Transforming Dependency Structures to LTAG Derivation Trees

13th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+13)

Caio Corro Joseph Le Roux

September 1, 2017

Laboratoire Informatique de Paris Nord (LIPN), Université Paris 13 (France), CNRS UMR 7030

Introduction

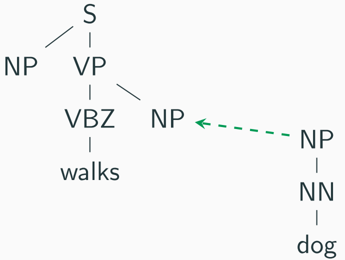
Lexicalized Tree Adjoining Grammar (LTAG)

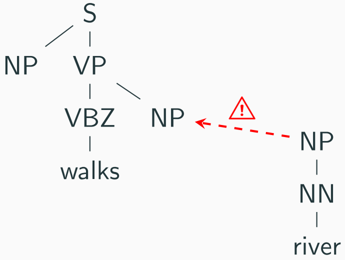
Why LTAGs?

- Constituency structure
- Linguistically plausible
- Built-in bi-lexical relations
- Deep syntax

Weighted grammars

- Disambiguation/Preference
- Robustness:
    - Unknown words
    - Errors





LTAG parsing

CKY-type algorithm

- Deduction-rule based
- Bottom-up

Complexity

$\mathcal{O}(n^6 \max(n, g) g t)$:

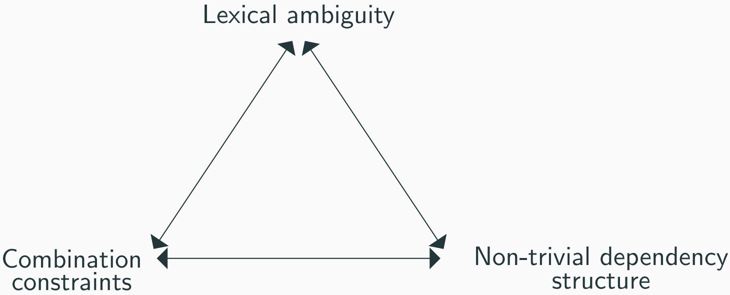
n: sentence length

t: maximum number of nodes in an elementary tree

g: maximum ambiguity

$\Rightarrow \mathcal{O}(n^7)$ asymptotically w.r.t. the sentence length [Eisner et al., 2000]

LTAG parsing problem

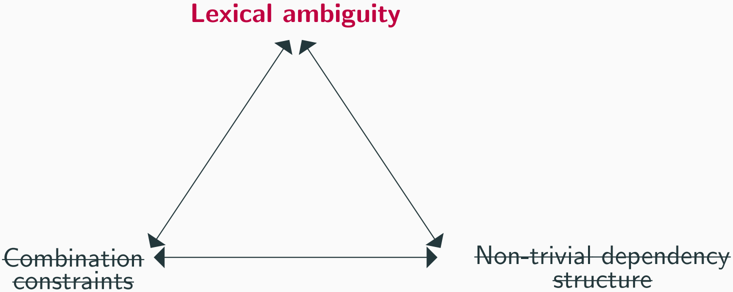


Lexical ambiguity

Combination constraints

Non-trivial dependency structure

Supertagging approach (1)



Lexical ambiguity

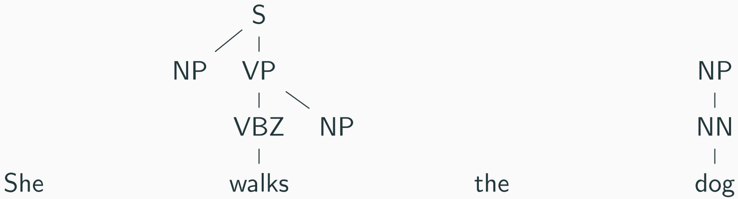
~~Combination constraints~~

~~Non-trivial dependency structure~~

Supertagging approach (2)

She walks the dog

Supertagging approach (2)



S

NP VP

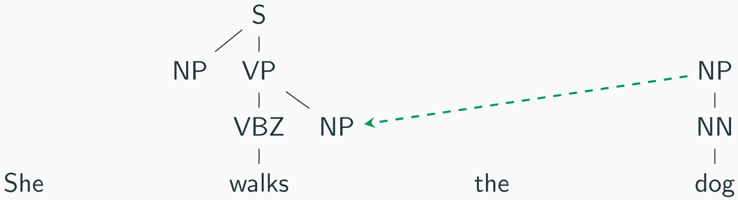
VBZ NP

NP

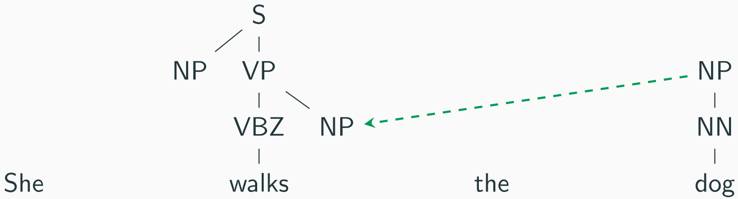
NN

She walks the dog

Supertagging approach (2)

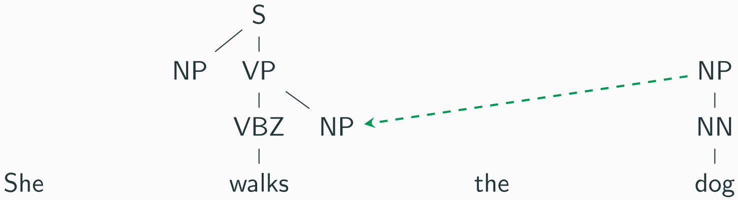


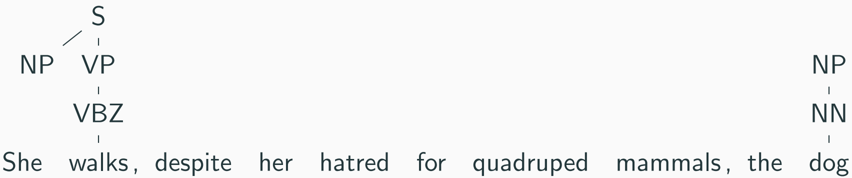
Supertagging approach (2)



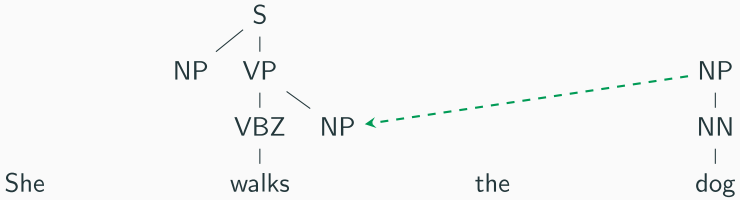
She walks , despite her hatred for quadruped mammals , the dog

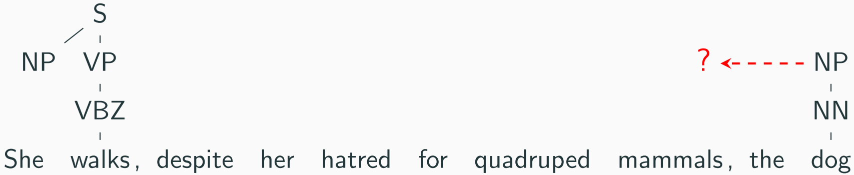
Supertagging approach (2)





Supertagging approach (2)





Supertagging approach (3)

Pipeline

1. Supertagging
2. Constraint LTAG parsing

Downsides

- Long distance relationship
- 2nd step complexity: $\mathcal{O}(n^7 t)$
  $\Rightarrow$ No lexical ambiguity

Phrase structure tree VS Dependency tree

> "…One should always distinguish the type of representation […]
> from the content of the representation..." [Rambow, 2010]

Syntactic content

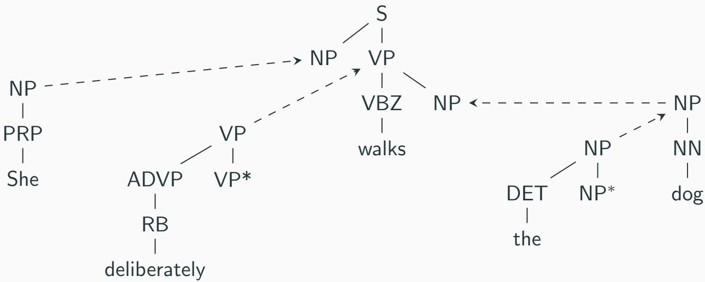
- Syntactic dependency
- Syntactic phrase/constituency structure

Representation types

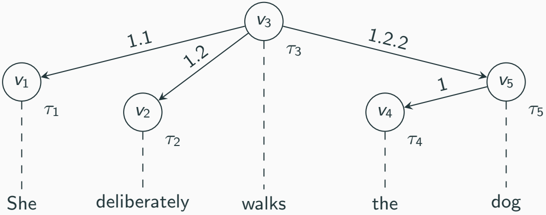
- Dependency tree
- Hierarchy structure tree

⇒ Syntactic phrase-structure parsing as a dependency structure parsing task

LTAG derivation tree



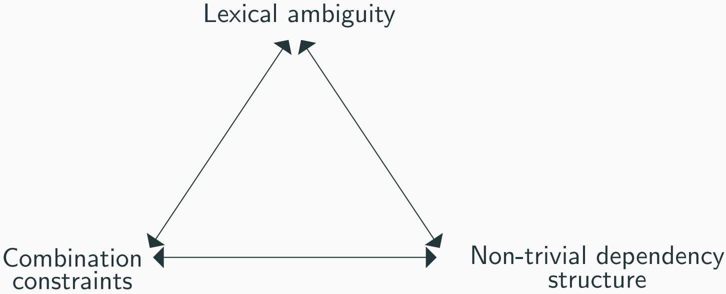
Bottom-up construction of the syntactic phrase structure



Representation alternative: the LTAG derivation tree is a dependency tree

[Rambow et al., 1997]

Proposed approach (1)

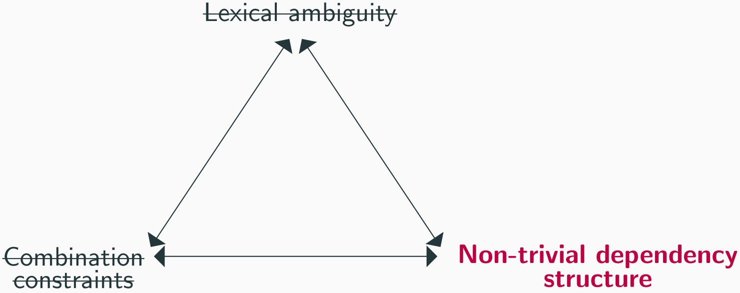


Lexical ambiguity

Combination constraints

Non-trivial dependency structure

Proposed approach (1)

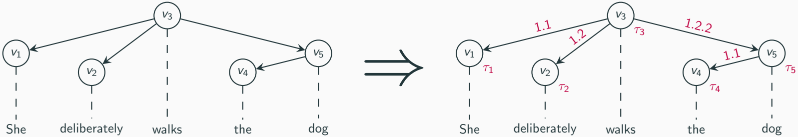


~~Lexical ambiguity~~

~~Combination constraints~~

Non-trivial dependency structure

Proposed approach (2)



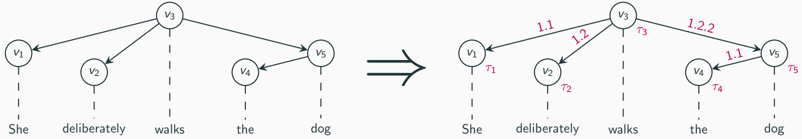
Alternative pipeline

1. Bi-lexical dependency parsing: long distance relationships
2. LTAG parse labeler

Downsides

- 1st step complexity: $\mathcal{O}(n^7)$ [Gómez-Rodríguez et al., 2009]
- 2nd step complexity?

Proposed approach (2)



Alternative pipeline

1. Bi-lexical dependency parsing: long distance relationships
2. LTAG parse labeler

Downsides

- 1st step complexity: $\mathcal{O}(n^7)$ [Gómez-Rodríguez et al., 2009]
  - $\Rightarrow$ Efficient decoding in practice via Lagrangian relaxation [Corro et al., 2016]
- 2nd step complexity?
  - $\Rightarrow$ This contribution!

Table of contents

Characterization of LTAG derivation trees

LTAG derivation trees

Structural properties **[Bodirsky et al., 2005]**

- Arborescence (directed tree)
- 2-bounded block degree
- Well-nestedness

2-bounded block degree

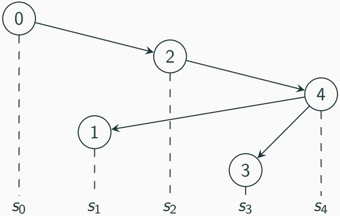
- Maximum 1 gap in the yield of a sub-arborescence
  ⇒ Due to wrapping adjunction

Well-nestedness

- Sub-arborescences must not interleave (not used in this presentation)

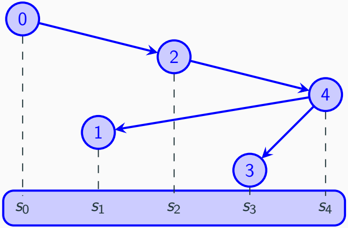
Yield

Yield of a vertex *v*: set of all nodes reachable from *v*



Yield

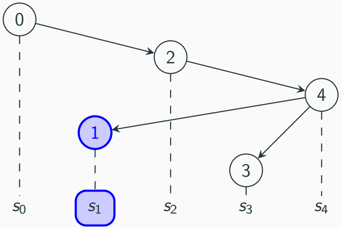
Yield of a vertex *v*: set of all nodes reachable from *v*



$\mathit{Yield}(0) = \{0, 1, 2, 3, 4\}$

Yield

Yield of a vertex v: set of all nodes reachable from v

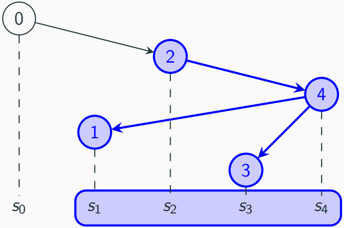


$Yield(0) = \{0, 1, 2, 3, 4\}$

$Yield(1) = \{1\}$

Yield

Yield of a vertex *v*: set of all nodes reachable from *v*



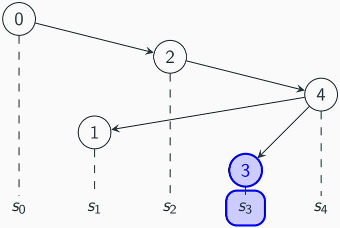
$Yield(0) = \{0, 1, 2, 3, 4\}$

$Yield(1) = \{1\}$

$Yield(2) = \{1, 2, 3, 4\}$

Yield

Yield of a vertex v: set of all nodes reachable from v



$Yield(0) = \{0, 1, 2, 3, 4\}$

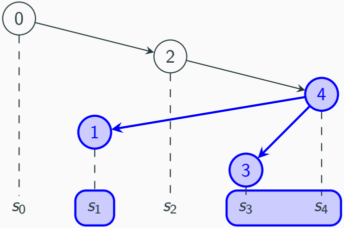
$Yield(1) = \{1\}$

$Yield(2) = \{1, 2, 3, 4\}$

$Yield(3) = \{3\}$

Yield

Yield of a vertex v: set of all nodes reachable from v



$Yield(0) = \{0, 1, 2, 3, 4\}$

$Yield(1) = \{1\}$

$Yield(2) = \{1, 2, 3, 4\}$

$Yield(3) = \{3\}$

$Yield(4) = \{1, 3, 4\}$

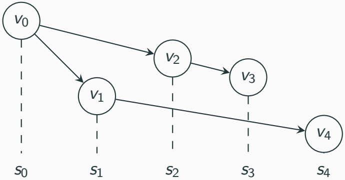
2-bounded block degree

Bound degree

- Vertex: number of contiguous intervals described by its yield
- Arborescence: the maximal block degree of its vertices

2 Bounded degree arborescence

- Arborescence with a bound degree less or equal to 2



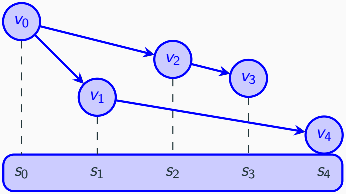
2-bounded block degree

Bound degree

- Vertex: number of contiguous intervals described by its yield
- Arborescence: the maximal block degree of its vertices

2 Bounded degree arborescence

- Arborescence with a bound degree less or equal to 2



$Yield(0) = [0 \ldots 4]$ $\quad$ $BD(0) = 1$

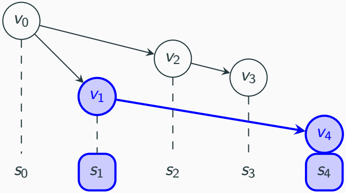
2-bounded block degree

Bound degree

- Vertex: number of contiguous intervals described by its yield
- Arborescence: the maximal block degree of its vertices

2 Bounded degree arborescence

- Arborescence with a bound degree less or equal to 2



$Yield(0) = [0 \ldots 4]$ $BD(0) = 1$

$Yield(1) = [1] \cup [4]$ $BD(1) = 2$

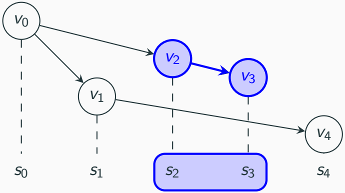
2-bounded block degree

Bound degree

- Vertex: number of contiguous intervals described by its yield
- Arborescence: the maximal block degree of its vertices

2 Bounded degree arborescence

- Arborescence with a bound degree less or equal to 2



$Yield(0) = [0 \dots 4]$ $\quad BD(0) = 1$

$Yield(1) = [1] \cup [4]$ $\quad BD(1) = 2$

$Yield(2) = [2 \dots 3]$ $\quad BD(2) = 1$

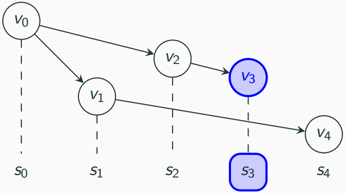
2-bounded block degree

Bound degree

- Vertex: number of contiguous intervals described by its yield
- Arborescence: the maximal block degree of its vertices

2 Bounded degree arborescence

- Arborescence with a bound degree less or equal to 2



$Yield(0) = [0 \dots 4]$ $\qquad BD(0) = 1$

$Yield(1) = [1] \cup [4]$ $\qquad BD(1) = 2$

$Yield(2) = [2 \dots 3]$ $\qquad BD(2) = 1$

$Yield(3) = [3]$ $\qquad BD(3) = 1$

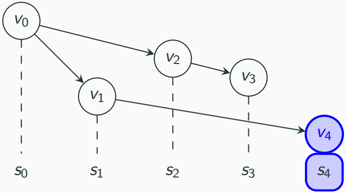
2-bounded block degree

Bound degree

- Vertex: number of contiguous intervals described by its yield
- Arborescence: the maximal block degree of its vertices

2 Bounded degree arborescence

- Arborescence with a bound degree less or equal to 2



$Yield(0) = [0 \ldots 4]$	$BD(0) = 1$
$Yield(1) = [1] \cup [4]$	$BD(1) = 2$
$Yield(2) = [2 \ldots 3]$	$BD(2) = 1$
$Yield(3) = [3]$	$BD(3) = 1$
$Yield(4) = [4]$	$BD(4) = 1$

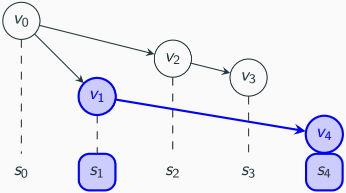
2-bounded block degree

Bound degree

- Vertex: number of contiguous intervals described by its yield
- Arborescence: the maximal block degree of its vertices

2 Bounded degree arborescence

- Arborescence with a bound degree less or equal to 2



$Yield(0) = [0 \dots 4]$ $BD(0) = 1$

$Yield(1) = [1] \cup [4]$ $BD(1) = 2$

$Yield(2) = [2 \dots 3]$ $BD(2) = 1$

$Yield(3) = [3]$ $BD(3) = 1$

$Yield(4) = [4]$ $BD(4) = 1$

Intuition

- Auxiliary tree anchored at s_1 adjoined via wrapping adjunction
- Anchors s_2 and s_3 attached below the foot node

Parsing

Dynamic programming **[Gómez-Rodríguez et al., 2009]**

- Complexity: $\mathcal{O}(n^7)$, intractable on long sentences

⇒ Asymptotically equivalent to LTAG parsing!

Combinatorial optimization **[Corro et al., 2016]**

- Complexity: exponential
- Practically: fast

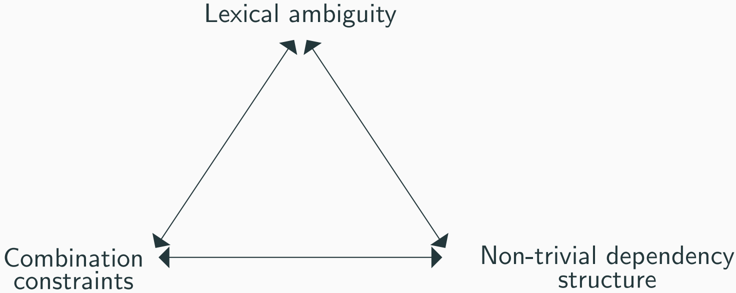
⇒ "Simple" optimization problem as there is no constraint on combination operations

Intuition

1. Non-trivial dependency structure parsing tackled via combinatorial optimization
2. Complexity of parse tree labeling?

Outline of the algorithm

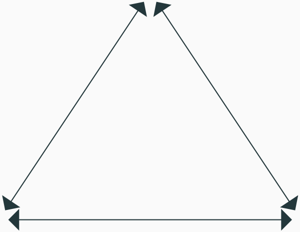
Parse tree labeling


Lexical ambiguity

Combination constraints

Non-trivial dependency structure

Parse tree labeling

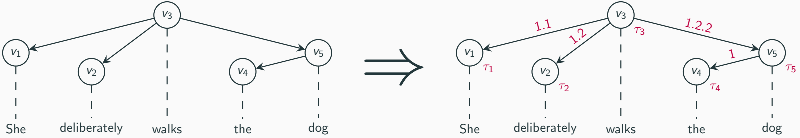


Lexical ambiguity

Combination constraints

~~Non-trivial dependency structure~~

Deduction system



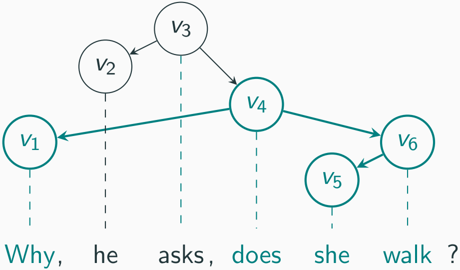
Dynamic program

- Deduction rule
- Agenda

Bottom-up

1. Dependency tree: words considered after its modifiers
2. Elementary tree: non-terminal considered after its children

Key idea: extract information from the dependency structure



Notation	Value
$(v_4)_{\Leftarrow}$	v_1
$(v_4)_{\Rightarrow}$	v_6
$(v_4)_{\leftarrow}$	v_2
$(v_4)_{\rightarrow}$	v_3
$(v_4)_{\uparrow}$	v_3

Information about v_4

- Parent: v_3
- Yield span: $[1, 6]$
- Gap span: $[2, 3]$

Key idea: no integer span

Main difference

Vertices are used to define spans instead of integers
$\Rightarrow$ combination rule constrained by arcs between vertices

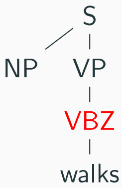
Standard LTAG parser items (CKY)	**Our parser items**
$[h, \tau, p, c, i, j, k, l]$ with:	$[v_h, \tau, p, c, b_l, b_r]$ with:
h: anchor word index	v_h: vertex (anchor word)
τ: elementary tree	τ: elementary tree
p: gorn address	p: gorn address
c: combination flag	c: combination flag
i, l: yield span (integers)	b_l: left boundary (vertex)
j, k: gap span (integers)	b_r: right boundary (vertex)

Moving

Let's start with something simple... :-)

Move unary:

$[v_h, \tau, 1.2.1, \top, b_l, b_r]$

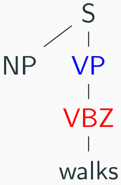


Moving

Let's start with something simple... :-)

Move unary:

$$\frac{[v_h, \tau, 1.2.1, \top, b_l, b_r]}{[v_h, \tau, 1.2, \bot, b_l, b_r]} \; (p \cdot 2) \notin \tau$$

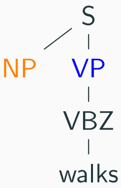


Moving

Let's start with something simple... :-)

Move binary:

$[v_h, \tau, 1.1, \top, b_{l1}, b_{r1}]$ $[v_h, \tau, 1.2, \top, b_{l2}, b_{r2}]$



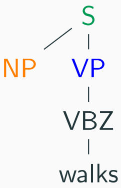
Moving

Let's start with something simple... :-)

Move binary:

$$\frac{[v_h, \tau, 1.1, \top, b_{l1}, b_{r1}] \qquad [v_h, \tau, 1.2, \top, b_{l2}, b_{r2}]}{[v_h, \tau, 1, \bot, b_{l1}, b_{r2}]} \; (b_{r1})_{\Rightarrow} + 1 = (b_{l2})_{\Leftarrow}$$

⇒ Similar to LTAG parsing but with constraint on boundary vertices

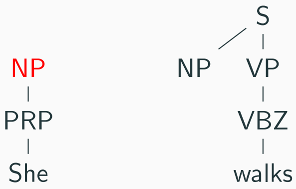


Substitution

And now let's see something nice! O_o

Substitute:

$[v_m, \tau', 1, \top, b_l, b_r]$



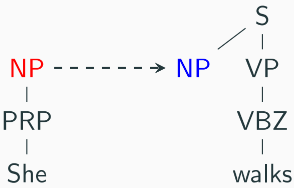
Substitution

And now let's see something nice! O_o

Substitute:

$$\frac{[v_m, \tau', 1, \top, b_l, b_r]}{[v_h, \tau, p, \top, v_m, v_m]} \; (v_m)_{\leftarrow} = -, f_{SS}(\tau, p, \tau')$$

⇒ Fixed boundaries for the antecedent by the dependency tree



Substitution

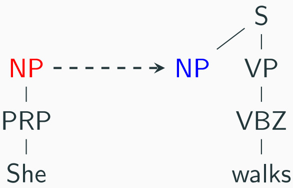
And now let's see something nice! O_o

Substitute:

$$\frac{[v_m, \tau']}{[v_h, \tau, p, \top, v_m, v_m]} \; (v_m)_{\leftarrow} = -, f_{SS}(\tau, p, \tau')$$

$\Rightarrow$ v_h fixed by the dependency tree

$\Rightarrow$ Number of applications linearly bounded

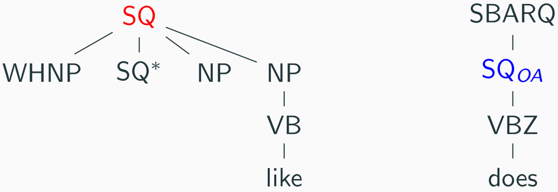


Wrapping adjunction

But for a more complicated operation? :/

Wrapping adjoin:

$[v_m, \tau', 1, \top, b_{l1}, b_{r1}]$ $[v_h, \tau, p, \perp, b_{l2}, b_{r2}]$



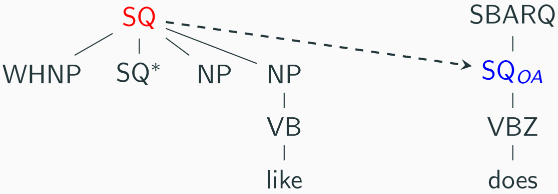
Wrapping adjunction

But for a more complicated operation? :/

Wrapping adjoin:

$$\frac{[v_m, \tau', 1, \top, b_{l1}, b_{r1}] \qquad [v_h, \tau, p, \bot, b_{l2}, b_{r2}]}{[v_h, \tau, p, \top, v_m, v_m]} \; f_{SA}(\tau, p, \tau')$$

⇒ Boundaries of the left antecedent are fixed (similarly to substitution)



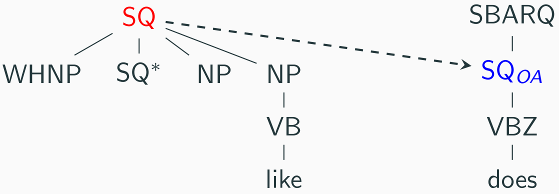
Wrapping adjunction

But for a more complicated operation? :/

Wrapping adjoin:

$$\frac{[v_m, \tau'] \qquad [v_h, \tau, p, \bot, b_l, b_r]}{[v_h, \tau, p, \top, v_m, v_m]} \; f_{SA}(\tau, p, \tau')$$

⇒ Gap filled with boundaries of the right antecedent?



Wrapping adjunction

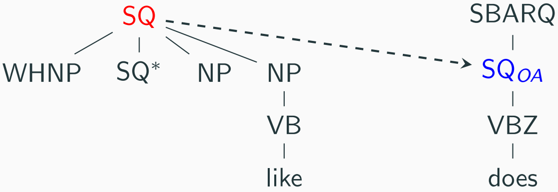
But for a more complicated operation? :/

Wrapping adjoin:

$$\frac{[v_m, \tau'] \qquad [v_h, \tau, p, \bot, b_l, b_r]}{[v_h, \tau, p, \top, v_m, v_m]} \; (v_m)_{\leftarrow} = (b_l)_{\Leftarrow}, (v_m)_{\rightarrow} = (b_r)_{\Rightarrow}, f_{SA}(\tau, p, \tau')$$

$\Rightarrow$ v_h fixed by the dependency tree

$\Rightarrow$ Number of applications linearly bounded, again



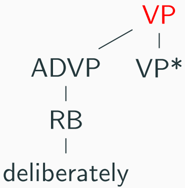
Left/Right adjunction

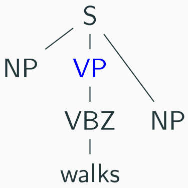
Wait, we don't know the gap boundaries for left/right adjunctions! :'(

Left adjoin:

$[v_m, \tau', 1, \top, b_{l1}, b_{r1}]$ $[v_h, \tau, p, \bot, b_{l2}, b_{r2}]$

$\Rightarrow$ Right limit of the gap b_{r1} unknown in the dependency tree





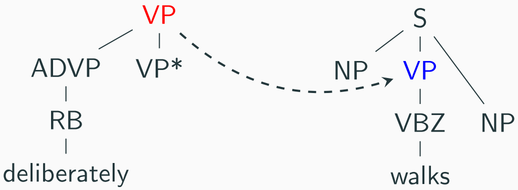
Left/Right adjunction

Wait, we don't know the gap boundaries for left/right adjunctions! :'(

Left adjoin:

$[v_m, \tau', 1, \top, b_{l1}, -]$ $[v_h, \tau, p, \bot, b_{l2}, b_{r2}]$

⇒ Workaround: $-$ boundary to prevent anything in the right side of the gap



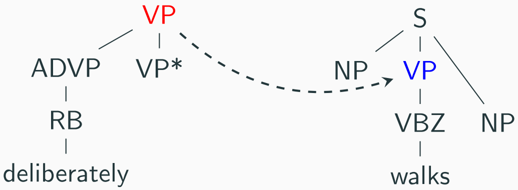
Left/Right adjunction

Wait, we don't know the gap boundaries for left/right adjunctions! :'(

Left adjoin:

$[v_m, \tau', \leftarrow]$ $[v_h, \tau, p, \perp, b_l, b_r]$

$\Rightarrow$ Left antecedent fixed by the dependency tree



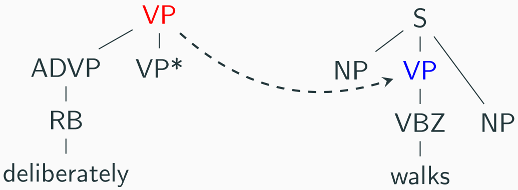
Left/Right adjunction

Wait, we don't know the gap boundaries for left/right adjunctions! :'(

Left adjoin:

$$\frac{[v_m, \tau', \leftarrow] \qquad [v_h, \tau, p, \bot, b_l, b_r]}{[v_h, \tau, p, \top, v_m, b_r]} \; (v_m)_{\Rightarrow} = (b_l)_{\Leftarrow} - 1, f_{SA}(\tau, p, \tau')$$

⇒ Is the number of applications linearly bounded?
(yes, proof in the paper)



Complexity

Complexity

Move binary:

$$\frac{[v_h, \tau, 1.1, \top, b_{l1}, b_{r1}] \qquad [v_h, \tau, 1.2, \top, b_{l2}, b_{r2}]}{[v_h, \tau, 1, \bot, b_{l1}, b_{r2}]} \; (b_{r1})_{\Rightarrow} + 1 = (b_{l2})_{\Leftarrow}$$

Proof intuition

3 boundaries $\Rightarrow \mathcal{O}(n^3)$?

Complexity

Move binary:

$$\frac{[v_h, \tau, 1.1, \top, b_{l1}, b_{r1}] \qquad [v_h, \tau, 1.2, \top, b_{l2}, b_{r2}]}{[v_h, \tau, 1, \bot, b_{l1}, b_{r2}]} \; (b_{r1})_{\Rightarrow} + 1 = (b_{l2})_{\Leftarrow}$$

Proof intuition

~~3 boundaries $\Rightarrow \mathcal{O}(n^3)$~~ ?

$\Rightarrow$ Bounded by the elementary tree size if no multiple adjunction

Complexity

Move binary:

$$\frac{[v_h, \tau, 1.1, \top, b_{l1}, b_{r1}] \qquad [v_h, \tau, 1.2, \top, b_{l2}, b_{r2}]}{[v_h, \tau, 1, \bot, b_{l1}, b_{r2}]} \; (b_{r1})_{\Rightarrow} + 1 = (b_{l2})_{\Leftarrow}$$

Proof intuition

~~3 boundaries $\Rightarrow \mathcal{O}(n^3)$~~ ?

$\Rightarrow$ Bounded by the elementary tree size if no multiple adjunction

Complexity

$\mathcal{O}(\min(t, n)^2 ntg)$ with:

n: sentence length

t: maximum number of nodes in an elementary tree

g: maximum ambiguity

$\Rightarrow$ Asymptotically linear w.r.t. the sentence length

Conclusion

Conclusion

Contributions

- New perspective on efficient LTAG parsing
- Linear time LTAG parse labeler

Future work

- Experimentation!
- Multiple adjunctions?
- Extension to other lexicalized formalisms:
  Lexicalized Linear Context-Free Rewriting Systems, . . .

Questions?