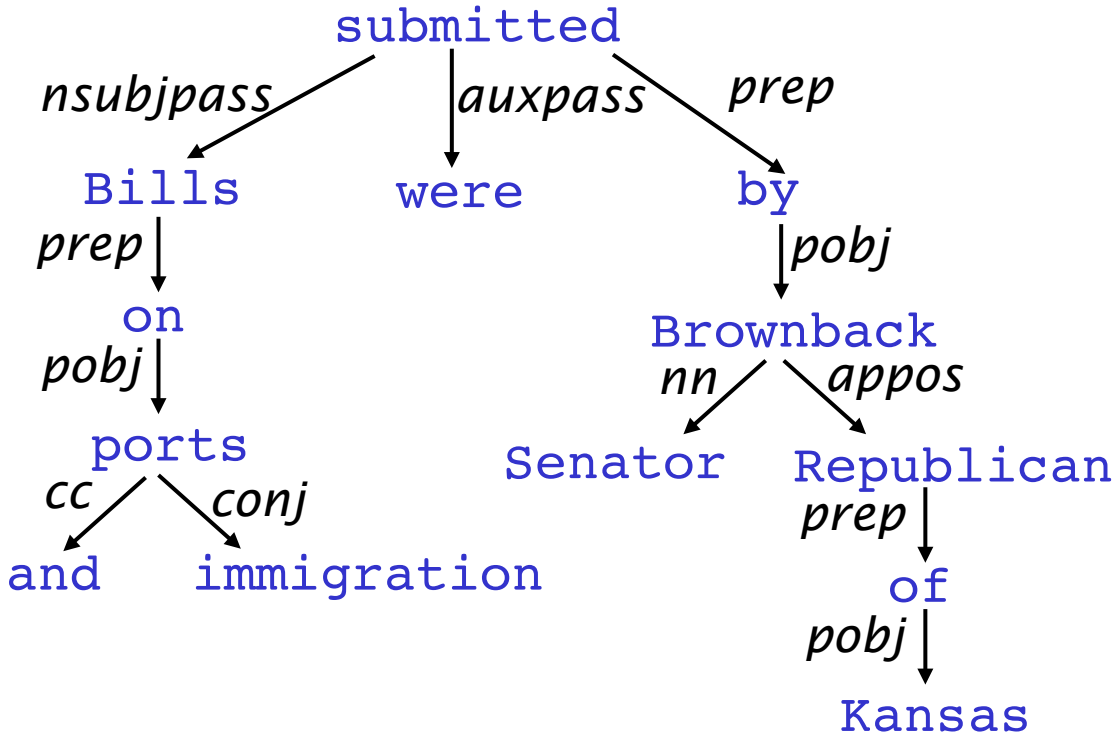
Dependency Parsing

Introduction

Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc.)

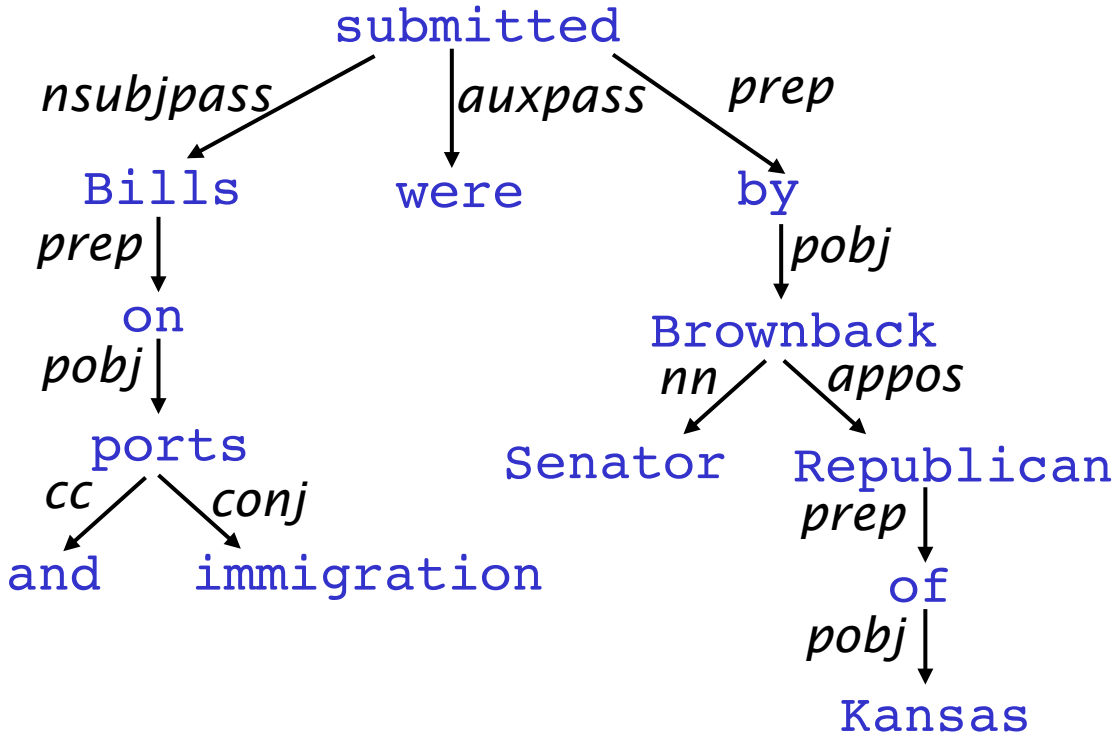


Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrow connects a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate)

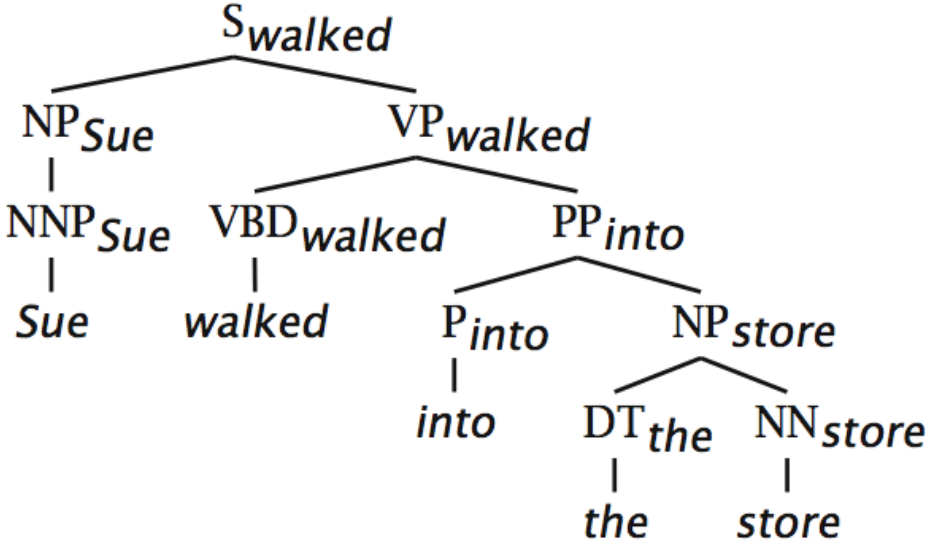
Usually, dependencies form a tree (connected, acyclic, single-head)



Relation between phrase structure and dependency structure

- A dependency grammar has a notion of a head. Officially, CFGs don't.
- But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, …) do, via hand-written phrasal "head rules":
  - The head of a Noun Phrase is a noun/number/adj/…
  - The head of a Verb Phrase is a verb/modal/….
- The head rules can be used to extract a dependency parse from a CFG parse

- The closure of dependencies give constituency from a dependency tree
- But the dependents of a word must be at the same level (i.e., "flat") – there can be no VP!



Methods of Dependency Parsing

1. Dynamic programming (like in the CKY algorithm)

   You can do it similarly to lexicalized PCFG parsing: an $O(n^5)$ algorithm

   Eisner (1996) gives a clever algorithm that reduces the complexity to $O(n^3)$, by producing parse items with heads at the ends rather than in the middle

2. Graph algorithms

   You create a Maximum Spanning Tree for a sentence

   McDonald et al.'s (2005) MSTParser scores dependencies independently using a ML classifier (he uses MIRA, for online learning, but it could be MaxEnt)

3. Constraint Satisfaction

   Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.

4. "Deterministic parsing"

   Greedy choice of attachments guided by machine learning classifiers

   MaltParser (Nivre et al. 2008) – discussed in the next segment

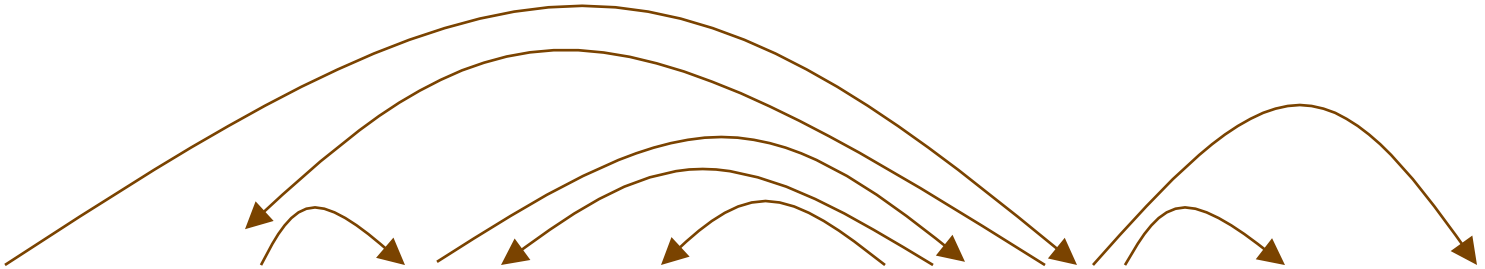
Dependency Conditioning Preferences

What are the sources of information for dependency parsing?

1. Bilexical affinities [issues → the] is plausible
2. Dependency distance mostly with nearby words
3. Intervening material

   Dependencies rarely span intervening verbs or punctuation
4. Valency of heads

   How many dependents on which side are usual for a head?



Quiz question!

- Consider this sentence:

Retail sales drop in April cools afternoon market trading.

- Which word are these words a dependent of?
  1. sales
  2. April
  3. afternoon
  4. trading

Greedy Transition-Based Parsing

MaltParser

MaltParser

[Nivre et al. 2008]

- A simple form of greedy discriminative dependency parser
- The parser does a sequence of bottom up actions
  - Roughly like "shift" or "reduce" in a shift-reduce parser, but the "reduce" actions are specialized to create dependencies with head on left or right
- The parser has:
  - a stack σ, written with top to the right
    - which starts with the ROOT symbol
  - a buffer β, written with top to the left
    - which starts with the input sentence
  - a set of dependency arcs A
    - which starts off empty
  - a set of actions

Basic transition-based dependency parser

Start: σ = [ROOT], β = w_1, …, w_n , A = ∅

1. Shift σ, w_i|β, A ➔ σ|w_i, β, A
2. Left-Arc$_r$ σ|w_i, w_j|β, A ➔ σ, w_j|β, A ∪ {$r(w_j,w_i)$}
3. Right-Arc$_r$ σ|w_i, w_j|β, A ➔ σ, w_i|β, A ∪ {$r(w_i,w_j)$}

Finish: β = ∅

Notes:

- Unlike the regular presentation of the CFG reduce step, dependencies combine one thing from each of stack and buffer

Actions ("arc-eager" dependency parser)

Start: σ = [ROOT], β = w_1, …, w_n , A = ∅

1. Left-Arc$_r$ σ|w_i, w_j|β, A ➔ σ, w_j|β, A ∪ {$r(w_j,w_i)$}

   Precondition: $r'(w_k, w_i) \notin$ A, $w_i \neq$ ROOT

2. Right-Arc$_r$ σ|w_i, w_j|β, A ➔ σ|w_i|w_j, β, A ∪ {$r(w_i,w_j)$}
3. Reduce σ|w_i, β, A ➔ σ, β, A

   Precondition: $r'(w_k, w_i) \in$ A

4. Shift σ, w_i|β, A ➔ σ|w_i, β, A

Finish: β = ∅

This is the common "arc-eager" variant: a head can immediately take a right dependent, before *its* dependents are found

Example

1. Left-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j,w_i)\}$
   Precondition: $(w_k, r', w_i) \notin A$, $w_i \neq$ ROOT
2. Right-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma|w_i|w_j, \beta, A \cup \{r(w_i,w_j)\}$
3. Reduce $\sigma|w_i, \beta, A \rightarrow \sigma, \beta, A$
   Precondition: $(w_k, r', w_i) \in A$
4. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$

Happy children like to play with their friends .

	[ROOT]	[Happy, children, …]	∅
Shift	[ROOT, Happy]	[children, like, …]	∅
LA$_{amod}$	[ROOT]	[children, like, …]	{amod(children, happy)} = A_1
Shift	[ROOT, children]	[like, to, …]	A_1
LA$_{nsubj}$	[ROOT]	[like, to, …]	$A_1 \cup$ {nsubj(like, children)} = A_2
RA$_{root}$	[ROOT, like]	[to, play, …]	$A_2 \cup$ {root(ROOT, like) = A_3
Shift	[ROOT, like, to]	[play, with, …]	A_3
LA$_{aux}$	[ROOT, like]	[play, with, …]	$A_3 \cup$ {aux(play, to) = A_4
RA$_{xcomp}$	[ROOT, like, play]	[with their, …]	$A_4 \cup$ {xcomp(like, play) = A_5

Example

1. Left-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j,w_i)\}$
   Precondition: $(w_k, r', w_i) \notin A$, $w_i \neq$ ROOT
2. Right-Arc$_r$ $\sigma|w_i, w_j|\beta, A \rightarrow \sigma|w_i|w_j, \beta, A \cup \{r(w_i,w_j)\}$
3. Reduce $\sigma|w_i, \beta, A \rightarrow \sigma, \beta, A$
   Precondition: $(w_k, r', w_i) \in A$
4. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$

Happy children like to play with their friends .

RA$_{xcomp}$	[ROOT, like, play]	[with their, …]	$A_4 \cup \{$xcomp(like, play) = A_5
RA$_{prep}$	[ROOT, like, play, with]	[their, friends, …]	$A_5 \cup \{$prep(play, with) = A_6
Shift	[ROOT, like, play, with, their]	[friends, .]	A_6
LA$_{poss}$	[ROOT, like, play, with]	[friends, .]	$A_6 \cup \{$poss(friends, their) = A_7
RA$_{pobj}$	[ROOT, like, play, with, friends]	[.]	$A_7 \cup \{$pobj(with, friends) = A_8
Reduce	[ROOT, like, play, with]	[.]	A_8
Reduce	[ROOT, like, play]	[.]	A_8
Reduce	[ROOT, like]	[.]	A_8
RA$_{punc}$	[ROOT, like, .]	[]	$A_8 \cup \{$punc(like, .) = A_9

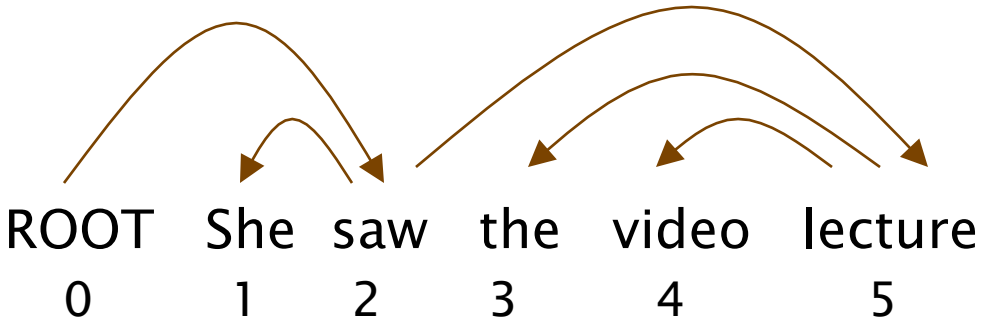
You terminate as soon as the buffer is empty. Dependencies = A_9

MaltParser

[Nivre et al. 2008]

- We have left to explain how we choose the next action
- Each action is predicted by a discriminative classifier (often SVM, could be maxent classifier) over each legal move
  - Max of 4 untyped choices, max of |R| × 2 + 2 when typed
  - Features: top of stack word, POS; first in buffer word, POS; etc.
- There is NO search (in the simplest and usual form)
  - But you could do some kind of beam search if you wish
- The model's accuracy is *slightly* below the best LPCFGs (evaluated on dependencies), but
- It provides close to state of the art parsing performance
- It provides VERY fast linear time parsing

Evaluation of Dependency Parsing: (labeled) dependency accuracy



$$\text{Acc} = \frac{\text{\# correct deps}}{\text{\# of deps}}$$

UAS = 4 / 5 = 80%
LAS = 2 / 5 = 40%

Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	dobj

Parsed

1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

Representative performance numbers

- The CoNLL-X (2006) shared task provides evaluation numbers for various dependency parsing approaches over 13 languages
  - MALT: LAS scores from 65–92%, depending greatly on language/treebank
- Here we give a few UAS numbers for English to allow some comparison to constituency parsing

Parser	UAS%
Sagae and Lavie (2006) ensemble of dependency parsers	92.7
Charniak (2000) generative, constituency	92.2
Collins (1999) generative, constituency	91.7
McDonald and Pereira (2005) – MST graph-based dependency	91.5
Yamada and Matsumoto (2003) – transition-based dependency	90.4

Projectivity

- Dependencies from a CFG tree using heads, must be projective
  - There must not be any crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words.
- But dependency theory normally does allow non-projective structures to account for displaced constituents
  - You can't easily get the semantics of certain constructions right without these nonprojective dependencies

Handling non-projectivity

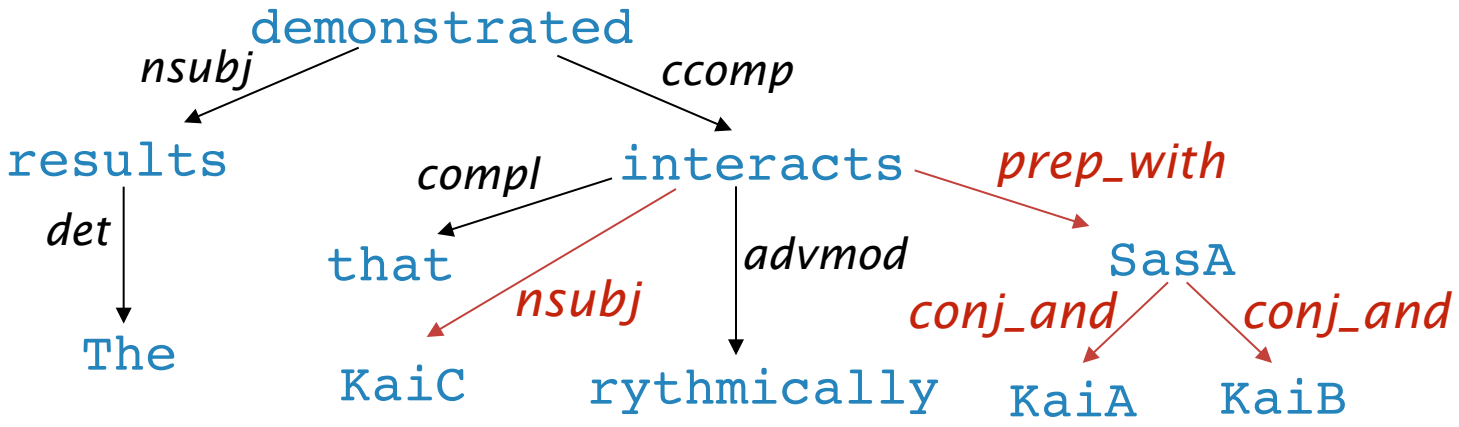
- The arc-eager algorithm we presented only builds projective dependency trees
- Possible directions to head:
  1. Just declare defeat on nonprojective arcs
  2. Use a dependency formalism which only admits projective representations (a CFG doesn't represent such structures…)
  3. Use a postprocessor to a projective dependency parsing algorithm to identify and resolve nonprojective links
  4. Add extra types of transitions that can model at least most non-projective structures
  5. Move to a parsing mechanism that does not use or require any constraints on projectivity (e.g., the graph-based MSTParser)

Dependencies encode relational structure

Relation Extraction with Stanford Dependencies

Dependency paths identify relations like protein interaction

[Erkan et al. EMNLP 07, Fundel et al. 2007]



KaiC ←nsubj interacts prep_with→ SasA

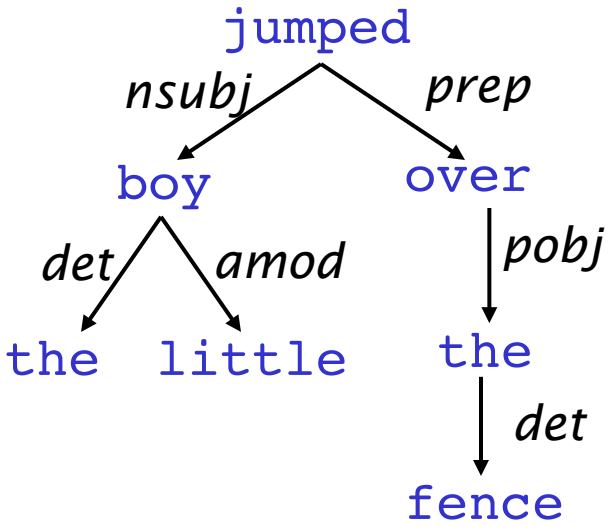
KaiC ←nsubj interacts prep_with→ SasA conj_and→ KaiA

KaiC ←nsubj interacts prep_with→ SasA conj_and→ KaiB

Stanford Dependencies

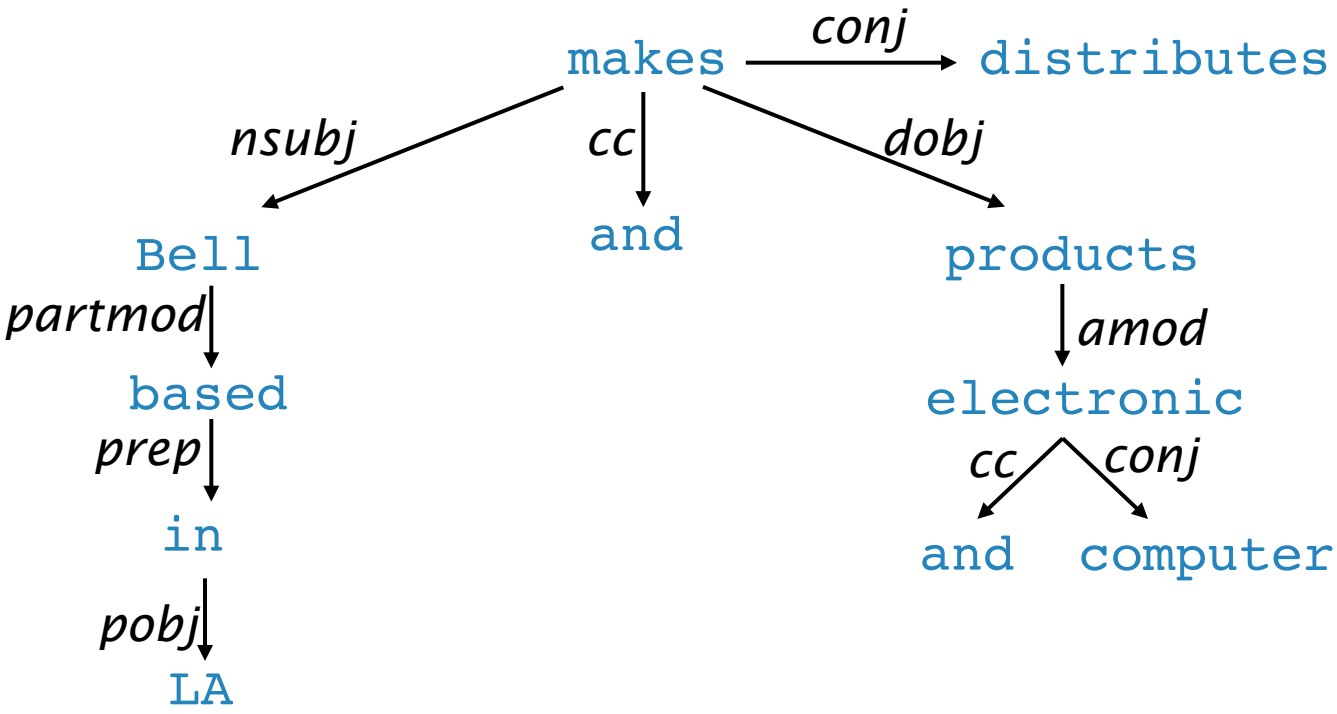
[de Marneffe et al. LREC 2006]

- The basic dependency representation is projective
- It can be generated by postprocessing headed phrase structure parses (Penn Treebank syntax)
- It can also be generated directly by dependency parsers, such as MaltParser, or the Easy-First Parser



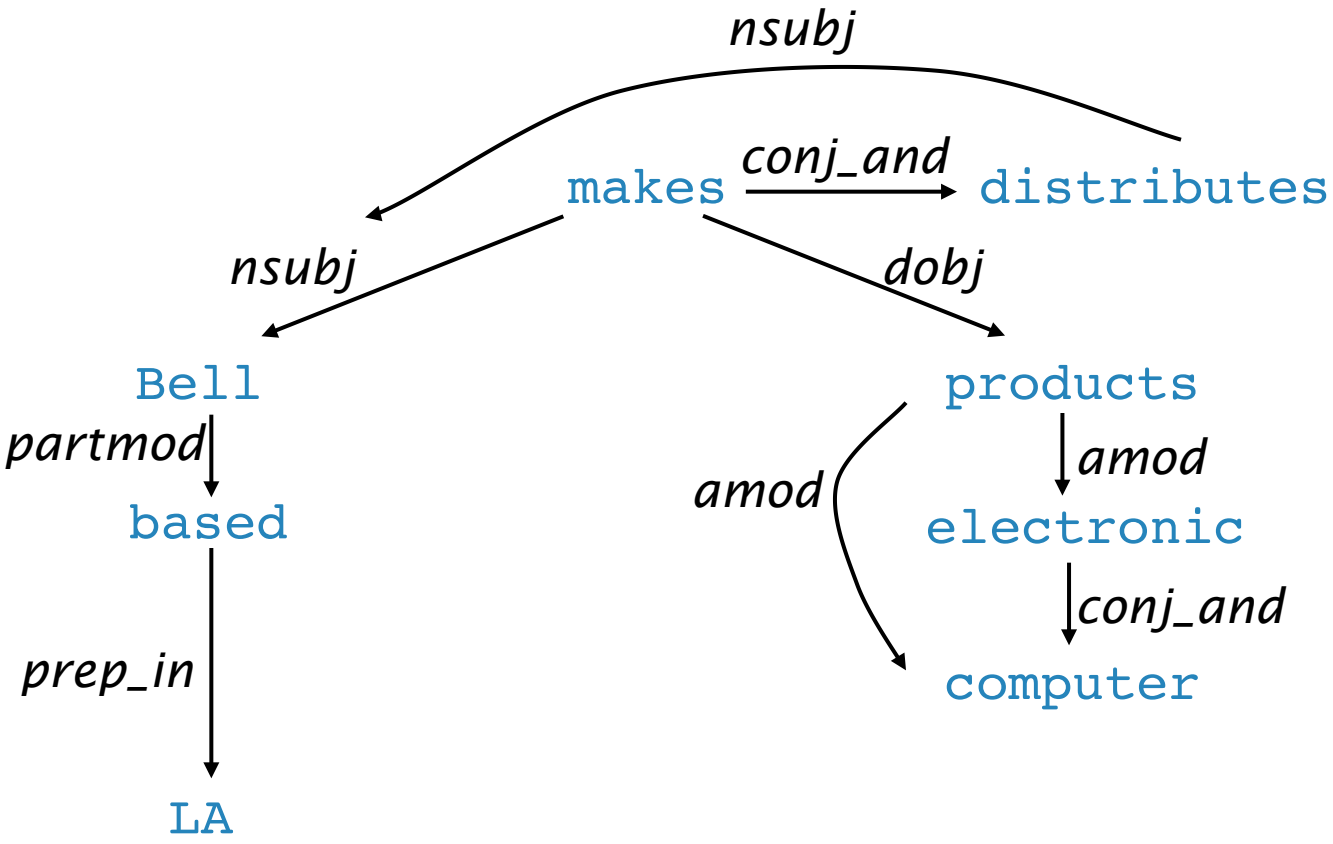
Graph modification to facilitate semantic analysis

Bell, based in LA, makes and distributes electronic and computer products.



Graph modification to facilitate semantic analysis

Bell, based in LA, makes and distributes electronic and computer products.



BioNLP 2009/2011 relation extraction shared tasks [Björne et al. 2009]