Escuela Politécnica Superior

Master course

**Deep learning for natural language processing**

Unit 2 – Natural language tasks (part III)

# Dependency parsing

**Iván Cantador**, ivan.cantadodor@uam.es

**Francisco Jurado**, francisco.jurado@uam.es

# Contents

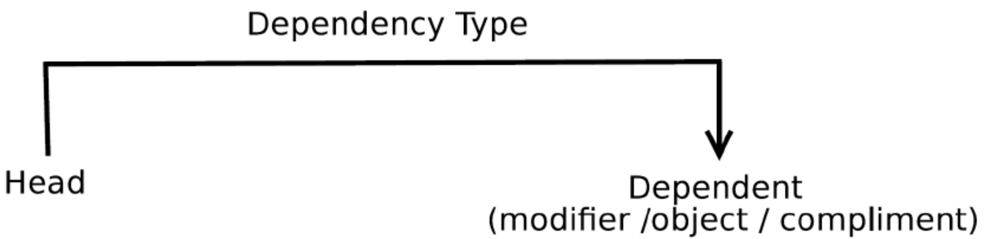


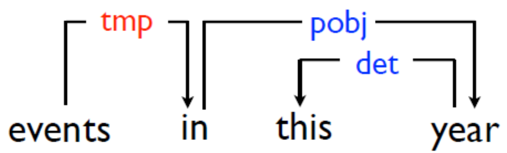
**Disclaimer**: some materials of this presentation come from or is based on the slides and chapters written by Dan Jurafsky and James H. Martin for their book "Speech and Language Processing," and the slide written by Christopher Manning at Stanford University.

# Contents

# Dependency relations

- A **dependency** is an asymmetric syntactic or semantic relation between two lexical tokens
  - Its two tokens are called **head** and **dependent**
  - It may be labelled with a **type**, e.g., *tmp* (temporal), *pobj* (object of a preposition), and *det* (determiner)



# Dependency relations

- The **Universal Dependency (UD) set** – 37 relations, ~200 treebanks, >100 languages
  - **Examples of relations**

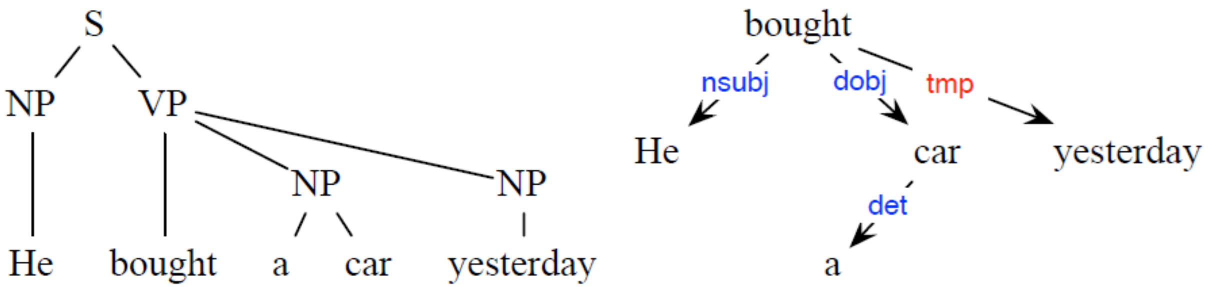
| **Clausal Argument Relations** | **Description** |
|---|---|
| NSUBJ | Nominal subject |
| DOBJ | Direct object |
| IOBJ | Indirect object |
| CCOMP | Clausal complement |
| XCOMP | Open clausal complement |
| **Nominal Modifier Relations** | **Description** |
| NMOD | Nominal modifier |
| AMOD | Adjectival modifier |
| NUMMOD | Numeric modifier |
| APPOS | Appositional modifier |
| DET | Determiner |
| CASE | Prepositions, postpositions and other case markers |
| **Other Notable Relations** | **Description** |
| CONJ | Conjunct |
| CC | Coordinating conjunction |

| Relation | Examples with *head* and **dependent** |
|---|---|
| NSUBJ | **United** *canceled* the flight. |
| DOBJ | United *diverted* the **flight** to Reno. |
| | We *booked* her the first **flight** to Miami. |
| IOBJ | We *booked* **her** the flight to Miami. |
| NMOD | We took the **morning** *flight*. |
| AMOD | Book the **cheapest** *flight*. |
| NUMMOD | Before the storm JetBlue canceled **1000** *flights*. |
| APPOS | *United*, a **unit** of UAL, matched the fares. |
| DET | **The** *flight* was canceled. |
| | **Which** *flight* was delayed? |
| CONJ | We *flew* to Denver and **drove** to Steamboat. |
| CC | We flew to Denver **and** *drove* to Steamboat. |
| CASE | Book the flight **through** *Houston*. |

https://universaldependencies.org/u/dep/index.html

# Dependency relations

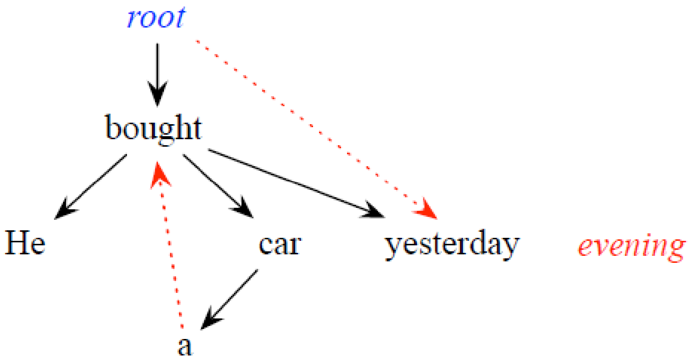
- **Dependency relations** do not correspond to **constituent relations**
  - **Constituent structure**
    - Starts with the bottom level constituents (tokens)
    - Groups smaller constituents into bigger constituents (phrases)
  - **Dependency structure**
    - Builds an acyclic graph (tree) by adding edges between tokens



# Contents

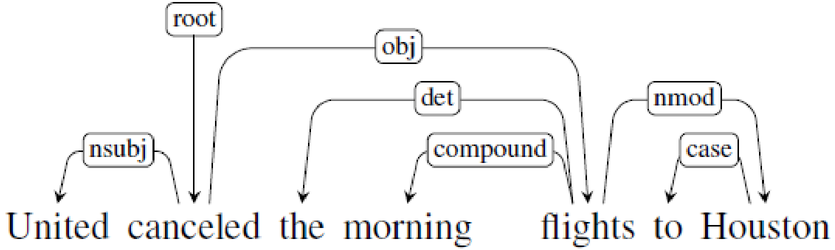
# Dependency trees

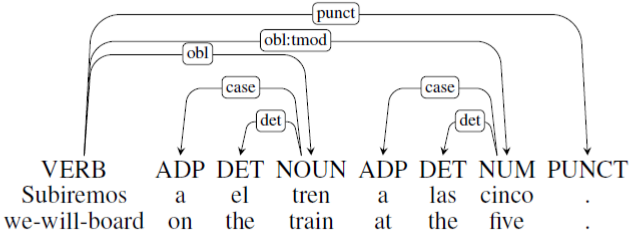
- **Dependency tree**
  - For a **sentence s = $w_1$ … $w_n$**, a dependency tree is a **graph $G_s$ = ($V_s$, $A_s$)** so that
    - $V_s = \{w_0 = \text{root}, w_1, \ldots, w_n\}$
    - $A_s = \{(w_i, r, w_j) : i \neq j, w_i \in V_s, w_j \in V_s - \{w_0\}, r \in R_s\}$
    - $R_s$ = a subset of dependency relations in s
  - It has the following **properties**:
    - **single-head**: each word (vertex) has only one head (incoming arc), except for the root
    - **connected**: contains all the words in the sentence (i.e., there is a path from the root to each word)
    - **acyclic**



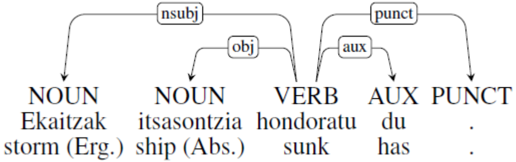
# Dependency trees

- **Treebanks - labeled dependency trees**

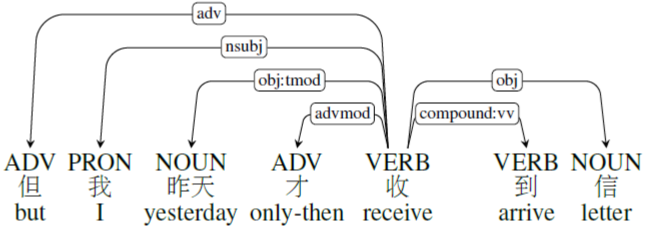



[Spanish] Subiremos al tren a las cinco. “We will be boarding the train at five.”



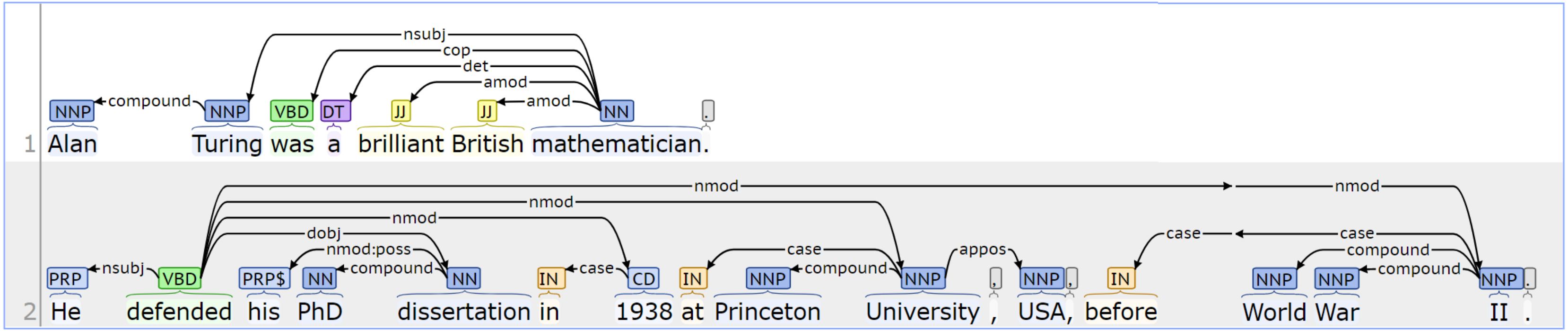

[Basque] Ekaitzak itsasontzia hondoratu du. “The storm has sunk the ship.”




[Chinese] 但我昨天才收到信 “But I didn’t receive the letter until yesterday”

# Extraction of dependency trees

- **Dependency parsing** is the task of extracting the dependency tree of a sentence, defining the relationships between "head" and "dependent" words
  - Example: "Alan Turing was a brilliant British mathematician. He took a leading role in breaking Nazi ciphers during World War II."






Escuela Politécnica Superior

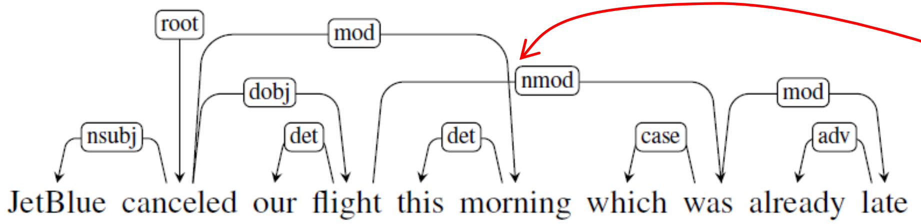
# Extraction of dependency trees

- **Dependency parsing**
  - Provides useful information for many **NLP applications**: machine translation, information extraction, question answering, sentiment analysis, etc.
  - Is **faster** (about 1 millisecond per sentence) and more **language-independent** than most other parsing tasks

**Projective trees** make computation easier, but most theoretical frameworks do not assume projectivity; there is need to capture long-distance dependencies and free word order in some languages.

# Projectivity in dependency trees

- **Projectivity** is an additional property of a dependency tree; if it is satisfied, the tree is *well-formed*
  - An **arc** from a head to a dependent is said to be **projective** <u>if there is a path from the head to every word that lies between the head and the dependent in the sentence</u>
    - If word A depends on word B, then all words between A and B are also subordinate to B (i.e., dominated by B)
  - A **dependency tree** is said to be **projective** if all its arcs are projective
    - Example: the following tree is <u>non-projective</u> since there is no path from "flight" to the intervening words "this" and "morning" (which are before "was")




**Trick**

A **projective dependency tree** has no "crossing" arc.

# Contents

# Types of dependency parsers

- **Transition-based parsers**
  - Perform a **local, greedy parsing**
  - Have problems when the heads are very far from the dependents
  - Tend to produce non-projective trees, which is not a significant issue in English, but it is a problem for many other languages
- **Graph-based parsers**
  - Perform a **global, exhaustive parsing**
  - Are more accurate, especially on long sentences, since they score entire trees
  - Can produce non-projective trees

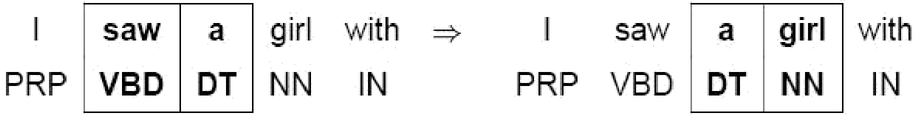
# Types of dependency parsers

- **Transition-based parsers**
  - **Transition**: an operation (e.g., *shift*, *reduce*) that searches for a dependency relation between each pair of words
  - Greedy search that finds *local optima* (locally optimized transitions) → do better for **local dependencies**
  - Projective: O(n), non-projective: $O(n^2)$
- **Graph-based parsers**
  - Build a **complete graph** with **directed/weighted edges** and find on it a spanning tree with the highest score (sum of all weighted edges)
  - Exhaustive search that finds for the *global optimum* (maximum spanning tree) → do better for **long-distance dependencies**
  - $O(n^3)$

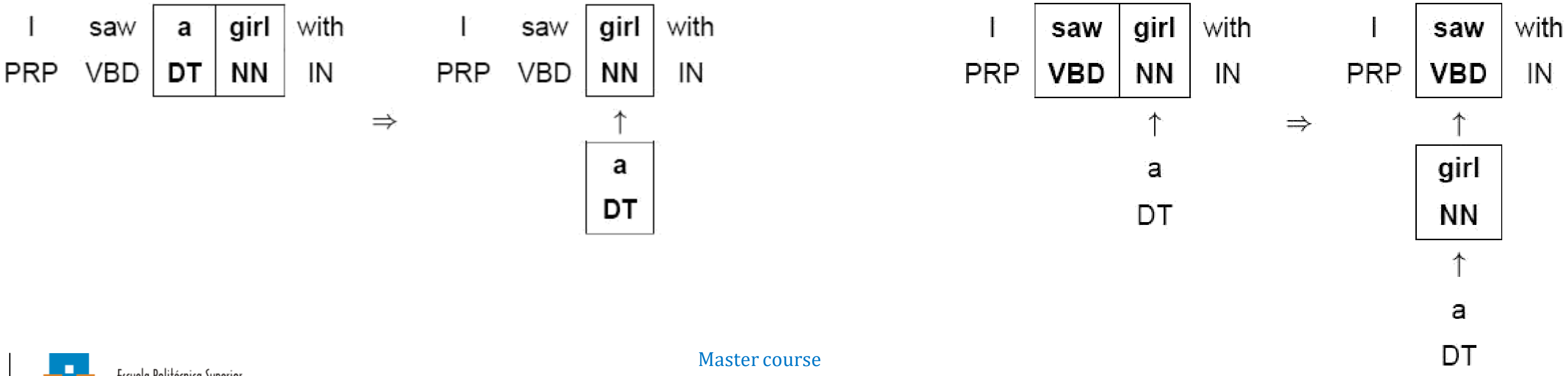
# Contents

# Transition-based dependency parsing

- Local, greedy methods that focus on **pairs of words** and are based on 3 main **actions**: shift, left, right
  - **Shift**: moves the focus to the next word pair

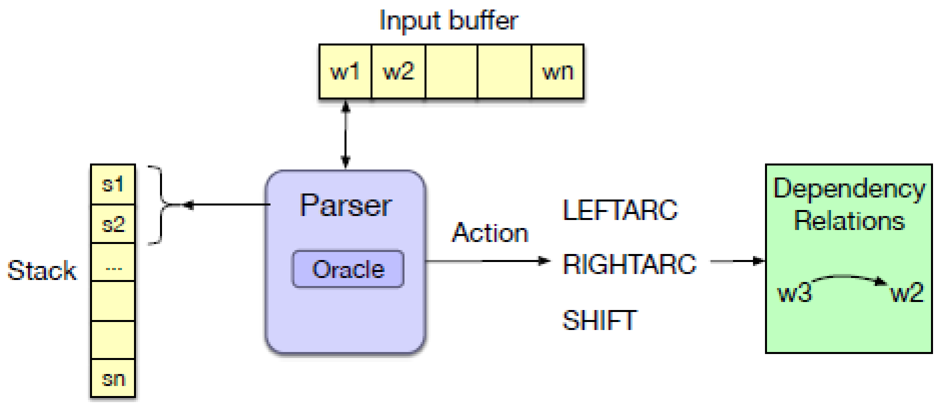



  - **Left/Right**: decides if the left/right word depends on the right/left word



# Transition-based dependency parsing

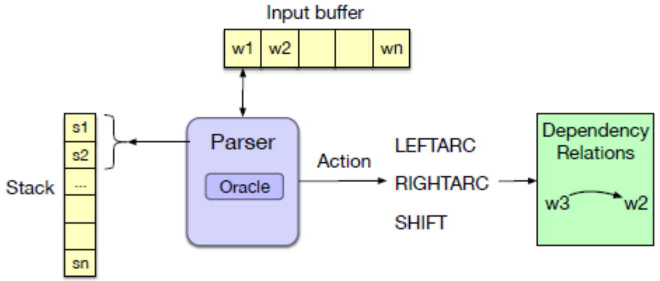
- Builds on a stack-based approach called **shift-reduce parsing** originally developed for analyzing programming languages (Aho & Ullman, 1972)
- **Configuration**
  - **Stack**
  - **Input: buffer (queue) of words**
  - **Output: set of dependency relations**
- **Goal**
  - Finding a final configuration where all words accounted for relations form dependency tree




The parser examines the top two elements **s1** and **s2** of the stack, and selects an action based on consulting an **oracle** that examines the current configuration.

# Transition-based dependency parsing

- **Transitions** produce a new configuration given the current configuration
- **Parsing** is the task of finding a sequence of transitions that leads from the start state to the desired end state
  - **Start state**
    - **Stack**: initialized with ROOT node
    - **Buffer**: initialized with words in sentence
    - **Dependency relation set**: empty
  - **End state**
    - **Stack**: empty
    - **Buffer**: empty
    - **Dependency relations set**: final parse

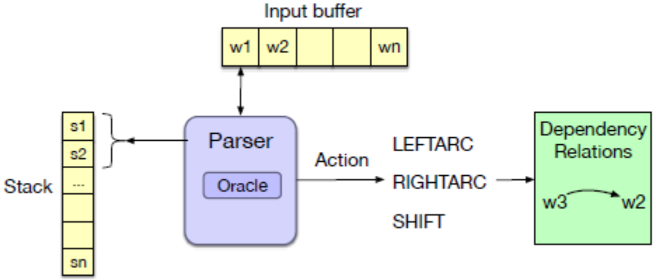


# Transition-based dependency parsing

- **A generic transition-based dependency parser**
  - Assumes the existence of an **oracle**
  - Follows a greedy algorithm
  - Has linear complexity

```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
    t ← ORACLE(state)        ; choose a transition operator to apply
    state ← APPLY(t, state)  ; apply it, creating a new state
  return state
```

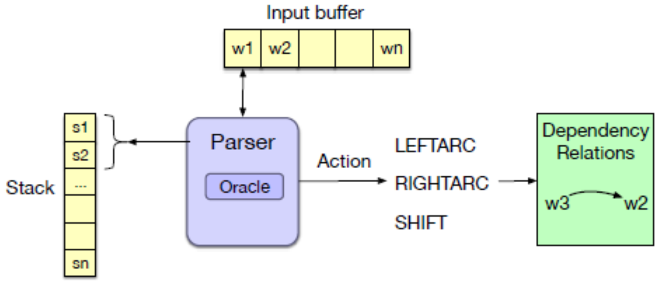


# Transition-based dependency parsing

- **A generic transition-based dependency parser**
  - Given a current configuration with stack $S$, dependency relations $R_c$, and reference parse $R_P$, **the oracle** chooses transitions as follows:

$$
\begin{aligned}
&\text{LEFTARC(r): } \textbf{if } (S_1\ r\ S_2) \in R_p \\
&\text{RIGHTARC(r): } \textbf{if } (S_2\ r\ S_1) \in R_p \textbf{ and } \forall r', w\ s.t.(S_1\ r'\ w) \in R_p \textbf{ then } (S_1\ r'\ w) \in R_c \\
&\text{SHIFT: } \textbf{otherwise}
\end{aligned}
$$

where $S_1$ and $S_2$ are the 1st (top) the 2nd words in the stack respectively



# Transition-based dependency parsing

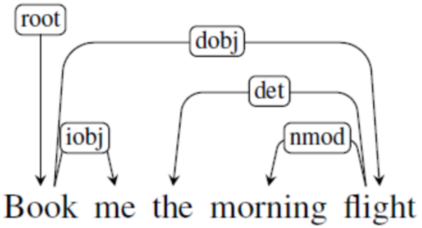
- **Transition operators**
  - **LEFT-ARC**
    - creating a head-dependent relation $s_2 \leftarrow s_1$ between the word $s_1$ at the top of the stack and the word $s_2$ under top
    - removing $s_2$ from stack
  - **RIGHT-ARC**
    - creating head-dependent relation $s_2 \rightarrow s_1$ between $s_2$ and $s_1$
    - removing $s_1$ from stack
  - **SHIFT**
    - removing the word $w_1$ at the head of the buffer
    - pushing $w_1$ on the stack

**Preconditions**

- ROOT cannot have incoming arcs
- LEFT-ARC cannot be applied when ROOT is the 2$^{nd}$ element in the stack
- LEFT-ARC and RIGHT-ARC require two elements in the stack to be applied

# Transition-based dependency parsing

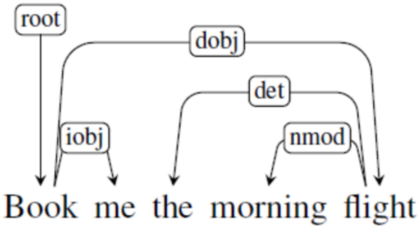
- **Trace of an example transition-based parse**




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | | |
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

# Transition-based dependency parsing

- **Trace of an example transition-based parse**




| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | SHIFT | |
| 1 | [root, book] | [me, the, morning, flight] | SHIFT | |
| 2 | [root, book, me] | [the, morning, flight] | RIGHTARC | (book → me) |
| 3 | [root, book] | [the, morning, flight] | SHIFT | |
| 4 | [root, book, the] | [morning, flight] | SHIFT | |
| 5 | [root, book, the, morning] | [flight] | SHIFT | |
| 6 | [root, book, the, morning, flight] | [] | LEFTARC | (morning ← flight) |
| 7 | [root, book, the, flight] | [] | LEFTARC | (the ← flight) |
| 8 | [root, book, flight] | [] | RIGHTARC | (book → flight) |
| 9 | [root, book] | [] | RIGHTARC | (root → book) |
| 10 | [root] | [] | Done | |

# Transition-based dependency parsing

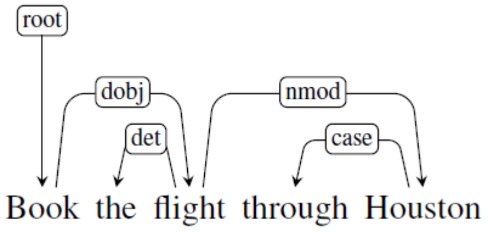
- **Comments**
  - Due to **ambiguity**, there may be several transition sequences that lead to **different valid parses**
  - Given the **greedy nature** of the algorithm, the oracle may provide a **wrong operator** at some point in the parse
    - There are techniques that allow transition-based techniques to explore the search space more fully
  - To produce labeled trees, LEFT-ARC and RIGHT-ARC should be **parametrize operators** with **dependency labels**, as in LEFT-ARC(nsubj) or RIGHT-ARC(dobj)
    - The job of the oracle is more difficult

# Transition-based dependency parsing

- **Creating the oracle**
  - State-of-the-art transition-based parses use **supervised machine learning** methods to train a classifier which plays the role of the oracle
    - Given appropriate **training data**, these methods learn a function that maps from configurations to transition operators
  - The oracle takes as **input** a configuration and returns as **output** a transition operator
    - To train its classifier, **configurations paired with transition operators** (i.e., LEFT-ARC, RIGHT-ARC or SHIFT) are needed
    - But treebanks consist of **entire sentences paired with their corresponding trees**

# Transition-based dependency parsing

- **Creating the oracle**
  - How to generate **training instances** consisting of [configuration, transition] pairs **from a treebank** composed of sentence parses?
    - By simulating the operations of a parser in the context of a reference dependency tree




| Step | Stack | Word List | Predicted Action |
|---|---|---|---|
| 0 | [root] | [book, the, flight, through, houston] | SHIFT |
| 1 | [root, book] | [the, flight, through, houston] | SHIFT |
| 2 | [root, book, the] | [flight, through, houston] | SHIFT |
| 3 | [root, book, the, flight] | [through, houston] | LEFTARC |
| 4 | [root, book, flight] | [through, houston] | SHIFT |
| 5 | [root, book, flight, through] | [houston] | SHIFT |
| 6 | [root, book, flight, through, houston] | [] | LEFTARC |
| 7 | [root, book, flight, houston ] | [] | RIGHTARC |
| 8 | [root, book, flight] | [] | RIGHTARC |
| 9 | [root, book] | [] | RIGHTARC |
| 10 | [root] | [] | Done |

# Transition-based dependency parsing

- **Creating the oracle**
  - Having generated appropriate [configuration, transition] training instances, useful **features from the configurations** are extracted to train classifiers
    - Focus on top words of the stack, and use of word forms and PoS
    - Consideration of context: word neighbors in stack and buffer

| Stack | Word buffer | Relations |
|---|---|---|
| [root, canceled, flights] | [to Houston] | (canceled → United)<br>(flights → morning)<br>(flights → the) |

$\langle s_1.w = flights, op = shift\rangle$
$\langle s_2.w = canceled, op = shift\rangle$
$\langle s_1.t = NNS, op = shift\rangle$
$\langle s_2.t = VBD, op = shift\rangle$
$\langle b_1.w = to, op = shift\rangle$
$\langle b_1.t = TO, op = shift\rangle$
$\langle s_1.wt = flightsNNS, op = shift\rangle$
$\langle s_1.t \circ s_2.t = NNSVBD, op = shift\rangle$

| Source | Feature templates | | |
|---|---|---|---|
| One word | $s_1.w$ | $s_1.t$ | $s_1.wt$ |
| | $s_2.w$ | $s_2.t$ | $s_2.wt$ |
| | $b_1.w$ | $b_1.w$ | $b_0.wt$ |
| Two word | $s_1.w \circ s_2.w$ | $s_1.t \circ s_2.t$ | $s_1.t \circ b_1.w$ |
| | $s_1.t \circ s_2.wt$ | $s_1.w \circ s_2.w \circ s_2.t$ | $s_1.w \circ s_1.t \circ s_2.t$ |
| | $s_1.w \circ s_1.t \circ s_2.t$ | $s_1.w \circ s_1.t$ | |

# Contents

# Graph-based dependency parsing

- **Searching** for a tree or trees that maximize some score through the **space of possible trees for a given sentence**

$$\hat{T}(S) = \underset{t \in \mathcal{G}_S}{\operatorname{argmax}}\, score(t, S)$$

  - **Edge-factored algorithms**: the score for a tree is based on the scores of the edges that comprise the tree

$$score(t, S) = \sum_{e \in t} score(e)$$

- Employing methods drawn from graph theory to search for optimal solutions

# Graph-based dependency parsing

- **Chu-Liu-Edmonds' algorithm**
  - Based on **maximum spanning tree** (MST) algorithm for weighted, directed graphs
  - Given an input sentence, a **fully-connected, weighted, directed graph** is created
    - The vertices are the input words
    - The directed edges represent all possible head-dependent assignments
    - An additional ROOT vertex is included with outgoing edges directed at all the other vertices
    - The **weights** in the graph reflect the score for each possible head-dependent relation as provided by a model generated from training data
  - Given these weights, a maximum spanning tree of this graph emanating from the ROOT represents the preferred dependency parse for the sentence

# Graph-based dependency parsing

- **Chu-Liu-Edmonds' algorithm**

**function** MAXSPANNINGTREE($G=(V,E)$, *root*, *score*) **returns** *spanning tree*

$F \leftarrow []$
$T' \leftarrow []$
$score' \leftarrow []$
**for each** $v \in V$ **do**
  $bestInEdge \leftarrow \text{argmax}_{e=(u,v) \in E}\ score[e]$
  $F \leftarrow F \cup bestInEdge$
  **for each** $e=(u,v) \in E$ **do**
    $score'[e] \leftarrow score[e] - score[bestInEdge]$

**if** $T=(V,F)$ is a spanning tree **then return** it
**else**
  $C \leftarrow$ a cycle in $F$
  $G' \leftarrow$ CONTRACT($G$, $C$)
  $T' \leftarrow$ MAXSPANNINGTREE($G'$, *root*, *score'*)
  T $\leftarrow$ EXPAND($T'$, $C$)
  **return** $T$

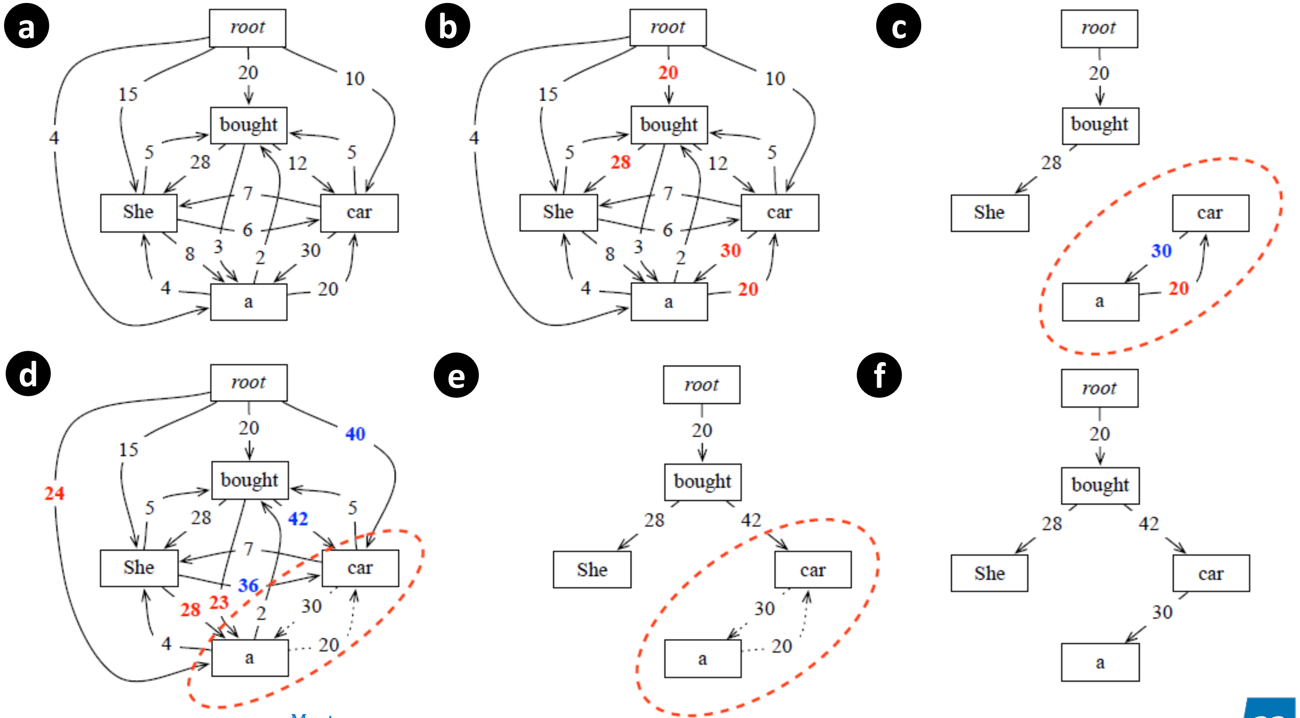
**function** CONTRACT($G$, $C$) **returns** *contracted graph*

**function** EXPAND($T$, $C$) **returns** *expanded graph*

# Graph-based dependency parsing

- **Chu-Liu-Edmonds' algorithm**

1. Keep only incoming edges with the maximum scores
2. If there is a cycle:
   - Pretend vertices in the cycle as one vertex, and update scores for all incoming edges to the cycle
   - Go to #1
3. Break all cycles by removing inappropriate edges in the cycle

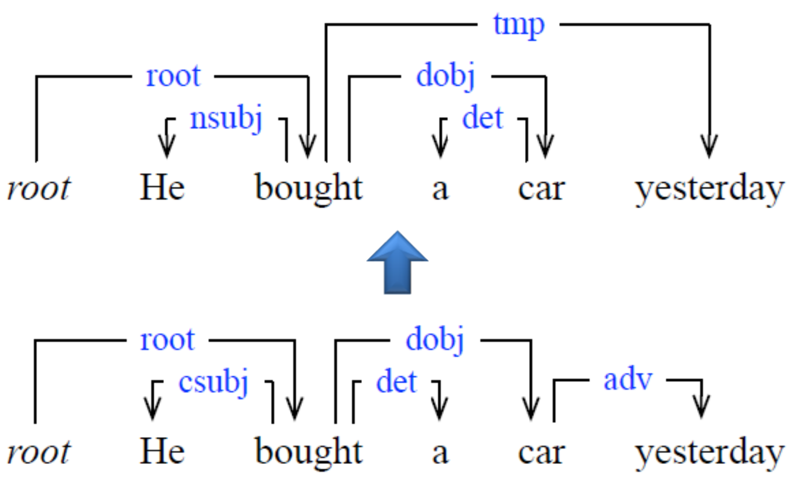


# Contents

# Evaluation of dependency parsers

- **Corpus-scale metrics**
  - Percentage of sentences with the correct root
  - Percentage of correctly parsed sentences
- **Sentence-scale metrics**
  - For each tree, the percentages of words with…
    - **correct heads → unlabeled attachment score (UAS)**
    - **correct labels → label accuracy score (LS)**
    - **correct heads and labels → labeled attachment score (LAS)**

# Evaluation of dependency parsers




- **Metrics – example**
  - **Unlabeled attachment score, UAS** (correct heads)
    - Mismatches: bought → a, car → yesterday
    - UAS = 3/5 = 60%
  - **Label accuracy, LS** (correct labels)
    - Mismatches: He–csubj, yesterday–adv
    - LS = 3/5 = 60%
  - **Labeled attachment score, LAS** (correct heads and labels)
    - Mismatches: He–csubj, bought → a, car → yesterday–adv
    - LAS = 2/5 = 40%

# Evaluation of dependency parsers

- **Metrics – representative performance numbers**
  - The CoNLL-X (2006) shared task provides evaluation numbers for various dependency Parsing approaches over 13 languages
    - LAS scores from 65–92%, depending greatly on language/treebank
    - Some UAS scores for English

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

# Contents

# Resources

- **Typed dependencies manual**
  - https://nlp.stanford.edu/software/dependencies_manual.pdf
- **Repository of dependency parsers, papers and evaluation results**
  - http://nlpprogress.com/english/dependency_parsing.html
- **Stanford parsers**
  - https://nlp.stanford.edu/software/lex-parser.html
  - https://stanfordnlp.github.io/CoreNLP/depparse.html
- **NLTK parsers**
  - https://www.nltk.org/api/nltk.parse.html
  - http://www.nltk.org/howto/dependency.html

# References

- **Jurafsky, D., Martin, J. H.** (2025) **Speech and language processing** (3rd edition – draft January 12, 2025).
  - Chapter 19
- **Eisenstein, J.** (2019). **Introduction to Natural Language Processing**. The MIT Press.
  - Chapter 11

Master course

**Deep learning for natural language processing**

Unit 2 – Natural language tasks (part III)

# Dependency parsing

**Iván Cantador**, ivan.cantadodor@uam.es

**Francisco Jurado**, francisco.jurado@uam.es