CS498JH: Introduction to NLP (Fall 2012)
http://cs.illinois.edu/class/cs498jh

Lecture 15: Dependency Grammars

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center
Office Hours: Wednesday, 12:15-1:15pm

Dependency grammar

Word-word dependencies are a component of many (most/all?) grammar formalims.

Dependency grammar assumes that syntactic structure consists *only* of dependencies.
Many variants. Modern DG began with Tesniere (1959).

DG is often used for **free word order languages**.

DG is **purely descriptive** (not a generative system like CFGs etc.), but certain formal equivalences are known.

What is a dependency?

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

There is a syntactic relation between a head H and a dependent D in a construction C if:
- the head H **determines the syntactic category** of the construction C.
- the head H **determines the semantic category** of the construction C; D gives semantic specification.
- the head H is **obligatory.** D may be optional.
- the head **selects** D and determines whether D is obligatory or not.
- The **form of D** depends on the head H (agreement)
- The linear position of D depends on the head H.

Different kinds of dependencies

Head-argument ('exocentric'): *eat sushi*
Arguments may be obligatory, but can only occur once.
The head alone cannot necessarily replace the construction.

Head-modifier ('endocentric'): *fresh sushi*
Modifiers are optional, and can occur more than once.
The head alone can replace the entire construction.

Head-specifier ('exocentric'; Tesniere's transfer): *the sushi*
Between function words (e.g. prepositions, determiners) and their arguments. Syntactic head ≠ semantic head

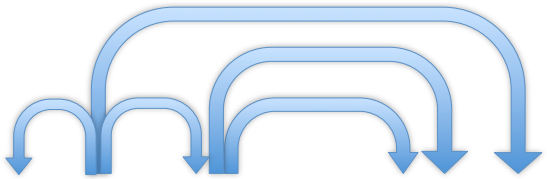
Coordination: (Tesniere's junction): *sushi and sashimi*
Unclear where the head is.

Context-free grammars

CFGs capture only **nested** dependencies

The dependency graph is a **tree**

The dependencies **do not cross**

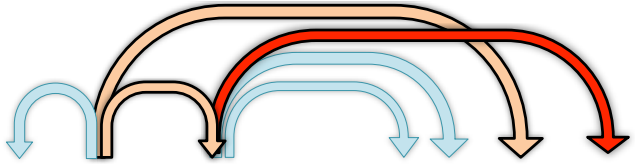


Beyond CFGs: Nonprojective dependencies

Dependencies: tree with crossing branches

Arise in the following constructions

- (Non-local) **scrambling** (free word order languages)
  Die Pizza hat Klaus versprochen zu bringen
- **Extraposition** (*The guy is coming who is wearing a hat*)
- **Topicalization** (*Cheeseburgers, I thought he likes*)



Dependency structures

Dependencies form a graph over the words in a sentence.

This graph is connected (every word is a node) and (typically) acyclic (no loops).

Single-head constraint:
Every node has at most one incoming edge.
This implies that the graph is a **rooted tree**.

Dependency trees and the linear order of words

Dependency trees do not specify the order of words in a sentence.
(Sometimes additional linear precedence constraints are introduced).

A dependency tree is projective if there are no crossing links.

Projective DG is weakly equivalent to CFG.

Parsing is more difficult for non-projective DGs

Dependency Treebanks

Dependency treebanks exist for many languages:

- Czech
- Arabic
- Turkish
- Danish,
- Portuguese
- Estonian,
- ....

Phrase-structure treebanks (e.g. the Penn Treebank) can also be translated into dependency trees
(although there might be noise in the translation)

The Prague Dependency Treebank

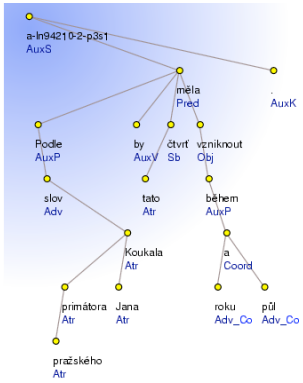
Three levels of annotation:

morphological: [<2M tokens]
Lemma (dictionary form) + detailed analysis
(15 categories with many possible values = 4,257 tags)

surface-syntactic ("analytical"): [1.5M tokens]
Labeled dependency tree encoding grammatical functions
(subject, object, conjunct, etc.)

semantic ("tectogrammatical"): [0.8M tokens]
Labeled dependency tree for predicate-argument structure,
information structure, coreference (not all words included)
(39 labels: agent, patient, origin, effect, manner, etc....)

Examples: analytical level

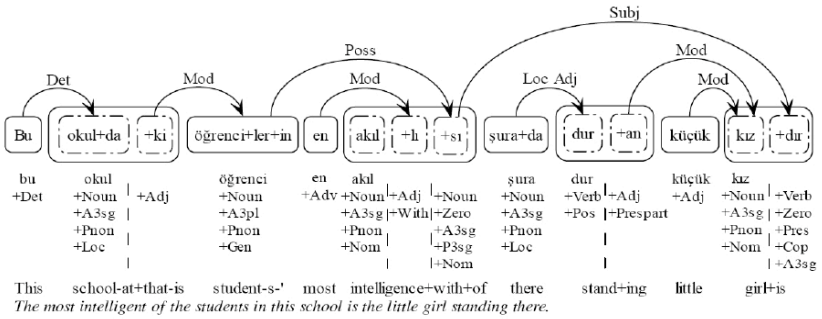


METU-Sabanci Turkish Treebank

Very small -- about 5000 sentences
Turkish is an agglutinative language with free word order:

- iyileştiriliyorken
  - (literally) while it is being caused to become good
  - while it is being improved
- iyi+Adj ^DB+Verb+Become^DB+Verb+Caus
  ^DB+Verb+Pass+Pos+Pres^DB+Adverb+While

Dependencies are at the morpheme level



[this and prev. example from Kemal Oflazer's talk at Rochester, April 2007]

Dependency or phrase structure annotation?

No clear consensus which is better.
May depend on the language.

It may also depend on the annotation guidelines:

- Early phrase-structure treebanks (Penn Treebank) are not explicit enough (not all nodes have function tags).
- Dependency treebanks (e.g. Sabanci) often omit long-range dependencies.
- They also can't express scope relations.

Parsing algorithms for DG

'Transition-based' parsers:

learn a sequence of actions to parse sentences

Models:

State = stack of partially processed items + queue of remaining tokens;
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

Transition-based parsing (Nivre et al.)

Transition-based parsing

Transition-based shift-reduce parsing processes the sentence $S = w_0 w_1 ... w_n$ from left to right.
Unlike CKY, it constructs a single tree.

N.B: this only works for projective dependency trees

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 words $w_i \in V_S$

β: a buffer of 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)

Stack σ: a list of words that are partially processed

We push and pop words onto σ

σ|w : w is on top of the stack

The buffer β is the remaining input words

We read words from β and push them onto σ

w|β : w is on top of the buffer

The set of arcs A defines the current tree

Initial configuration: $([w_0], [w_1, ..., w_n], \{\})$

Terminal configuration: $(\sigma, [], A)$

Parser actions

words that are attached to other words are fully processed

SHIFT: push the next input word onto the stack

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

LEFT-ARC$_r$: attach w_i (top of stack) to w_j (top of buffer)

If stack and buffer not empty, w_i not Root:

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

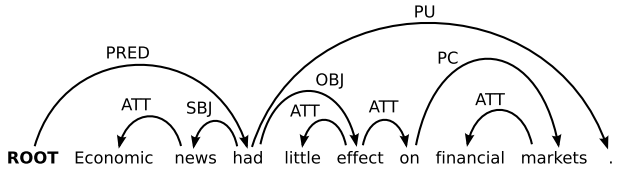
RIGHT-ARC$_r$: attach w_j (top of buffer) to w_j (top of stack)

Move w_j back to the buffer.

If stack and buffer not empty:

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

An example sentence & parse



Economic news had little effect on financial markets .

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