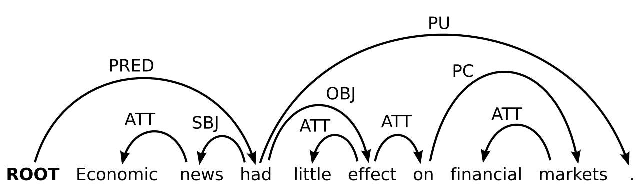
CS447: Natural Language Processing
http://courses.engr.illinois.edu/cs447

Lecture 12: Dependency Parsing; Expressive Grammars

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Dependency Parsing

A dependency parse



Dependencies are (labeled) asymmetrical binary relations between two lexical items (words).

Parsing algorithms for DG

‘Transition-based’ parsers:
learn a sequence of actions to parse sentences
Models:
State = stack of partially processed items
+ queue/buffer of remaining tokens
+ set of dependency arcs that have been found already
Transitions (actions) = add dependency arcs; stack/queue operations

‘Graph-based’ parsers:
learn a model over dependency graphs
Models:
a function (typically sum) of local attachment scores
For dependency trees, you can use a minimum spanning tree algorithm

Transition-based parsing (Nivre et al.)

Transition-based parsing: assumptions

This algorithm works for projective dependency trees.

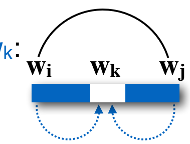
Dependency tree:

Each word has a single parent
(Each word is a dependent of [is attached to] one other word)

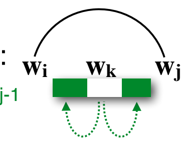
Projective dependencies:

There are no crossing dependencies.
For any i, j, k with $i < k < j$: if there is a dependency between w_i and w_j, the parent of w_k is a word w_l between (possibly including) i and j: $i \leq l \leq j$, while any child w_m of w_k has to occur between (excluding) i and j: $i<m<j$

the parent of w_k: one of $w_i \dots w_j$



any child of w_k: one of $w_{i+1} \dots w_{j-1}$



Transition-based parsing

Transition-based shift-reduce parsing processes the sentence $S = w_0w_1...w_n$ from left to right.
Unlike CKY, it constructs a **single tree**.

Notation:

w_0 is a special ROOT token.
$V_S = \{w_0, w_1, ..., w_n\}$ is the vocabulary of the sentence
R is a set of dependency relations

The parser uses three data structures:

σ: a **stack** of partially processed words $w_i \in V_S$
β: a **buffer** of remaining input words $w_i \in V_S$
A: a **set of dependency arcs** $(w_i, r, w_j) \in V_S \times R \times V_S$

Parser configurations (σ, β, A)

The **stack** $\boldsymbol{\sigma}$ is a list of partially processed words

We push and pop words onto/off of $\boldsymbol{\sigma}$.
$\boldsymbol{\sigma|w}$: $\mathbf{w}$ is on top of the stack.
Words on the stack are not (yet) attached to any other words.
Once we attach $\mathbf{w}$, $\mathbf{w}$ can't be put back onto the stack again.

The **buffer** $\boldsymbol{\beta}$ is the remaining input words

We read words from $\boldsymbol{\beta}$ (left-to-right) and push them onto σ
$\mathbf{w|\boldsymbol{\beta}}$: $\mathbf{w}$ is on top of the buffer.

The **set of arcs** $\mathbf{A}$ defines the current tree.

We can add new arcs to $\mathbf{A}$ by attaching the word on top of the stack to the word on top of the buffer, or vice versa.

Parser configurations (σ, β, A)

We start in the **initial configuration** $([w_0], [w_1, ..., w_n], \{\})$

(Root token, Input Sentence, Empty tree)

We can attach the first word (w_1) to the root token w_0,
or we can push w_1 onto the stack.
(w_0 is the only token that can't get attached to any other word)

We want to end in the **terminal configuration** $([], [], A)$

(Empty stack, Empty buffer, Complete tree)

Success!
We have read all of the input words (empty buffer) and have attached all input words to some other word (empty stack)

Transition-based parsing

We process the sentence $S = w_0 w_1 ... w_n$ from left to right ("incremental parsing")

In the parser configuration $(\sigma|w_i, w_j|\beta, A)$:

- w_i is on top of the stack. w_i may have some children
- w_j is on top of the buffer. w_j may have some children
- w_i precedes w_j $(i < j)$

We have to either attach w_i to w_j, attach w_j to w_i, or decide that there is no dependency between w_i and w_j

If we reach $(\sigma|w_i, w_j|\beta, A)$, all words w_k with $i < k < j$ have already been attached to a parent w_m with $i \leq m \leq j$

Parser actions

(σ, β, A): Parser configuration with stack σ, buffer β, set of arcs A
(w, r, w'): Dependency with head w, relation r and dependent w'

SHIFT: Push the next input word w_i from the buffer β onto the stack σ

$$(\sigma, w_i|\beta, A) \Rightarrow (\sigma|w_i, \beta, A)$$

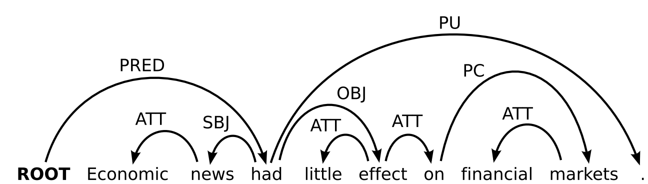
LEFT-ARC$_r$: … w_i…w_j… (dependent precedes the head)
Attach dependent w_i (top of stack σ) to head w_j (top of buffer β) with relation r from w_j to w_i. Pop w_i off the stack.

$$(\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_j|\beta, A \cup \{(w_j, r, w_i)\})$$

RIGHT-ARC$_r$: …w_i…w_j … (dependent follows the head)
Attach dependent w_j (top of buffer β) to head w_i (top of stack σ) with relation r from w_i to w_j. Move w_i back to the buffer

$$(\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_i|\beta, A \cup \{(w_i, r, w_j)\})$$

An example sentence & parse



Economic news had little effect on financial markets .

Economic news had little effect on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)

Economic news had little effect on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)

Economic **news** had little effect on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)

Economic news had little effect on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	$\emptyset$)
SH $\Rightarrow$	([ROOT, Economic],	[news, . . . , .],	$\emptyset$)
LA_{ATT} $\Rightarrow$	([ROOT],	[news, . . . , .],	$A_1 = \{(\text{news, ATT, Economic})\}$)

Economic **news had** little effect on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	$\emptyset$)
SH $\Rightarrow$	([ROOT, Economic],	[news, . . . , .],	$\emptyset$)
LA_{ATT} $\Rightarrow$	([ROOT],	[news, . . . , .],	$A_1 = \{(\text{news, ATT, Economic})\}$)
SH $\Rightarrow$	([ROOT, news],	[had, . . . , .],	A_1)

Economic **news had** little effect on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	$\emptyset$)
SH $\Rightarrow$	([ROOT, Economic],	[news, . . . , .],	$\emptyset$)
LA_{ATT} $\Rightarrow$	([ROOT],	[news, . . . , .],	$A_1 = \{(\text{news, ATT, Economic})\}$)
SH $\Rightarrow$	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} $\Rightarrow$	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup \{(\text{had, SBJ, news})\}$)

Economic news **had little** effect on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	$\emptyset$)
SH $\Rightarrow$	([ROOT, Economic],	[news, . . . , .],	$\emptyset$)
LA_{ATT} $\Rightarrow$	([ROOT],	[news, . . . , .],	$A_1 = \{(\text{news, ATT, Economic})\}$)
SH $\Rightarrow$	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} $\Rightarrow$	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup \{(\text{had, SBJ, news})\}$)
SH $\Rightarrow$	([ROOT, had],	[little, . . . , .],	A_2)

Economic news had **little** **effect** on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	$A_1 = \{(news, ATT, Economic)\}$)
SH ⇒	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup \{(had, SBJ, news)\}$)
SH ⇒	([ROOT, had],	[little, . . . , .],	A_2)
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	A_2)

Economic news **had** **little** **effect** on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	$A_1 = \{(news, ATT, Economic)\}$)
SH ⇒	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup \{(had, SBJ, news)\}$)
SH ⇒	([ROOT, had],	[little, . . . , .],	A_2)
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	A_2)
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup \{(effect, ATT, little)\}$)

Economic news had little **effect** **on** financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	$A_1 = \{(news, ATT, Economic)\}$)
SH ⇒	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup \{(had, SBJ, news)\}$)
SH ⇒	([ROOT, had],	[little, . . . , .],	A_2)
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	A_2)
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup \{(effect, ATT, little)\}$)
SH ⇒	([ROOT, had, effect],	[on, . . . , .],	A_3)

Economic news had little effect **on** **financial** markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	$A_1 = \{(news, ATT, Economic)\}$)
SH ⇒	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup \{(had, SBJ, news)\}$)
SH ⇒	([ROOT, had],	[little, . . . , .],	A_2)
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	A_2)
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup \{(effect, ATT, little)\}$)
SH ⇒	([ROOT, had, effect],	[on, . . . , .],	A_3)
SH ⇒	([ROOT, . . . on],	[financial, markets, .],	A_3)

Economic news had little effect on **financial** **markets** .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	$A_1 = \{(\text{news, ATT, Economic})\})$
SH ⇒	([ROOT, news],	[had, . . . , .],	$A_1)$
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup \{(\text{had, SBJ, news})\})$
SH ⇒	([ROOT, had],	[little, . . . , .],	$A_2)$
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	$A_2)$
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup \{(\text{effect, ATT, little})\})$
SH ⇒	([ROOT, had, effect],	[on, . . . , .],	$A_3)$
SH ⇒	([ROOT, . . . on],	[financial, markets, .],	$A_3)$
SH ⇒	([ROOT, . . . , financial],	[markets, .],	$A_3)$

Economic news had little effect **on** **financial** **markets** .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	$A_1 = \{(\text{news, ATT, Economic})\})$
SH ⇒	([ROOT, news],	[had, . . . , .],	$A_1)$
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup \{(\text{had, SBJ, news})\})$
SH ⇒	([ROOT, had],	[little, . . . , .],	$A_2)$
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	$A_2)$
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup \{(\text{effect, ATT, little})\})$
SH ⇒	([ROOT, had, effect],	[on, . . . , .],	$A_3)$
SH ⇒	([ROOT, . . . on],	[financial, markets, .],	$A_3)$
SH ⇒	([ROOT, . . . , financial],	[markets, .],	$A_3)$
LA_{ATT} ⇒	([ROOT, . . . on],	[markets, .],	$A_4 = A_3 \cup \{(\text{markets, ATT, financial})\})$

Economic news had little **effect** **on** financial **markets** .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	$A_1 = \{(\text{news, ATT, Economic})\})$
SH ⇒	([ROOT, news],	[had, . . . , .],	$A_1)$
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup \{(\text{had, SBJ, news})\})$
SH ⇒	([ROOT, had],	[little, . . . , .],	$A_2)$
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	$A_2)$
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup \{(\text{effect, ATT, little})\})$
SH ⇒	([ROOT, had, effect],	[on, . . . , .],	$A_3)$
SH ⇒	([ROOT, . . . on],	[financial, markets, .],	$A_3)$
SH ⇒	([ROOT, . . . , financial],	[markets, .],	$A_3)$
LA_{ATT} ⇒	([ROOT, . . . on],	[markets, .],	$A_4 = A_3 \cup \{(\text{markets, ATT, financial})\})$
RA_{PC} ⇒	([ROOT, had, effect],	[on, .],	$A_5 = A_4 \cup \{(\text{on, PC, markets})\})$

Economic news **had** little **effect** **on** financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	$A_1 = \{(\text{news, ATT, Economic})\})$
SH ⇒	([ROOT, news],	[had, . . . , .],	$A_1)$
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup \{(\text{had, SBJ, news})\})$
SH ⇒	([ROOT, had],	[little, . . . , .],	$A_2)$
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	$A_2)$
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup \{(\text{effect, ATT, little})\})$
SH ⇒	([ROOT, had, effect],	[on, . . . , .],	$A_3)$
SH ⇒	([ROOT, . . . on],	[financial, markets, .],	$A_3)$
SH ⇒	([ROOT, . . . , financial],	[markets, .],	$A_3)$
LA_{ATT} ⇒	([ROOT, . . . on],	[markets, .],	$A_4 = A_3 \cup \{(\text{markets, ATT, financial})\})$
RA_{PC} ⇒	([ROOT, had, effect],	[on, .],	$A_5 = A_4 \cup \{(\text{on, PC, markets})\})$
RA_{ATT} ⇒	([ROOT, had],	[effect, .],	$A_6 = A_5 \cup \{(\text{effect, ATT, on})\})$

Economic news **had** little **effect** on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	A_1 = {(news, ATT, Economic)})
SH ⇒	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup$ {(had, SBJ, news)})
SH ⇒	([ROOT, had],	[little, . . . , .],	A_2)
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	A_2)
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup$ {(effect, ATT, little)})
SH ⇒	([ROOT, had, effect],	[on, . . . , .],	A_3)
SH ⇒	([ROOT, . . . on],	[financial, markets, .],	A_3)
SH ⇒	([ROOT, . . . , financial],	[markets, .],	A_3)
LA_{ATT} ⇒	([ROOT, . . . on],	[markets, .],	$A_4 = A_3 \cup$ {(markets, ATT, financial)})
RA_{PC} ⇒	([ROOT, had, effect],	[on, .],	$A_5 = A_4 \cup$ {(on, PC, markets)})
RA_{ATT} ⇒	([ROOT, had],	[effect, .],	$A_6 = A_5 \cup$ {(effect, ATT, on)})
RA_{OBJ} ⇒	([ROOT],	[had, .],	$A_7 = A_6 \cup$ {(had, OBJ, effect)})

Economic news **had** little effect on financial markets **.**

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	A_1 = {(news, ATT, Economic)})
SH ⇒	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup$ {(had, SBJ, news)})
SH ⇒	([ROOT, had],	[little, . . . , .],	A_2)
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	A_2)
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup$ {(effect, ATT, little)})
SH ⇒	([ROOT, had, effect],	[on, . . . , .],	A_3)
SH ⇒	([ROOT, . . . on],	[financial, markets, .],	A_3)
SH ⇒	([ROOT, . . . , financial],	[markets, .],	A_3)
LA_{ATT} ⇒	([ROOT, . . . on],	[markets, .],	$A_4 = A_3 \cup$ {(markets, ATT, financial)})
RA_{PC} ⇒	([ROOT, had, effect],	[on, .],	$A_5 = A_4 \cup$ {(on, PC, markets)})
RA_{ATT} ⇒	([ROOT, had],	[effect, .],	$A_6 = A_5 \cup$ {(effect, ATT, on)})
RA_{OBJ} ⇒	([ROOT],	[had, .],	$A_7 = A_6 \cup$ {(had, OBJ, effect)})
SH ⇒	([ROOT, had],	[.],	A_7)

Economic news **had** little effect on financial markets **.**

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	A_1 = {(news, ATT, Economic)})
SH ⇒	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup$ {(had, SBJ, news)})
SH ⇒	([ROOT, had],	[little, . . . , .],	A_2)
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	A_2)
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup$ {(effect, ATT, little)})
SH ⇒	([ROOT, had, effect],	[on, . . . , .],	A_3)
SH ⇒	([ROOT, . . . on],	[financial, markets, .],	A_3)
SH ⇒	([ROOT, . . . , financial],	[markets, .],	A_3)
LA_{ATT} ⇒	([ROOT, . . . on],	[markets, .],	$A_4 = A_3 \cup$ {(markets, ATT, financial)})
RA_{PC} ⇒	([ROOT, had, effect],	[on, .],	$A_5 = A_4 \cup$ {(on, PC, markets)})
RA_{ATT} ⇒	([ROOT, had],	[effect, .],	$A_6 = A_5 \cup$ {(effect, ATT, on)})
RA_{OBJ} ⇒	([ROOT],	[had, .],	$A_7 = A_6 \cup$ {(had, OBJ, effect)})
SH ⇒	([ROOT, had],	[.],	A_7)
RA_{PU} ⇒	([ROOT],	[had],	$A_8 = A_7 \cup$ {(had, PU, .)})

Economic news **had** little effect on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	A_1 = {(news, ATT, Economic)})
SH ⇒	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1 \cup$ {(had, SBJ, news)})
SH ⇒	([ROOT, had],	[little, . . . , .],	A_2)
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	A_2)
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2 \cup$ {(effect, ATT, little)})
SH ⇒	([ROOT, had, effect],	[on, . . . , .],	A_3)
SH ⇒	([ROOT, . . . on],	[financial, markets, .],	A_3)
SH ⇒	([ROOT, . . . , financial],	[markets, .],	A_3)
LA_{ATT} ⇒	([ROOT, . . . on],	[markets, .],	$A_4 = A_3 \cup$ {(markets, ATT, financial)})
RA_{PC} ⇒	([ROOT, had, effect],	[on, .],	$A_5 = A_4 \cup$ {(on, PC, markets)})
RA_{ATT} ⇒	([ROOT, had],	[effect, .],	$A_6 = A_5 \cup$ {(effect, ATT, on)})
RA_{OBJ} ⇒	([ROOT],	[had, .],	$A_7 = A_6 \cup$ {(had, OBJ, effect)})
SH ⇒	([ROOT, had],	[.],	A_7)
RA_{PU} ⇒	([ROOT],	[had],	$A_8 = A_7 \cup$ {(had, PU, .)})
RA_{PRED} ⇒	([ ],	[ROOT],	$A_9 = A_8 \cup$ {(ROOT, PRED, had)})

Economic news had little effect on financial markets .

Transition	Configuration		
	([ROOT],	[Economic, . . . , .],	∅)
SH ⇒	([ROOT, Economic],	[news, . . . , .],	∅)
LA_{ATT} ⇒	([ROOT],	[news, . . . , .],	A_1 = {(news, ATT, Economic)})
SH ⇒	([ROOT, news],	[had, . . . , .],	A_1)
LA_{SBJ} ⇒	([ROOT],	[had, . . . , .],	$A_2 = A_1$∪{(had, SBJ, news)})
SH ⇒	([ROOT, had],	[little, . . . , .],	A_2)
SH ⇒	([ROOT, had, little],	[effect, . . . , .],	A_2)
LA_{ATT} ⇒	([ROOT, had],	[effect, . . . , .],	$A_3 = A_2$∪{(effect, ATT, little)})
SH ⇒	([ROOT, had, effect],	[on, . . . , .],	A_3)
SH ⇒	([ROOT, . . . on],	[financial, markets, .],	A_3)
SH ⇒	([ROOT, . . . , financial],	[markets, .],	A_3)
LA_{ATT} ⇒	([ROOT, . . . on],	[markets, .],	$A_4 = A_3$∪{(markets, ATT, financial)})
RA_{PC} ⇒	([ROOT, had, effect],	[on, .],	$A_5 = A_4$∪{(on, PC, markets)})
RA_{ATT} ⇒	([ROOT, had],	[effect, .],	$A_6 = A_5$∪{(effect, ATT, on)})
RA_{OBJ} ⇒	([ROOT],	[had, .],	$A_7 = A_6$∪{(had, OBJ, effect)})
SH ⇒	([ROOT, had],	[.],	A_7)
RA_{PU} ⇒	([ROOT],	[had],	$A_8 = A_7$∪{(had, PU, .)})
RA_{PRED} ⇒	([ ],	[ROOT],	$A_9 = A_8$∪{(ROOT, PRED, had)})
SH ⇒	([ROOT],	[ ],	A_9)

Transition-based parsing in practice

Which action should the parser take under the current configuration?

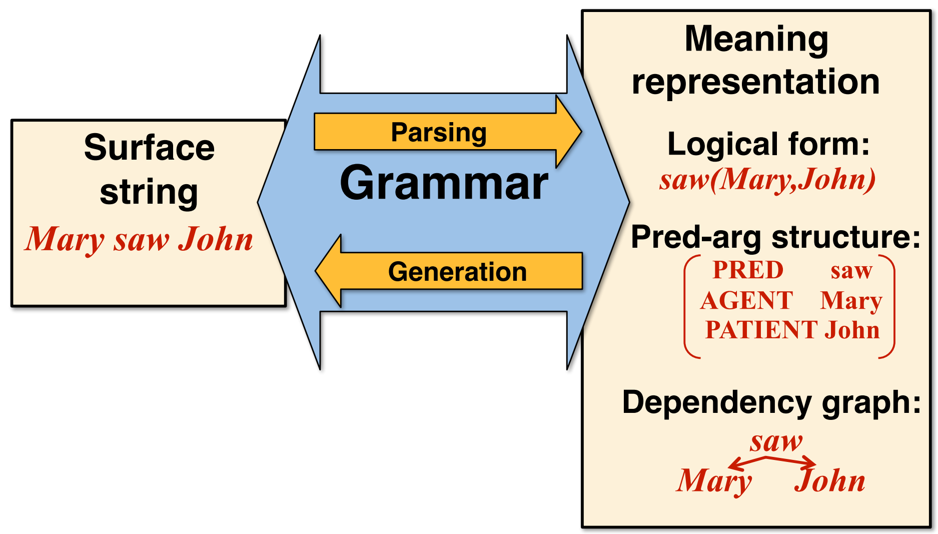
We also need a parsing model that assigns a score to each possible action given a current configuration.

- Possible actions:
  SHIFT, and for any relation r: LEFT-ARC$_r$, or RIGHT-ARC$_r$
- Possible features of the current configuration:
  The top {1,2,3} words on the buffer and on the stack, their POS tags, distances between the words, etc.

We can learn this model from a dependency treebank.

Expressive Grammars

Why grammar?



Grammar formalisms

Formalisms provide a **language** in which linguistic theories can be expressed and implemented

Formalisms define **elementary objects** (trees, strings, feature structures) and **recursive operations** which generate complex objects from simple objects.

Formalisms may impose **constraints** (e.g. on the kinds of dependencies they can capture)

How do grammar formalisms differ?

Formalisms define different **representations**

Tree-adjoining Grammar (TAG):
- Fragments of phrase-structure trees

Lexical-functional Grammar (LFG):
- Annotated phrase-structure trees (c-structure) linked to feature structures (f-structure)

Combinatory Categorial Grammar (CCG):
- Syntactic categories paired with meaning representations

Head-Driven Phrase Structure Grammar(HPSG):
- Complex feature structures (Attribute-value matrices)

The dependencies so far:

Arguments:
- Verbs take arguments: subject, object, complements, ...
- **Heads subcategorize for their arguments**

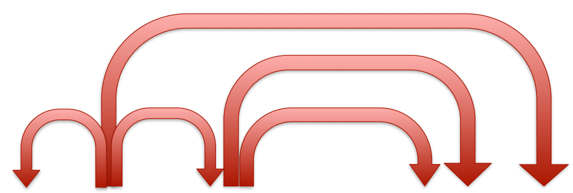
Adjuncts/Modifiers:
- Adjectives modify nouns, adverbs modify VPs or adjectives, PPs modify NPs or VPs
- **Modifiers subcategorize for the head**

Typically, these are *local* dependencies: they can be expressed *within individual CFG rules*

VP → Adv Verb NP

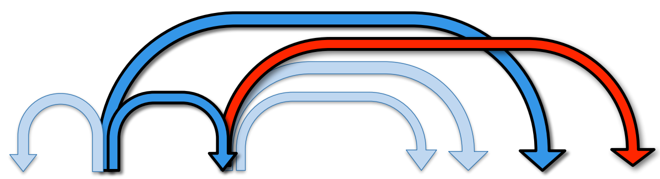
Context-free grammars

CFGs capture only **nested** dependencies
- The dependency graph is a **tree**
- The dependencies **do not cross**



Beyond CFGs: Nonprojective dependencies

Dependencies form a **tree with crossing branches**



Non-projective dependencies

(Non-local) scrambling: In a sentence with multiple verbs, the argument of a verb appears in a different clause from that which contains the verb (arises in languages with freer word order than English)

Die Pizza *hat Klaus versprochen zu* ***bringen***
The pizza has Klaus promised to bring
Klaus has promised to bring the pizza

Extraposition: Here, a modifier of the subject NP is moved to the end of the sentence

*The **guy** is coming **who is wearing a hat***
Compare with the non-extraposed variant
*The **[guy [who is wearing a hat]]** is coming*

Topicalization: Here, the argument of the embedded verb is moved to the front of the sentence.

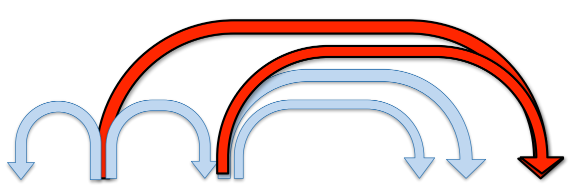
Cheeseburgers*, I [thought [he* ***likes****]]*

Beyond CFGs: Nonlocal dependencies

Dependencies form a **DAG**
(a node may have **multiple incoming edges**)

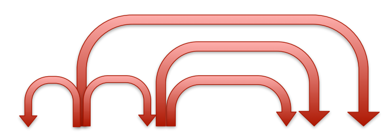
Arise in the following constructions:

- **Control** *(**He** has **promised** me to **go**)*, **raising** (**He seems** to **go**)
- **Wh-movement** *(the **man** who you **saw** yesterday **is** here again)*,
- **Non-constituent** coordination
  (right-node raising, gapping, argument-cluster coordination)

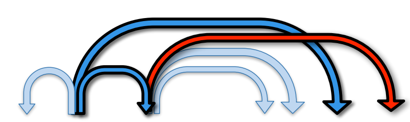


Dependency structures

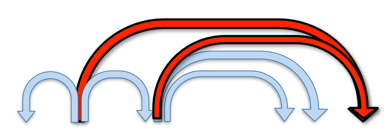
Nested (projective) dependency trees (CFGs)



Non-projective dependency trees



Non-local dependency graphs



Non-local dependencies

Long-range dependencies

Bounded long-range dependencies:
Limited distance between the head and argument

Unbounded long-range dependencies:
Arbitrary distance (within the same sentence) between the head and argument

Unbounded long-range dependencies cannot (in general) be represented with CFGs.

Chomsky's solution:
Add null elements (and co-indexation)

Unbounded nonlocal dependencies

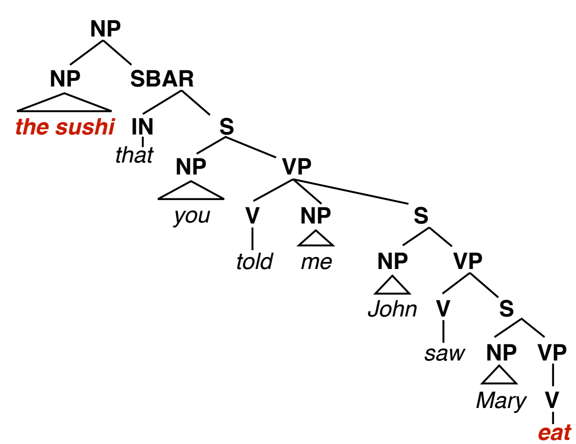
Wh-questions and relative clauses contain ***unbounded* nonlocal dependencies**, where the missing NP may be arbitrarily deeply embedded:

'the sushi that [you told me [John saw [Mary eat]]]'

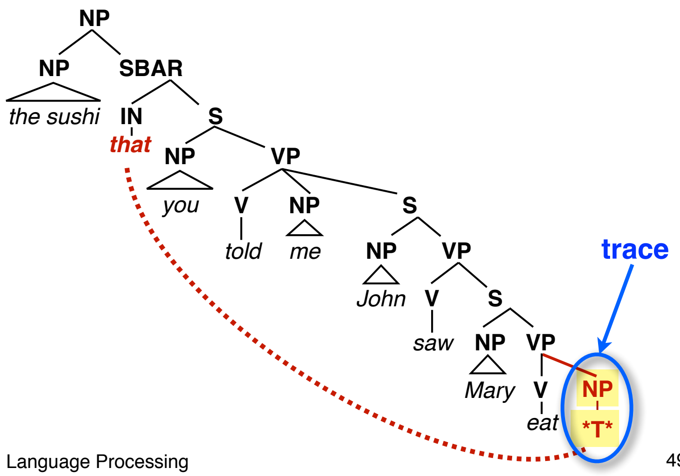
'what [did you tell me [John saw [Mary eat]]]?'

Linguists call this phenomenon **wh-extraction** (wh-movement).

Non-local dependencies in *wh*-extraction

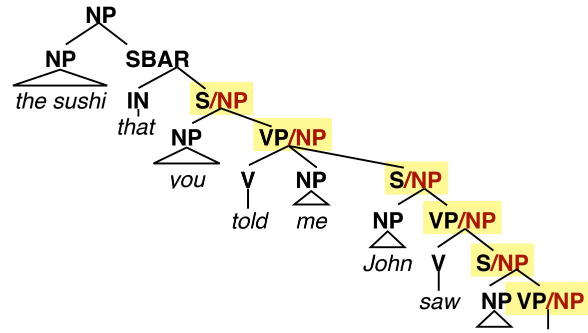


The trace analysis of *wh*-extraction



Slash categories for wh-extraction

Because only one element can be extracted, we can use **slash categories.**
This is still a CFG: the set of nonterminals is finite.



Generalized Phrase Structure Grammar (GPSG), Gazdar et al. (1985)

German: center embedding

...daß ich [Hans schwimmen] sah
...that I Hans swim saw
...that I saw [Hans swim]

...daß ich [Maria [Hans schwimmen] helfen] sah
...that I Maria Hans swim help saw
...that I saw [Mary help [Hans swim]]

...daß ich [Anna [Maria [Hans schwimmen] helfen] lassen] sah
...that I Anna Maria Hans swim help let saw
...that I saw [Anna let [Mary help [Hans swim]]]

Dutch: cross-serial dependencies

...dat ik Hans zag zwemmen
...that I Hans saw swim
...that I saw [Hans swim]

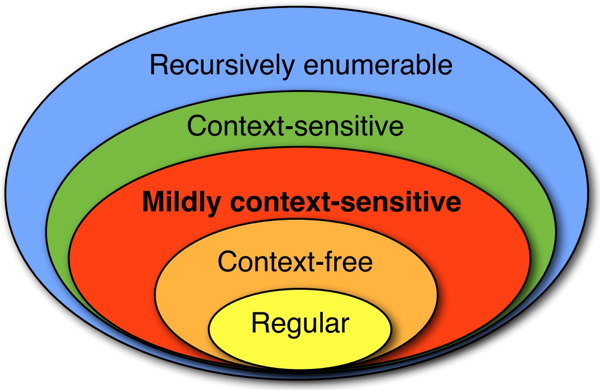
...dat ik Maria Hans zag helpen zwemmen
...that I Maria Hans saw help swim
...that I saw [Mary help [Hans swim]]

...dat ik Anna Maria Hans zag laten helpen zwemmen
...that I Anna Maria Hans saw let help swim
...that I saw [Anna let [Mary help [Hans swim]]]

Such cross-serial dependencies require *mildly context-sensitive grammars*

Two mildly context-sensitive formalisms: TAG and CCG

The Chomsky Hierarchy



Mildly context-sensitive grammars

Contain all context-free grammars/languages

Can be **parsed in polynomial time** (TAG/CCG: $O(n^6)$)

(*Strong* generative capacity) capture certain kinds of dependencies: **nested** (like CFGs) and **cross-serial** (like the Dutch example), but not the MIX language:
MIX: the set of strings $w \in \{a, b, c\}^*$ that contain equal numbers of as, bs and cs

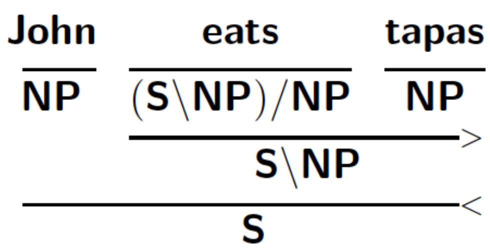
Have the **constant growth** property:
the length of strings grows in a linear way
The power-of-2 language $\{a^{2n}\}$ does not have the constant growth propery.

TAG and CCG are lexicalized formalisms

The lexicon:
- pairs words with elementary objects
- specifies all language-specific information (e.g. subcategorization information)

The grammatical operations:
- are universal
- define (and impose constraints on) recursion.

A (C)CG derivation



CCG categories are defined recursively:

- Categories are atomic (S, NP) or complex (S\NP, (S\NP)/NP)
- Complex categories (X/Y or X\Y) are functions:

X/Y combines with an adjacent argument to its right of category Y to return a result of category X.

Function categories can be composed, giving more expressive power than CFGs

More on CCG in one of our Semantics lectures!

Tree-Adjoining Grammar

(Lexicalized) Tree-Adjoining Grammar

TAG is a tree-rewriting formalism:

TAG defines operations (**substitution**, **adjunction**) on trees.
The **elementary objects** in TAG are trees (not strings)

TAG is lexicalized:

Each elementary tree is **anchored** to a lexical item (word)
"Extended domain of locality":
The elementary tree contains all arguments of the anchor.
TAG requires a linguistic theory which specifies the shape of these elementary trees.

TAG is mildly context-sensitive:

can capture Dutch cross-serial dependencies
but is still efficiently parseable

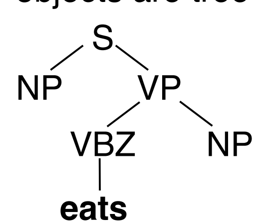
AK Joshi and Y Schabes (1996) Tree Adjoining Grammars. In G. Rosenberg and A. Salomaa, Eds., Handbook of Formal Languages

Extended domain of locality

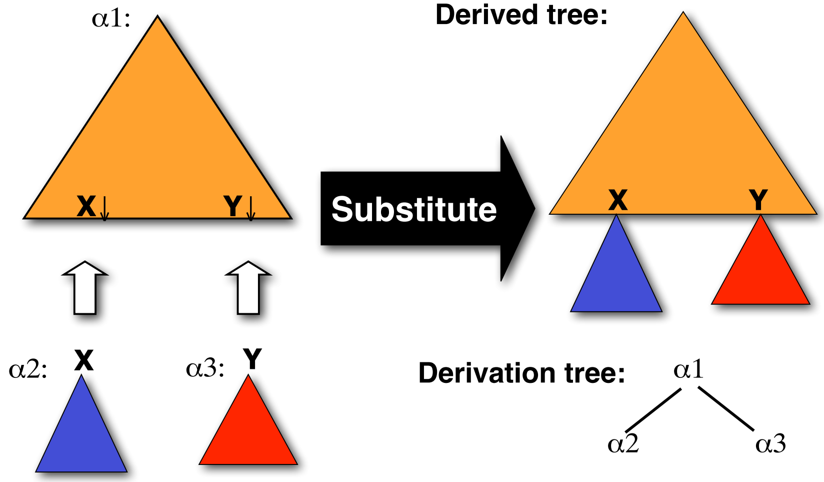
We want to capture **all arguments of a word** in a **single elementary object**.

We also want to retain certain syntactic structures (e.g. VPs).

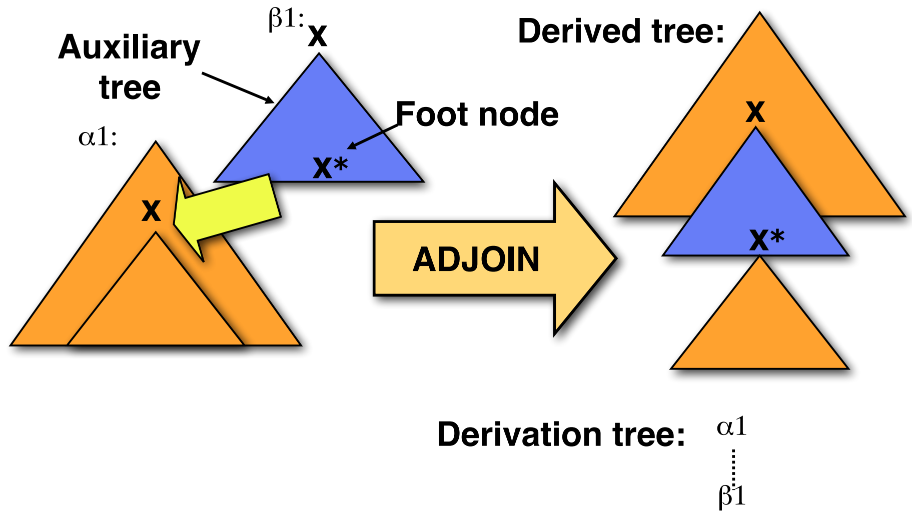
Our elementary objects are tree fragments:



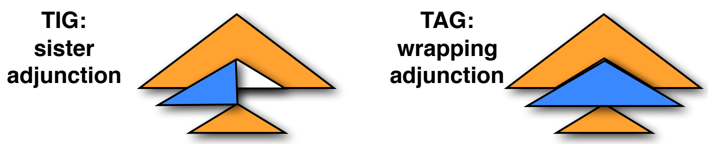
TAG substitution (arguments)



TAG adjunction



The effect of adjunction

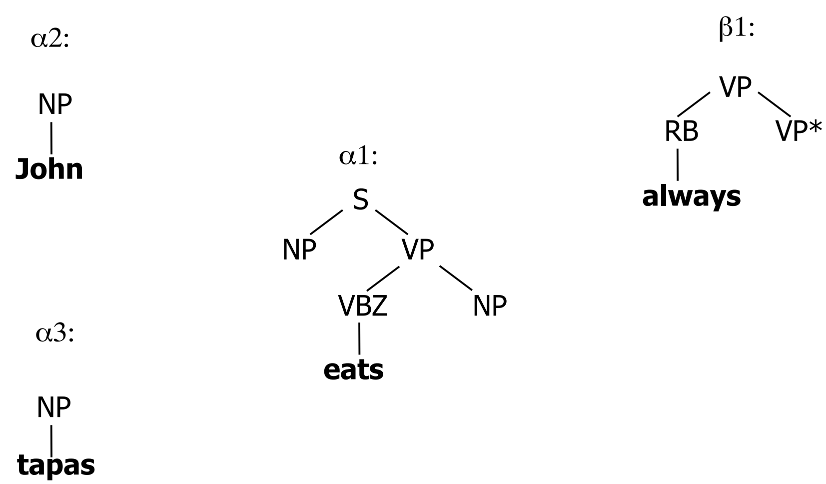


No adjunction: TSG (Tree substitution grammar)
- TSG is context-free

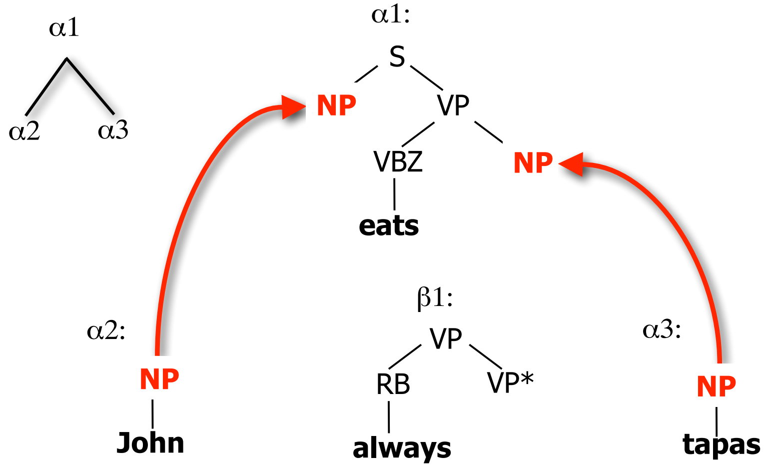
Sister adjunction: TIG (Tree insertion grammar)
- TIG is also context-free, but has a linguistically more adequate treatment of modifiers

Wrapping adjunction: TAG (Tree-adjoining grammar)
- TAG is mildy context-sensitive

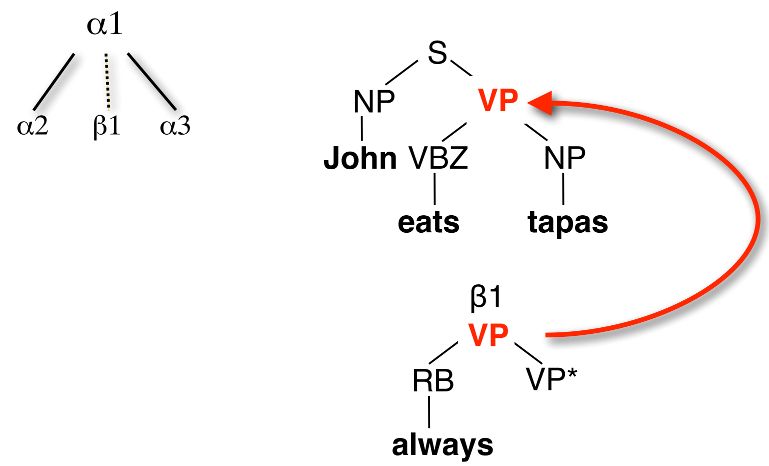
A small TAG lexicon



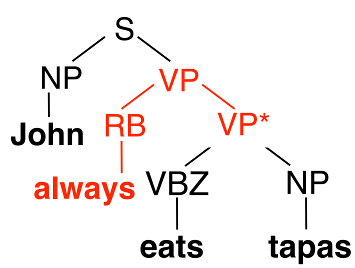
A TAG derivation



A TAG derivation



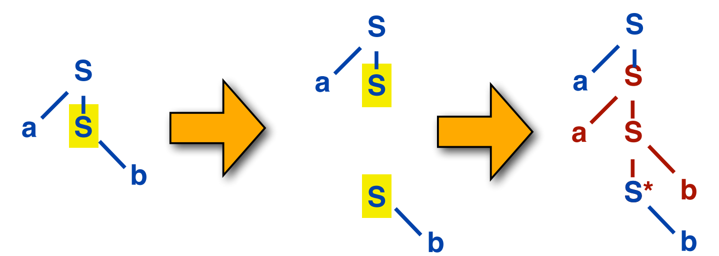
A TAG derivation



$a^n b^n$: Cross-serial dependencies

Elementary trees:

Deriving aabb



Feature Structure Grammars

Simple grammars overgenerate

$$
\begin{array}{rcl}
S & \rightarrow & NP \;\; VP \\
VP & \rightarrow & Verb \;\; NP \\
NP & \rightarrow & Det \;\; Noun \\
Det & \rightarrow & the \mid a \mid these \\
Verb & \rightarrow & eat \mid eats \\
Noun & \rightarrow & cake \mid cakes \mid student \mid students
\end{array}
$$

This generates ungrammatical sentences like
“these student eats a cakes”

We need to capture (number/person) agreement

Refining the nonterminals

$$
\begin{array}{rcl}
S & \rightarrow & NPsg \;\; VPsg \\
S & \rightarrow & NPpl \;\; VPpl \\
VPsg & \rightarrow & VerbSg \;\; NP \\
VPpl & \rightarrow & VerbPl \;\; NP \\
NPsg & \rightarrow & DetSg \;\; NounSg \\
DetSg & \rightarrow & the \mid a \\
\ldots & \ldots & \ldots
\end{array}
$$

This yields very large grammars.
- What about person, case, ...?

Difficult to capture generalizations.
- Subject and verb have to have number agreement
- *NPsg*, *NPpl* and *NP* are three distinct nonterminals

Feature structures

Replace atomic categories with feature structures:

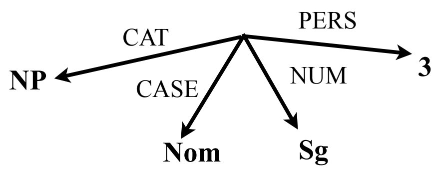
$$
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUM} & \text{SG} \\
\text{PERS} & 3 \\
\text{CASE} & \text{NOM}
\end{bmatrix}
\qquad
\begin{bmatrix}
\text{CAT} & \text{VP} \\
\text{NUM} & \text{SG} \\
\text{PERS} & 3 \\
\text{VFORM} & \text{FINITE}
\end{bmatrix}
$$

A feature structure is a list of features (= attributes), e.g. CASE, and values (eg NOM).

We often represent feature structures as attribute value matrices (AVM)
Usually, values are typed (to avoid CASE:SG)

Feature structures as directed graphs

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{CASE} & \text{NOM} \end{bmatrix} =$$



Complex feature structures

We distinguish between atomic and complex feature values.
A complex value is a feature structure itself.

This allows us to capture better generalizations.

Only atomic values:

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{CASE} & \text{NOM} \end{bmatrix}$$

Complex values:

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{CASE} & \text{NOM} \end{bmatrix} \end{bmatrix}$$

Feature paths

$$\textbf{NP:} \begin{bmatrix} \text{CAT} & \text{NP} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{CASE} & \text{NOM} \end{bmatrix} \end{bmatrix}$$

A feature path allows us to identify particular values in a feature structure:

$\langle \textbf{NP}\ \text{CAT} \rangle = \text{NP}$

$\langle \textbf{NP}\ \text{AGR CASE} \rangle = \text{NOM}$

Unification

Two feature structures A and B **unify** (A ⊔ B) if they can be merged into one consistent feature structure C:

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{SG} \\ \text{CASE} & \text{NOM} \end{bmatrix} \sqcup \begin{bmatrix} \text{CAT} & \text{NP} \\ \text{PERS} & 3 \end{bmatrix} = \begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{CASE} & \text{NOM} \end{bmatrix}$$

Otherwise. unification fails:

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{SG} \\ \text{CASE} & \text{NOM} \end{bmatrix} \sqcup \begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{PL} \end{bmatrix} = \emptyset$$

PATR-II style feature structures

CFG rules are augmented with constraints:

$A_0 \rightarrow A_1 \ldots A_n$
{set of constraints}

There are two kinds of constraints:

Unification constraints:
$\langle A_i$ feature-path$\rangle = \langle A_j$ feature-path$\rangle$

Value constraints:
$\langle A_i$ feature-path$\rangle =$ atomic value

A grammar with feature structures

S → **NP VP**			**Grammar rule**
⟨**NP** *NUM*⟩	=	⟨**VP** *NUM*⟩	**Constraints**
⟨**NP** *CASE*⟩	=	*nom*	

NP → **DT NOUN**			**Grammar rule**
⟨**NP** *NUM*⟩	=	⟨**NOUN** *NUM*⟩	**Constraints**
⟨**NP** *CASE*⟩	=	⟨**NOUN** *CASE*⟩	

NOUN → *cake*			**Lexical entry**
⟨**NOUN** *NUM*⟩	=	*sg*	**Constraints**

With complex feature structures

S → **NP VP**			**Grammar rule**
⟨**NP** *AGR*⟩	=	⟨**VP** *AGR*⟩	**Constraints**
⟨**NP** *CASE*⟩	=	*nom*	

NP → **DT NOUN**			**Grammar rule**
⟨**NP** *AGR*⟩	=	⟨**NOUN** *AGR*⟩	**Constraints**

NOUN → *cake*			**Lexical entry**
⟨**NOUN** AGR NUM⟩	=	*sg*	**Constraints**

Complex feature structures capture better generalizations (and hence require fewer constraints) — cf. the previous slide

Attribute-Value Grammars and CFGs

If every feature can only have a finite set of values, any attribute-value grammar can be compiled out into a (possibly huge) context-free grammar

Going beyond CFGs

The power-of-2 language: $L_2 = \{w^i \mid i \text{ is a power of } 2\}$

L_2 is a (fully) context-sensitive language.

(*Mildly* context-sensitive languages have the **constant growth property** (the length of words always increases by a constant factor c))

Here is a feature grammar which generates L_2:

$$\begin{aligned} A \rightarrow\ & a \\ & \langle A\ F \rangle = 1 \\ A \rightarrow\ & A_1 \quad A_2 \\ & \langle A\ F \rangle = \langle A_1 \rangle \\ & \langle A\ F \rangle = \langle A_2 \rangle \end{aligned}$$

Today's key concepts

Transition-based dependency parsing
- for projective dependency trees

Going beyond projective dependencies:
- non-projective dependencies
- non-local dependencies

Expressive Grammars
- TAG
- CCG
- Feature-Structure Grammars