Dependency Grammars

Data structures and algorithms
for Computational Linguistics III

Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2019–2020

So far ...

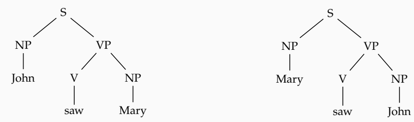
(second part of the course)

- Preliminaries: (formal) languages, grammars and automata
  - Chomsky hierarchy of language classes
  - Expressivity and computational complexity
  - Learnability
- Finite state automata, regular languages, regular grammars and regular expressions
  - DFA, NFA, determinization
  - Closure properties of regular languages
  - Minimization
- Finite state transducers and their applications in CL
- Constituency parsing (CKY, Earley)

Next ...

- Dependency grammars, and dependency treebanks
- Dependency parsing
  - Transition based dependency parsing (with a short introduction to classification)
  - Graph based dependency parsing

Why do we need syntactic parsing?



- Syntactic analysis is an intermediate step in (semantic) interpretation of sentences
- It is essential for understanding and generating natural language sentences (hence, also useful for applications like *question answering*, *information extraction*, ...)
- (Statistical) parsers are also used as *language models* for applications like *speech recognition* and *machine translation*
- It can be used for *grammar checking*, and can be a useful tool for linguistic research

Ingredients of a parser

- A grammar
- An algorithm for parsing
- A method for ambiguity resolution

Phrase structure (or constituency) grammars

The main idea is that a *span* of words form a natural unit, called a *constituent* or *phrase*.

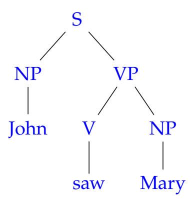
- Constituency grammars are common in modern linguistics (also in computer science)
- Most are based on a context-free 'backbone', extensions or restricted forms are common

An example: constituency grammar in action

Grammar

$$S \rightarrow NP\ VP \qquad VP \rightarrow V\ NP$$
$$NP \rightarrow John \mid Mary \qquad V \rightarrow saw$$

Parse tree



Derivations

$$\begin{aligned} S &\Rightarrow NP\ VP \\ &\Rightarrow John\ VP \\ &\Rightarrow John\ V\ NP \\ &\Rightarrow John\ saw\ NP \\ &\Rightarrow John\ saw\ Mary \end{aligned}$$

or, $S \overset{*}{\Rightarrow} John\ saw\ Mary$

An exercise

- Write down simple (phrase structure) grammar rules for parsing the sentence

  `I read a good book during the break`

  and construct the parse tree
- Repeat the same for a (more-or-less direct) translation of the same sentence in another language
- How about the following sentence?

  `During the break, I read a good book`

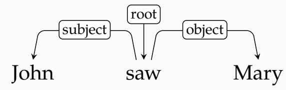
Where do grammars come from?

- Grammars for (constituency) parsing can be either
  - hand crafted (many years of expert effort)
  - extracted from *treebanks* (which also require lots of effort)
  - 'induced' from raw data (interesting, but not as successful)
- Current practice relies mostly on treebanks
- Hybrid approaches also exist
- Grammar induction is not common (for practical models), but exploiting unlabeled data for improving parsing is also a common trend

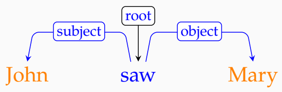
Dependency grammars

introduction

- Dependency grammars gained popularity in linguistics (particularly in CL) rather recently
- They are old: roots can be traced back to Pāṇini (approx. 5th century BCE)
- Modern dependency grammars are often attributed to Tesnière 1959
- The main idea is capturing the relations between words, rather than grouping them into (abstract) constituents

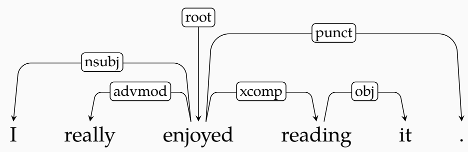


Dependency grammars

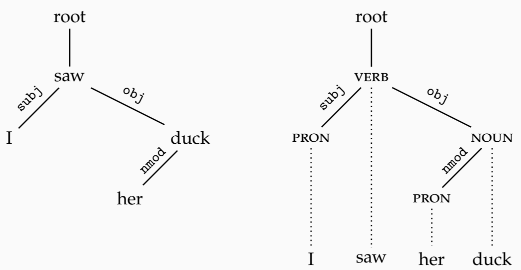


- No constituents, units of syntactic structure are words
- The structure of the sentence is represented by *asymmetric*, *binary* relations between syntactic units
- Each relation defines one of the words as the head and the other as dependent
- Typically, the links (relations) have labels (dependency types)
- Often an artificial *root* node is used for computational convenience

A more realistic example



Dependency grammars: alternative notation



Dependency grammar: definition

A dependency grammar is a tuple (V, A)

- V is a set of nodes corresponding to the (syntactic) words (we implicitly assume that words have indexes)
- A is a set of arcs of the form (w_i, r, w_j) where
  - $w_i \in V$ is the head
  - r is the type of the relation (arc label)
  - $w_j \in V$ is the dependent

This defines a directed graph.

Dependency grammars: common assumptions

- Every word has a single head
- The dependency graphs are acyclic
- The graph is connected
- With these assumptions, the representation is a tree
- Note that these assumptions are not universal but common for dependency parsing

How to determine heads

1. *Head* (H) determines the syntactic category of the *construction* (C) and can often replace C
2. H determines the semantic category of C; the *dependent* (D) gives semantic specification
3. H is obligatory, D may be optional
4. H selects D and determines whether D is obligatory or optional
5. The form and/or position of dependent is determined by the head
6. The form of D depends on H
7. The linear position of D is specified with reference to H

(from Kübler, McDonald, and Nivre 2009, p.3–4)

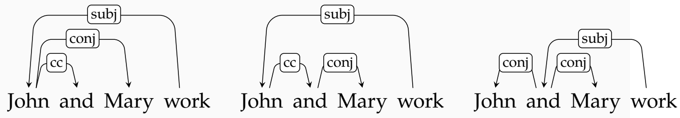
Issues with head assignment and dependency labels

- Determining heads are not always straightforward
- A construction is called *endocentric* if the head can replace the whole construction, *exocentric* otherwise

- It is often unclear whether dependency labels encode syntactic or semantic functions

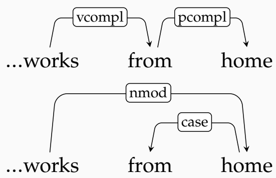
Some tricky constructions

Coordination



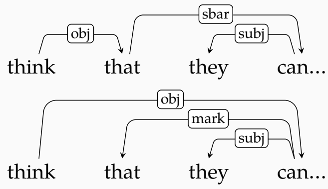
Some tricky constructions

Adpositional phrases



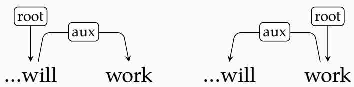
Some tricky constructions

Subordinate clauses

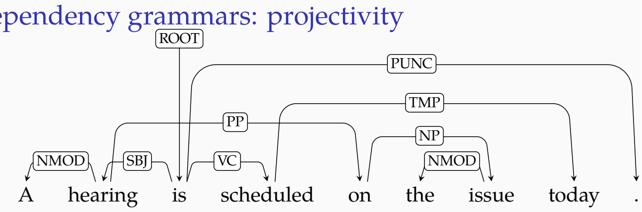


Some tricky constructions

Auxiliaries vs. main verbs



Dependency grammars: projectivity

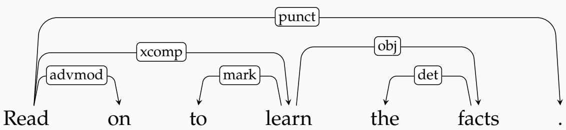


- If a dependency graph has no crossing edges, it is said to be *projective*, otherwise *non-projective*
- Non-projectivity stems from long-distance dependencies and free word order
- Projective dependency trees can be represented with context-free grammars
- In general, projective dependencies are parseable more efficiently

CONLL-X/U format for dependency annotation

Single-head assumption allows flat representation of dependency trees

```
1  Read  read  VERB  VB  Mood=Imp|VerbForm=Fin 0 root
2  on    on    ADV   RB  _                     1 advmod
3  to    to    PART  TO  _                     4 mark
4  learn learn VERB  VB  VerbForm=Inf          1 xcomp
5  the   the   DET   DT  Definite=Def          6 det
6  facts fact  NOUN  NNS Number=Plur           4 obj
7  .     .     PUNCT .   _                     1 punct
```



example from English Universal Dependencies treebank

Dependency parsing

- Dependency parsing has many similarities with context-free parsing (e.g., trees)
- They also have some different properties (e.g., number of edges and depth of trees are limited)
- Dependency parsing can be
  - grammar-driven (hand crafted rules or constraints)
  - data-driven (rules/model is learned from a treebank)
- There are two main approaches:

Graph-based similar to context-free parsing, search for the best tree structure

Transition-based similar to shift-reduce parsing (used for programming language parsing), but using greedy search for the best transition sequence

Grammar-driven dependency parsing

- Grammar-driven dependency parsers typically based on
  - lexicalized CF parsing
  - constraint satisfaction problem
    - start from fully connected graph, eliminate trees that do not satisfy the constraints
    - exact solution is intractable, often employ heuristics, approximate methods
    - sometimes 'soft', or weighted, constraints are used
  - Practical implementations exist
- Our focus will be on data-driven methods

Dependency grammars

Advantages and disadvantages

+ Close relation to semantics
+ Easier for flexible/free word order
+ Lots, lots of (multi-lingual) computational work, resources
+ Often much useful in downstream tasks
+ More efficient parsing algorithms
− No distinction between modification of head or the whole 'constituent'
− Some structures are difficult to capture, e.g., coordination

Summary

- Dependency grammars are based on *asymmetric*, *binary* relations between syntactic units
- Dependencies are (often) labeled
- Dependency analyses are used more in downstream tasks

Next:

- A hands-on introduction to Universal Dependencies
- Dependency parsing
  - Transition based
  - Graph based

A familiar exercise

- Construct a dependency tree for the sentence

  `I read a good book during the break`
- Repeat the same for a (more-or-less direct) translation of the same sentence in another language
- How about the following sentence?

  `During the break, I read a good book`

References / additional reading material

- Kübler, McDonald, and Nivre (2009, Chapters 1&2)
- The new version of Jurafsky and Martin (2009) also includes a draft chapter on dependency grammars and dependency parsing
- Universal Dependencies web site contains a wide range of information and examples. The tutorial slides at `http://universaldependencies.org/eacl17tutorial/` is a good starting point.

References / additional reading material (cont.)

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. second. Pearson Prentice Hall. ISBN: 978-0-13-504196-3.
Kübler, Sandra, Ryan McDonald, and Joakim Nivre (2009). *Dependency Parsing*. Synthesis lectures on human language technologies. Morgan & Claypool. ISBN: 9781598295962.
Tesnière, Lucien (1959). *Éléments de syntaxe structurale*. Paris: Éditions Klinksieck.