a tree):

$$\begin{bmatrix} \text{ATTR}_1 & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{ATTR}_1 & \boxed{1} \end{bmatrix}$$

TAG with feature structures

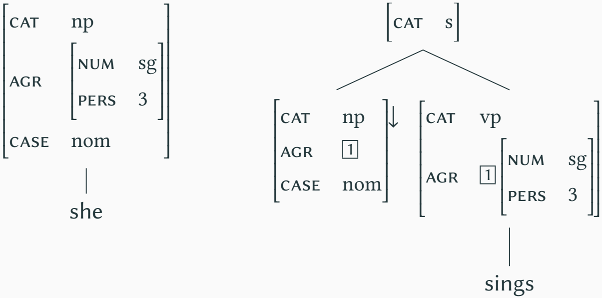
- **idea:** use feature structures as non-terminal nodes

TAG with feature structures

- **idea:** use feature structures as non-terminal nodes
- at substitution/adjunction the feature structures of the participating nodes are *unified*

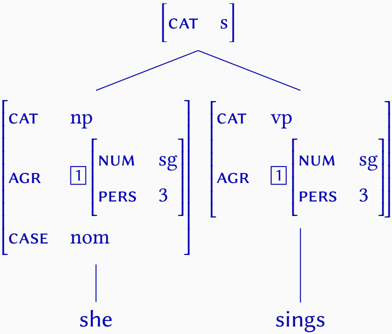
TAG with feature structures

- **idea:** use feature structures as non-terminal nodes
- at substitution/adjunction the feature structures of the participating nodes are *unified*



TAG with feature structures

- **idea:** use feature structures as non-terminal nodes
- at substitution/adjunction the feature structures of the participating nodes are *unified*



FTAG

- Feature-structure based TAG [Vijay-Shanker & Joshi 1988]

FTAG

- Feature-structure based TAG [Vijay-Shanker & Joshi 1988]
- annotate each node with two feature structures

FTAG

- Feature-structure based TAG [Vijay-Shanker & Joshi 1988]
- annotate each node with two feature structures
- split the feature structures → for adjunction
  - top features: the relation of the node to the tree above it
  - bottom features: the relation of the node to the tree below it

FTAG

- Feature-structure based TAG [Vijay-Shanker & Joshi 1988]
- annotate each node with two feature structures
- split the feature structures → for adjunction
  - top features: the relation of the node to the tree above it
  - bottom features: the relation of the node to the tree below it

FTAG description of node η

1. The relation of η to its supertree is called features structure t_η.
2. The relation of η to its descendants is called features structure b_η.

FTAG

- Feature-structure based TAG [Vijay-Shanker & Joshi 1988]
- annotate each node with two feature structures
- split the feature structures → for adjunction
  - top features: the relation of the node to the tree above it
  - bottom features: the relation of the node to the tree below it

FTAG description of node η

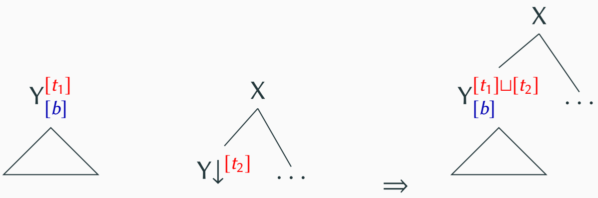
1. The relation of η to its supertree is called features structure t_η.
2. The relation of η to its descendants is called features structure b_η.

- at the final derived tree (i.e., after all substitutions/adjunctions) top and bottom features are unified for all nodes

FTAG: Substitution

Substitution in FTAG

The top features of the root of the tree to substitute unify with the top features of the substitution node.

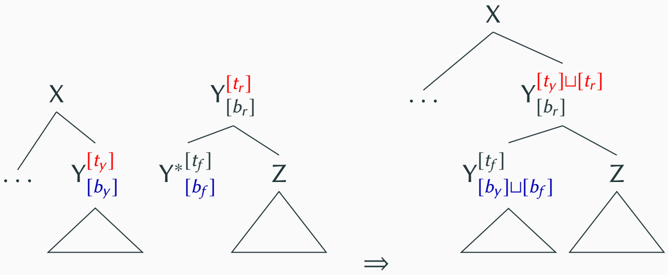


- substitution nodes (Y↓) have only top features

FTAG: Adjunction

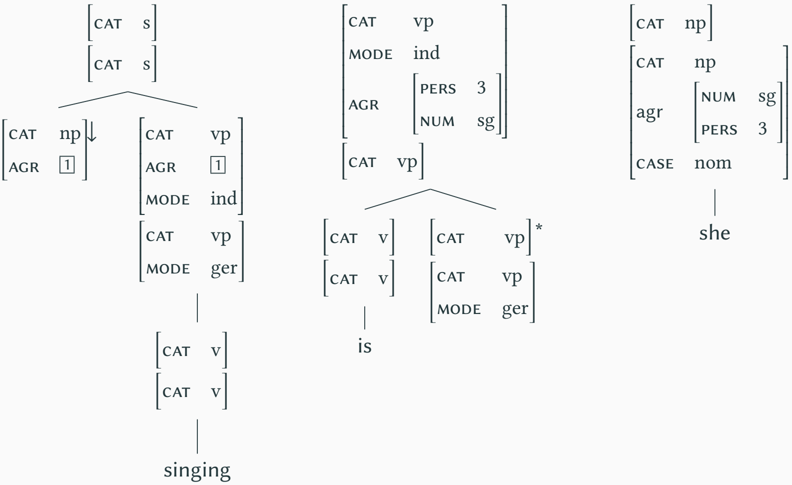
Adjunction in FTAG

The top features of the root of the auxiliary tree unify with the top features of the adjunction node, and the bottom features of the footnode of the auxiliary tree unify with the bottom features of the adjunction node.

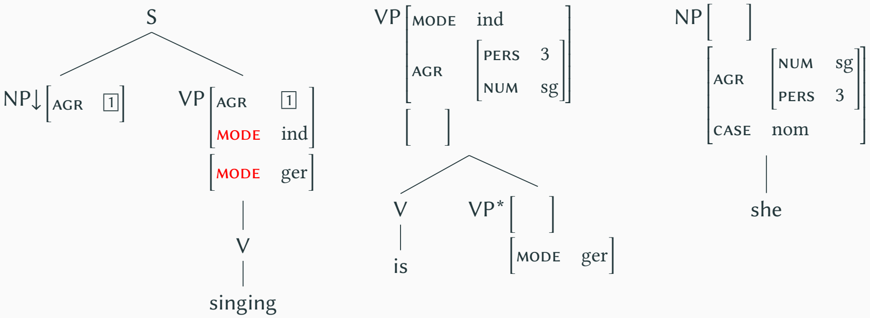


FTAG Example: *She is singing.*

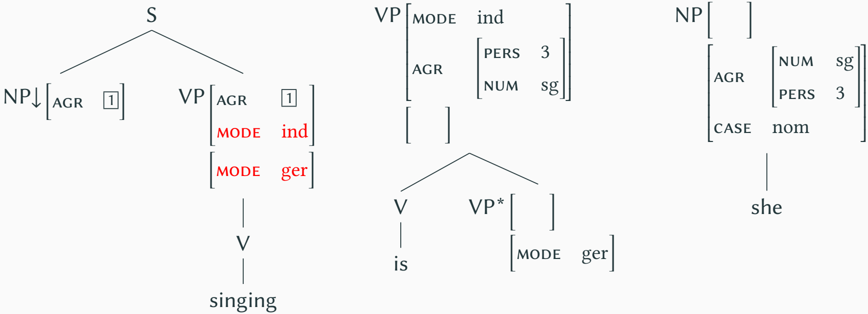
Obligatory adjunction: feature mismatch between top and bottom



FTAG Example: *She is singing.*

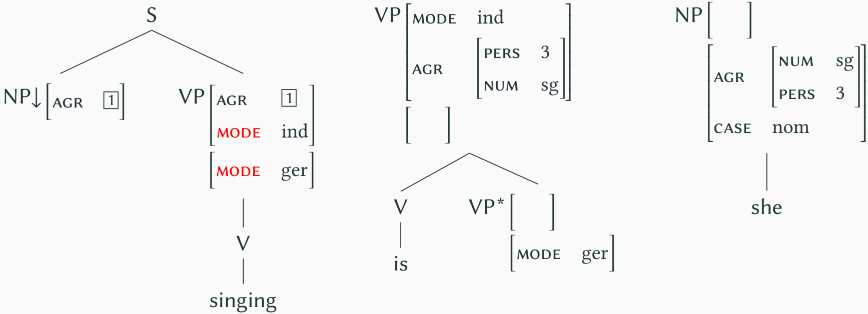


FTAG Example: *She is singing.*



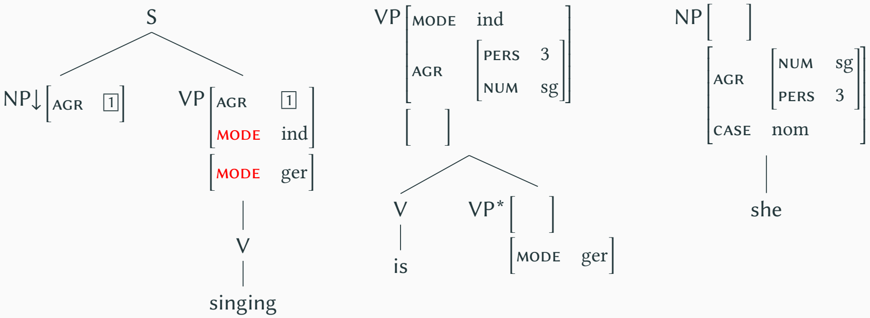
- feature mismatch at the VP node ⇒ adjunction at VP is obligatory

FTAG Example: *She is singing.*



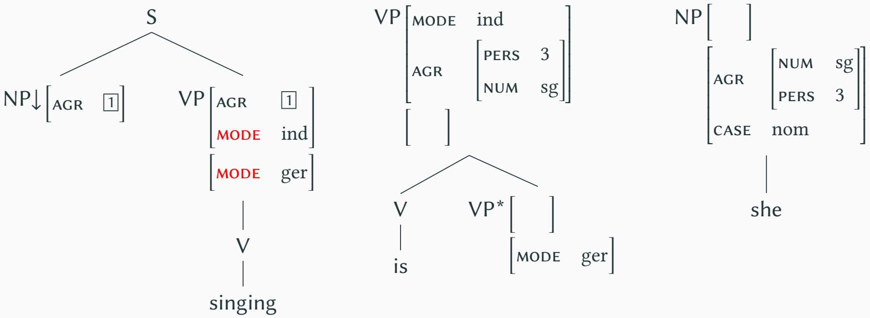
- feature mismatch at the VP node ⇒ adjunction at VP is obligatory
- at adjunction of *is* and substitution of *she*:

FTAG Example: *She is singing.*



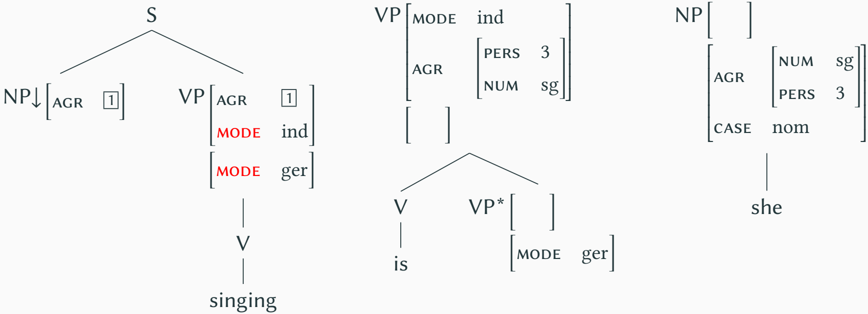
- feature mismatch at the VP node ⇒ adjunction at VP is obligatory
- at adjunction of *is* and substitution of *she*:
    - top feature of the root node of *is* unifies with the top feature of the VP node of *singing*

FTAG Example: *She is singing.*



- feature mismatch at the VP node ⇒ adjunction at VP is obligatory
- at adjunction of *is* and substitution of *she*:
    - top feature of the root node of *is* unifies with the top feature of the VP node of *singing*
    - bottom feature of the footnode of *is* unifies with the bottom feature of the VP node of *singing*

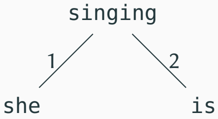
FTAG Example: *She is singing.*

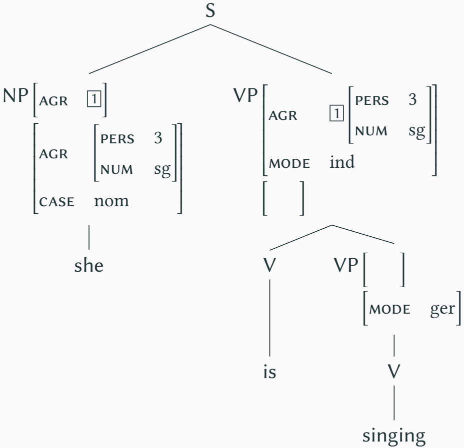


- feature mismatch at the VP node ⇒ adjunction at VP is obligatory
- at adjunction of *is* and substitution of *she*:
  - top feature of the root node of *is* unifies with the top feature of the VP node of *singing*
  - bottom feature of the footnode of *is* unifies with the bottom feature of the VP node of *singing*
  - top feature of *she* unifies with the top feature of the NP node of *singing*

FTAG Example: *She is singing.*

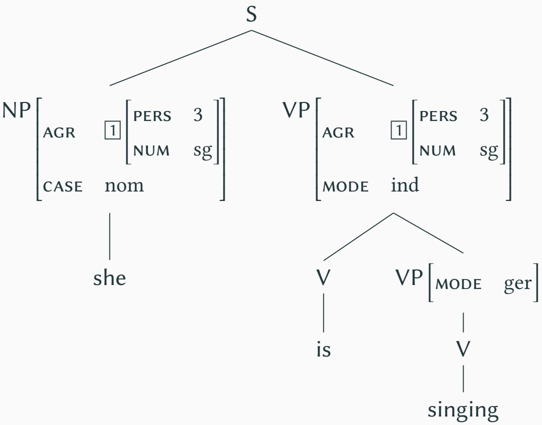
derivation tree:





FTAG example: *She is singing.*

at the final derived tree (after all substitutions/adjunctions) the top and bottom feature of each node unify:



Case assignment

- nouns carry the case, which is ‘checked’

Case assignment

- nouns carry the case, which is ‘checked’
- noun case is checked against the case value assigned by the verb during the unification

Case assignment

- nouns carry the case, which is ‘checked’
- noun case is checked against the case value assigned by the verb during the unification
- features of case-assignment:

Case assignment

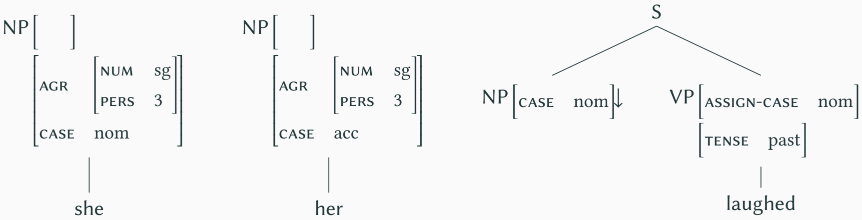
- nouns carry the case, which is ‘checked’
- noun case is checked against the case value assigned by the verb during the unification
- features of case-assignment:
    - ⟨case⟩ with values: nom | acc | gen | none
      ⇒ Ns, NPs

Case assignment

- nouns carry the case, which is ‘checked’
- noun case is checked against the case value assigned by the verb during the unification
- features of case-assignment:
    - ⟨case⟩ with values: nom | acc | gen | none
      ⇒ Ns, NPs
    - ⟨assign-case⟩ with values: nom | acc | none
      ⇒ case assigners (prepositions, verbs) and S, VP, PP nodes that dominate them

Case assignment

- nouns carry the case, which is ‘checked’
- noun case is checked against the case value assigned by the verb during the unification
- features of case-assignment:
    - ⟨case⟩ with values: nom | acc | gen | none
      ⇒ Ns, NPs
    - ⟨assign-case⟩ with values: nom | acc | none
      ⇒ case assigners (prepositions, verbs) and S, VP, PP nodes that dominate them



The XTAG-project

- was located at the University of Pennsylvania (ca. 1988-2001)
  - **grammar** (set of tree templates/families)
  - **tools** (browser, editor, parser, …)

The XTAG-project

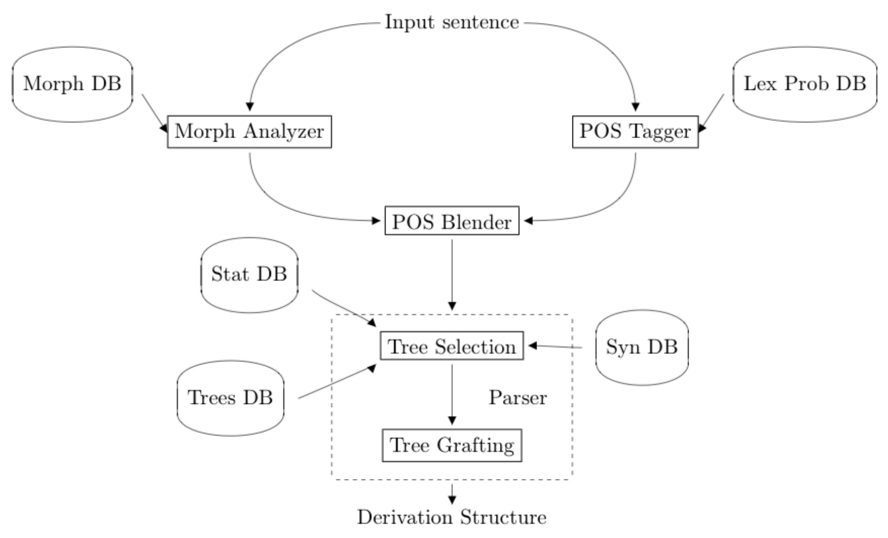
- was located at the University of Pennsylvania (ca. 1988-2001)
  - **grammar** (set of tree templates/families)
  - **tools** (browser, editor, parser, …)
- URL: `http://www.cis.upenn.edu/~xtag/`

The XTAG-project

- was located at the University of Pennsylvania (ca. 1988-2001)
  - **grammar** (set of tree templates/families)
  - **tools** (browser, editor, parser, …)
- URL: `http://www.cis.upenn.edu/~xtag/`
- the architecture of the XTAG-grammar

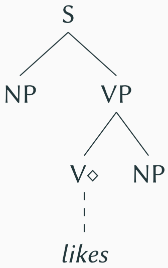
Morph Database	inflected form → root form, POS, inflectional information
Syntactic Database	root form, POS → list of tree templates or or tree families, list of feature equations
Tree Database	list of tree templates and tree families

The architecture of the XTAG-grammar



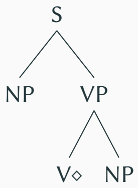
Lexical insertion

- drawing an edge between the lexical anchor and the lexical insertion site
- prior to substitution and adjunction
- the feature structures of the **lexical anchor** and the **insertion site** unify



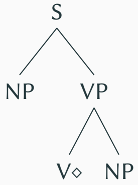
The architecture of the XTAG-grammar

tree template for the declarative transitive verb (α`nx0Vnx1`):



The architecture of the XTAG-grammar

tree template for the declarative transitive verb (α`nx0Vnx1`):



A tree family

- is a set of tree templates
- represents a subcategorization frame, and

  contains all syntactic configurations the subcategorization frame can be realized in

Example: α`nx0Vnx1` $\in$ `Tnx0Vnx1`

Tree families

Example tree families

- intransitive: `Tnx0V`
  tree templates: base tree, wh-moved subject, imperative, determiner gerund, ... etc.

Tree families

Example tree families

- intransitive: `Tnx0V`
  tree templates: base tree, wh-moved subject, imperative, determiner gerund, ... etc.
- transitive: `Tnx0Vnx1`
  tree templates: base tree, wh-moved subject, wh-moved object, imperative, determiner gerund, passive with *by*, ... etc.

Tree families

Example tree families

- intransitive: `Tnx0V`
  tree templates: base tree, wh-moved subject, imperative, determiner gerund, ... etc.
- transitive: `Tnx0Vnx1`
  tree templates: base tree, wh-moved subject, wh-moved object, imperative, determiner gerund, passive with *by*, ... etc.

Some figures

[Prolo 2002]

subcat. group	no. of families	no. of trees
intransitive	1	12
transitive	1	39
ditransitive	1	46
light verb constr.	2	53
⋮	⋮	⋮
TOTAL:	57	1008

References

Abeillé, Anne and Owen Rambow. 2000. Tree adjoining grammar: An overview. In A. Abeillé and O. Rambow (eds). *Tree adjoining grammars: Formalisms, linguistic analyses and processing* Vol. 107 CSLI Lecture Notes. Stanford: CSLI Publications. 1–68.

Frank, Robert. 2002. *Phrase structure composition and syntactic dependencies*. Cambridge: MIT Press.

Joshi, Aravind K. and Yves Schabes. 1991. Tree-Adjoining Grammars and lexicalized grammars. *Tech. Rep. MS-CIS-91-22 Department of Computer and Information Science*. University of Pennsylvania.

Joshi, Aravind K. 2004. Starting with complex primitives pays off: complicate locally, simplify globally. *Cognitive Science* 28. 637–668.

Prolo, Carlos A. 2002. Generating the XTAG english grammar using metarules. In *Coling 2002: The 19th international conference on computational linguistics*.

Schabes, Yves and Aravind K. Joshi. 1990. Parsing with lexicalized tree adjoining grammar. *Tech. Rep. MS-CIS-90-11 Department of Computer and Information Science*. University of Pennsylvania.

Vijay-Shanker, K. and Aravind K. Joshi. 1988. Feature Structures Based Tree Adjoining Grammars. In *COLING Budapest 1988*, Volume 2: International Conference on Computational Linguistics.

XTAG Research Group. 2001. A Lexicalized Tree Adjoining Grammar for English. *Tech. rep. Institute for Research in Cognitive Science*. University of Pennsylvania.