LECTURE 18: DEPENDENCY PARSING

Mehmet Can Yavuz, PhD.

Adapted from Julia Hockenmaier, NLP S2023 - course material
https://courses.grainger.illinois.edu/cs447/sp2023/

TODAY'S LECTURE

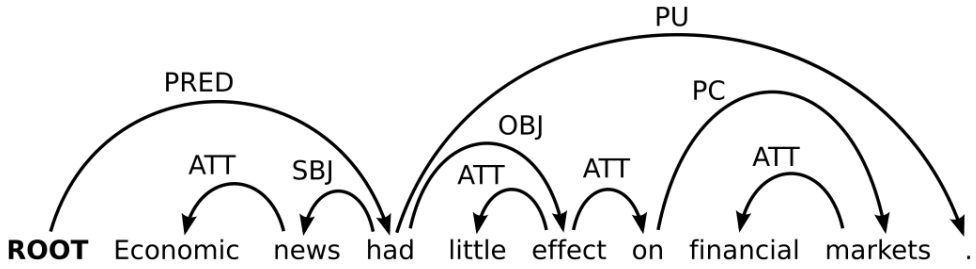
Part 1: Dependency Grammar

Part 2: Dependency Treebanks

Part 3: Dependency Parsing

PART 1: DEPENDENCY GRAMMAR

A DEPENDENCY PARSE



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

had —OBJ—>*effect* [*effect* is the object of *had*]
effect —ATT—> *little* [*little* is an attribute of *effect*]

We typically assume a special ROOT token as word 0

THE POPULARITY OF DEPENDENCY PARSING

Currently the main paradigm for syntactic parsing.

Dependencies are easier to use and interpret
for downstream tasks than phrase-structure trees.

For languages with free word order, dependencies are more natural than phrase-structure grammars

Dependency treebanks exist for many languages.

The Universal Dependencies project has dependency treebanks for dozens of languages that use a similar annotation standard.

DEPENDENCY GRAMMAR

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

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 generative like CFGs etc.), but some formal equivalences are known.

DEPENDENCY TREES

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** **(each word has one parent)**

Together with connectedness, this implies that the graph is a **rooted tree**.

DIFFERENT KINDS OF DEPENDENCIES

Head-argument: *eat sushi*

Arguments may be obligatory, but can only occur once.
The head alone cannot necessarily replace the construction.

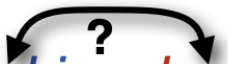
Head-modifier: *fresh sushi*

Modifiers are optional, and can occur more than once.
The head alone can replace the entire construction.

Head-specifier: *the sushi*

Between function words (e.g. prepositions, determiners)
and their arguments. Here, syntactic head ≠ semantic head

Coordination: *sushi and sashimi*



Unclear where the head is.

THERE ISN'T ONE RIGHT DEPENDENCY GRAMMAR

Some constructions can be represented in many different ways.

Different treebanks use different conventions:

Prepositional phrases (sushi [with wasabi])
Use the lexical head (the noun) as head (sushi→wasabi, wasabi→with), or the functional head (thepreposition) (sushi→with, with→wasabi)

Verb clusters, complex tenses (I [will have done] this) Which verb is the head? The main verb (done), or the auxiliaries?

Coordination (eat [sushi and sashimi], [sell and buy] shares) eat→and, and→sushi, and→sashimi

or (e.g.) eat→sushi, sushi→and, sushi→sashimi, etc.

Relative clauses (the cat [that I thought I saw])

These include non-local dependencies (saw-cat) [future lecture]

NB: Some constructions (e.g. coordination, relative clauses) break the assumption that each word has only one parent, and dependency trees cannot represent them correctly.

FROM CFGS TO DEPENDENCIES

Assume each CFG rule has **one head child** (bolded) The other children are **dependents** of the head.

S → NP **VP** **VP** is head, NP is a dependent
VP → **V** NP NP **V** is head, both NPs are dependents

NP → DT **NOUN**
NOUN → ADJ **N**

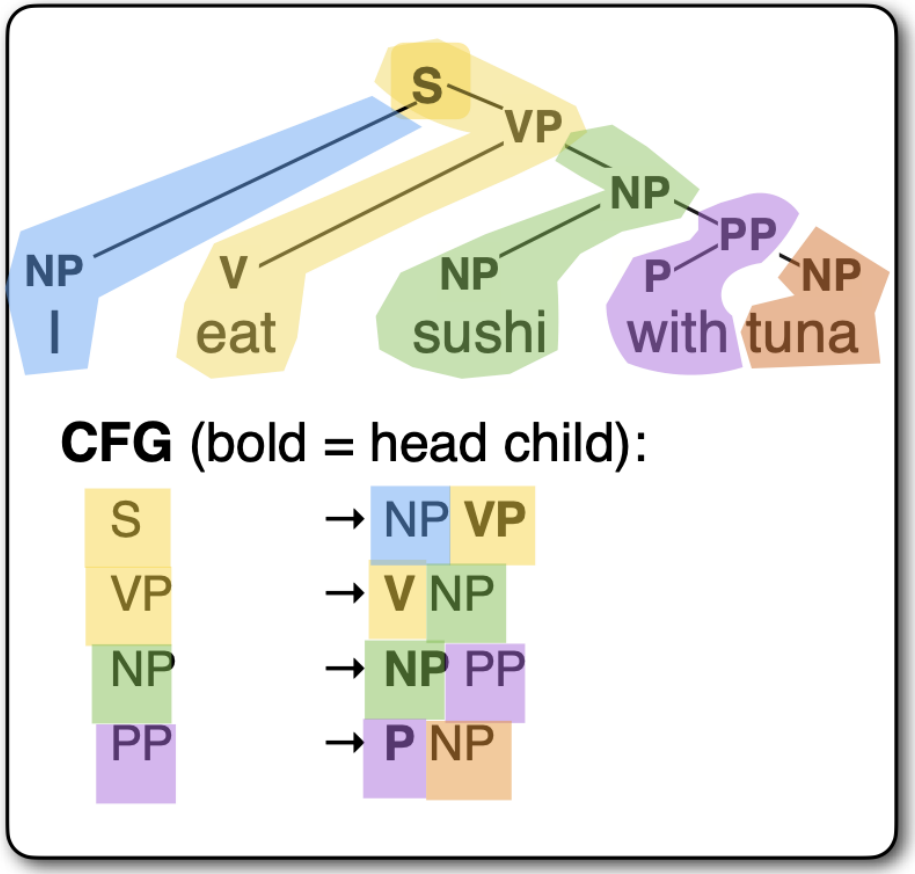
The **headword** of a constituent is the terminal that is reached by recursively following the head child.

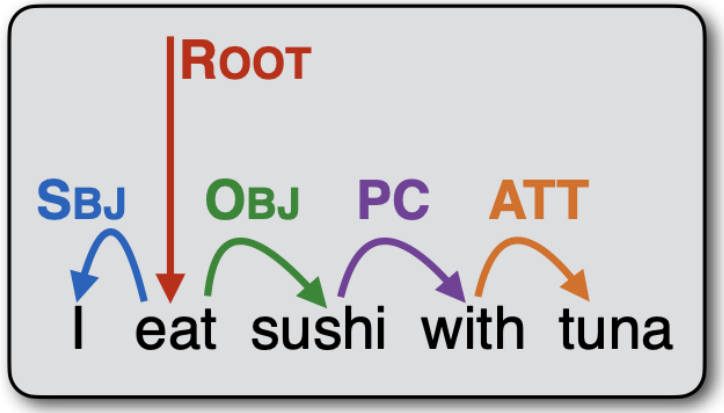
- (here, V is the head word of S, and N is the head word of NP).

If in rule XP → **X** Y, X is head child and Y dependent, the headword of Y depends on the headword of X.

- The **maximal projection** of a terminal *w* is the highest nonterminal in the tree that *w* is headword of.
  Here, Y is a maximal projection.

FROM CFGS TO DEPENDENCIES





Start at the **root** of the tree (S)

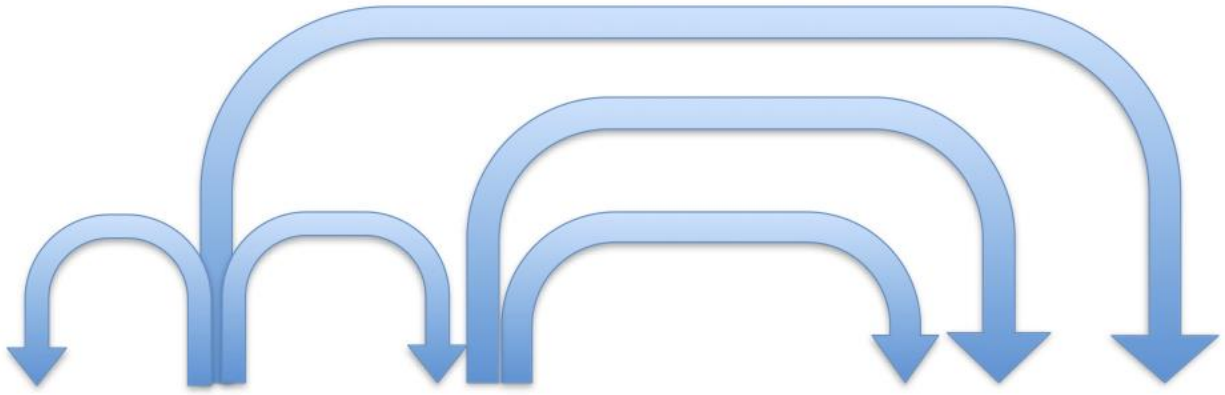
Follow the **head path** (‘**spine**’ of the tree) to the **head word** of the sentence (‘eat’).

Add a ROOT dependency to this word.

For all other **maximal projections**: follow their head paths to get their head words and add the corresponding dependencies

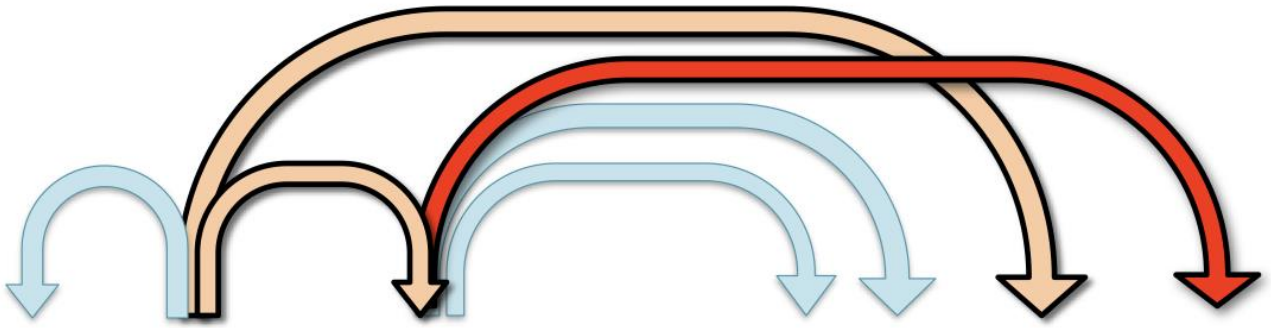
CONTEXT- FREE GRAMMARS

- 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****)*



PART 2: DEPENDENCY TREEBANKS

DEPENDENCY TREEBANKS

Dependency treebanks exist for many languages:

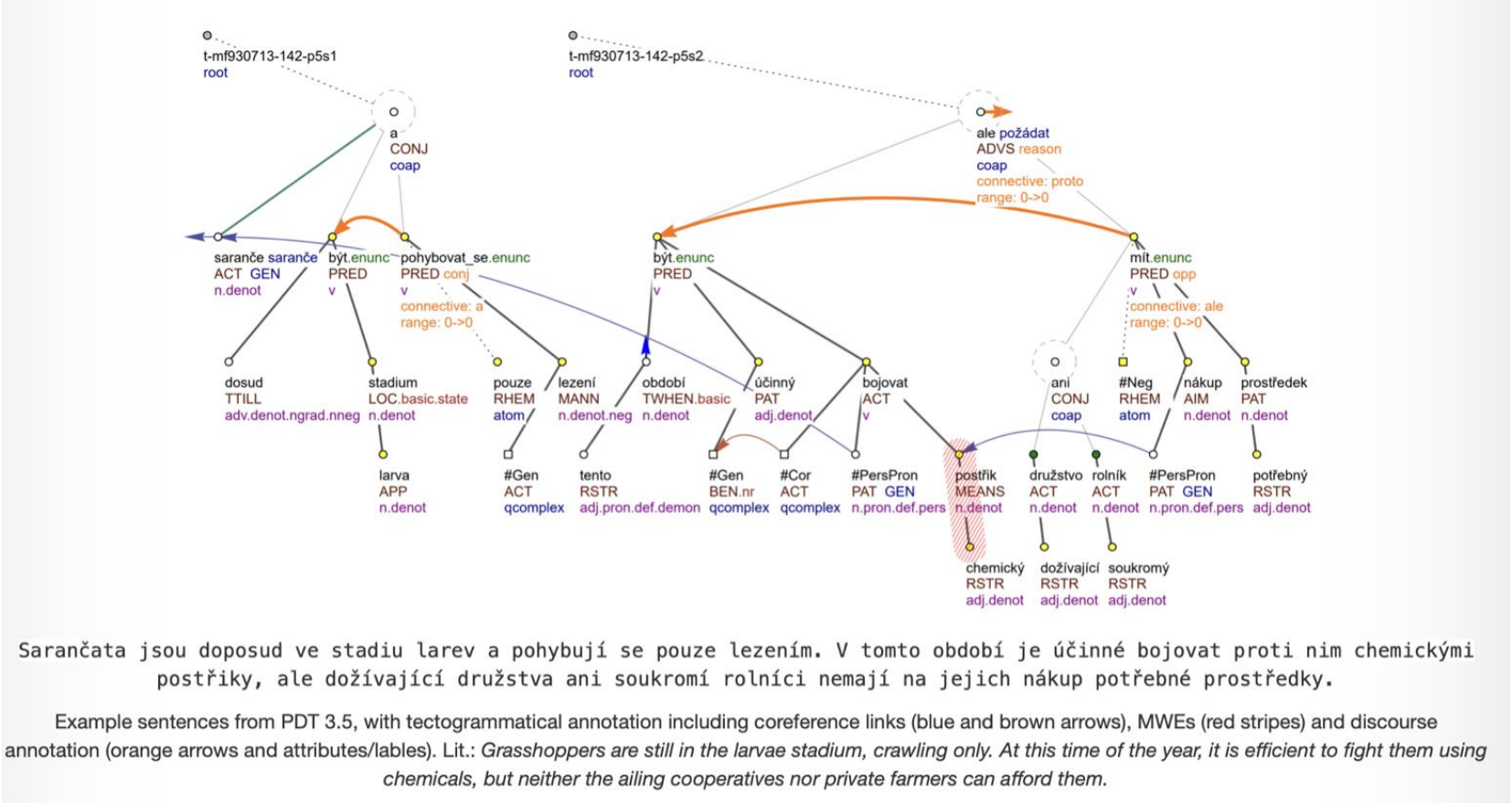
Czech	Arabic	Turkish	Danish	Portuguese	Estonian	….

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

- 2M words, three levels of annotation:
  - **morphological**: Lemma (dictionary form) + detailed analysis (15 categories with many possible values = 4,257 tags)
  - **surface-syntactic ("analytical"):** Labeled dependency tree encoding grammatical functions (subject, object, conjunct, etc.)
  - **semantic ("tectogrammatical"):** Labeled dependency tree for predicate-argument structure, information structure, coreference
    (39 labels: agent, patient, origin, effect, manner, etc....)
    - https://ufal.mff.cuni.cz/pdt3.5

EXAMPLE SENTENCES (PDT3.5)



Sarančata jsou doposud ve stadiu larev a pohybují se pouze lezením. V tomto období je účinné bojovat proti nim chemickými postřiky, ale dožívající družstva ani soukromí rolníci nemají na jejich nákup potřebné prostředky.

Example sentences from PDT 3.5, with tectogrammatical annotation including coreference links (blue and brown arrows), MWEs (red stripes) and discourse annotation (orange arrows and attributes/lables). Lit.: *Grasshoppers are still in the larvae stadium, crawling only. At this time of the year, it is efficient to fight them using chemicals, but neither the ailing cooperatives nor private farmers can afford them.*

METU-SABANCI TURKISH TREEBANK

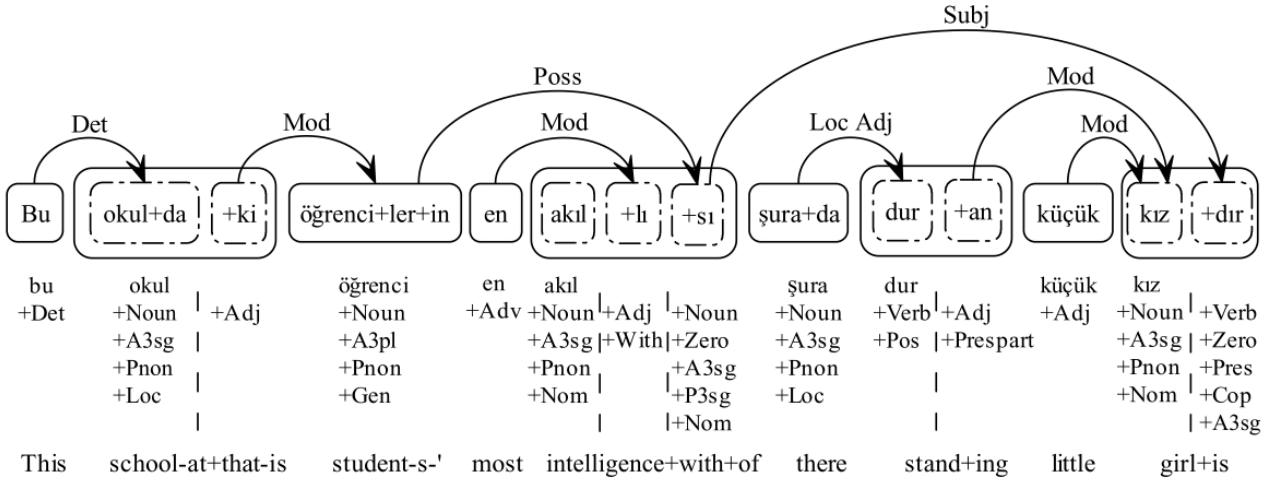


Figure 1
Dependency links in an example Turkish sentence.
'+'s indicate morpheme boundaries. The rounded rectangles show words, and IGs within words that have more than one IG are indicated by the dashed rounded rectangles. The inflectional features of each IG as produced by the morphological analyzer are listed below the IG.

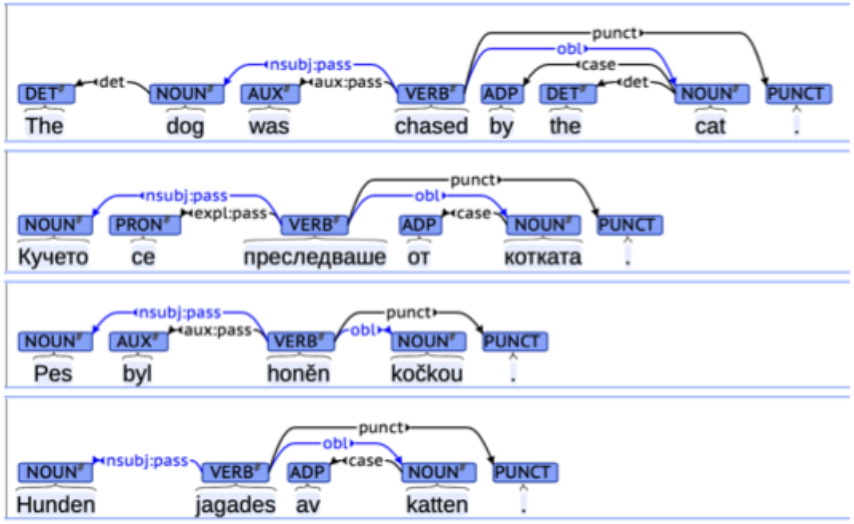
Eryigit, Nivre, and Oflazer, Dependency Parsing of Turkish, CL 2008

UNIVERSAL DEPENDENCIES

- 37 syntactic relations, intended to be applicable to all
- languages (“universal”), with slight modifications for
- each specific language, if necessary.

  http://universaldependencies.org

- Example: “*the dog was chased by the cat*” in English, Bulgarian, Czech and Swedish:
- All languages have dependencies corresponding to (*chased*, nsubj-pass, *dog*)
  (*chased*, obj, *cat*)



UNIVERSAL DEPENDENCY RELATIONS

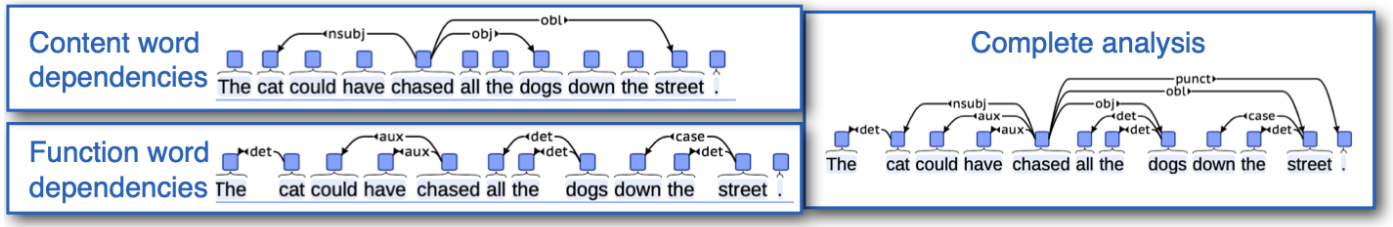
- **Nominal core arguments:** `nsubj` (nominal subject, incl. `nsubj-pass` (nominal subject in passive), `obj` (direct object), `iobj` (indirect object)
- **Clausal core arguments:** `csubj` (clausal subject), `ccomp` (clausal object [“complement”])
- **Non-core (“oblique”) dependents:** `obl` (oblique nominal argument or adjunct, e.g. for tools etc.), `advcl` (adverbial clause modifier),
  `aux` (auxiliary verb), `cop` (copula), `det` (determiner)
- **Nominal dependents:** `nmod` (nominal modifier), `amod` (adjectival modifier), `appos` (appositional modifier)
- **Function words:** `case` (case markers, prepositions), `det` (determiners), **Coordination:** `cc` (coordinating conjunction), `conj` (conjunct)
- **Multiword Expressions:** `compound` (within compound nouns), `flat` (dates, complex names, etc.),
- **Other:** `root` (from ROOT to the head of the sentence), `dep` (catch-all label), `punct` (to punctuation marks)

UD CONVENTIONS: PRIMACY OF CONTENT WORDS

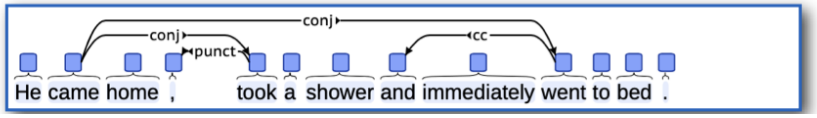
HTTPS://UNIVERSALDEPENDENCIES.ORG/U/OVERVIEW/SYNTAX.HTML

Dependency relations hold primarily between content words
(which vary less across languages than function words)

Function words (prepositions, copulas, auxiliaries, determiners)
attach to the most closely related content word,
and typically don't have dependents

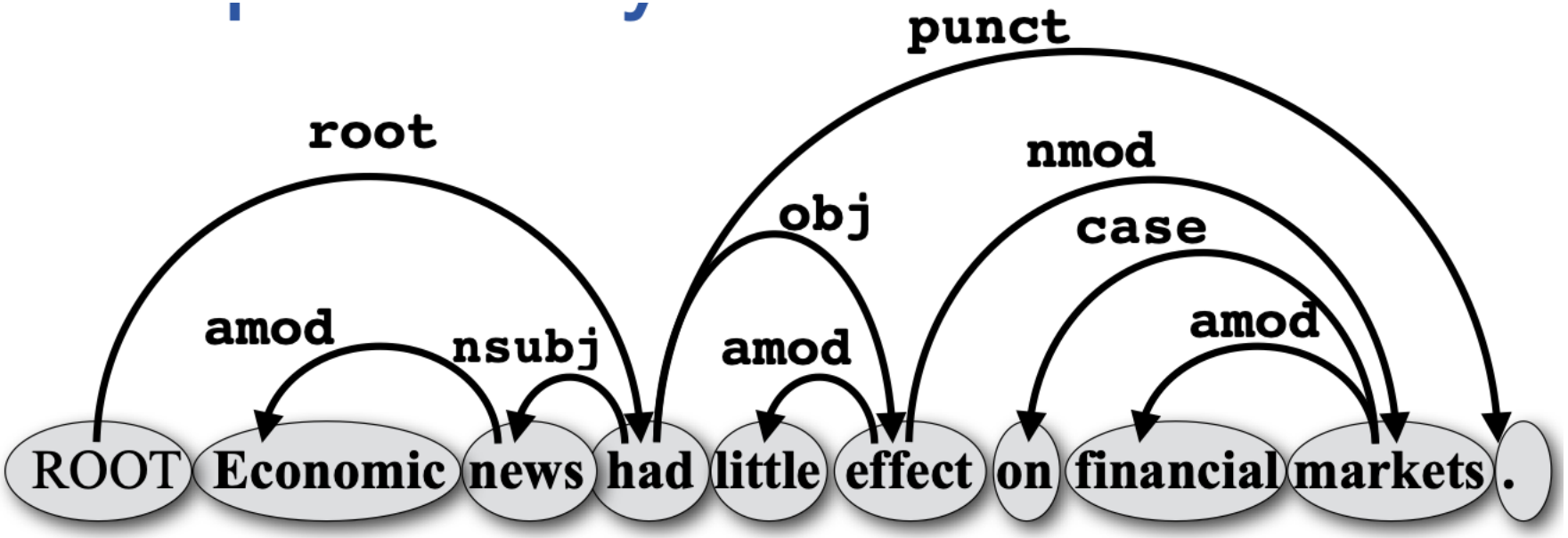


In **coordination**, the first conjunct (*came*) is head, and
the coordination (*and*) and subsequent conjuncts (*took, went*)
depend on the first conjunct:



PART 3: A TRANSITION-BASED DEPENDENCY PARSING ALGORITHM

A DEPENDENCY PARSE



The (non-root) **nodes** of a dependency tree are **the words/tokens** in the sentence.
The **root node** of a dependency tree is a **special token** ROOT
The **edges** of a dependency tree are **dependencies from a head** (parent) **to a dependent** (child)
ROOT has exactly *one* child (the **head of the sentence**).
Each non-root (i.e. each non-root node) has exactly *one* incoming edge (from its **parent/head**)
A complete dependency tree (parse) for a a sentence includes *all* tokens in the sentence.

DEPENDENCY PARSERS

We distinguish two main types of architectures:

‘Transition-based’ (shift-reduce) parsers:

- **Read the sentence incrementally**, word by word Actions (transitions):
- **Shift** (read the next word)
  Reduce (attach one word to another, i.e. add an arc)
- Model: Predict the next action
  Typically return a single, projective, dependency tree

‘Graph-based’ parsers:

- Consider **all words** in the sentence at once.
  Use a **minimum spanning tree algorithm** to find the best tree
  Models: Score each dependency edge
- May return the top *k* trees, including non-projective ones

TRANSITION-BASED PARSING

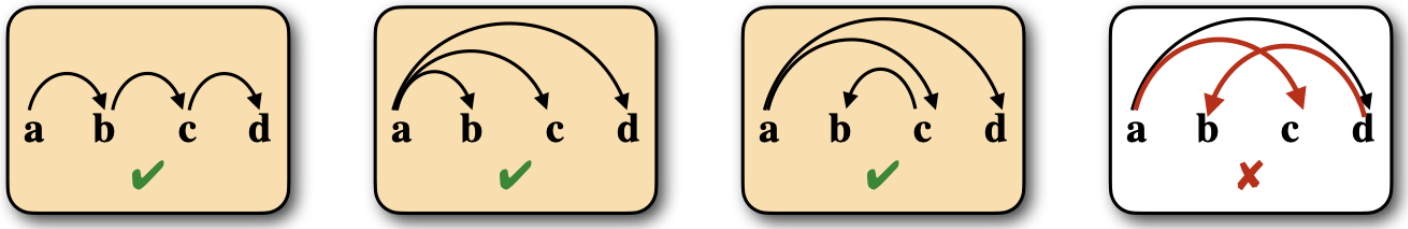
Transition-based or **shift-reduce parsing** is a general framework that can also be used for other grammar formalisms (including CFGs).

Transition-based parsers process sentences from left to right and return a single tree.

The algorithm we will consider can only produce ***projective*** **dependency trees** (no crossing edges)

NB: There exist other (equivalent) variants of this parsing algorithm that use slightly different actions (transitions).

PROJECTIVE DEPENDENCIES



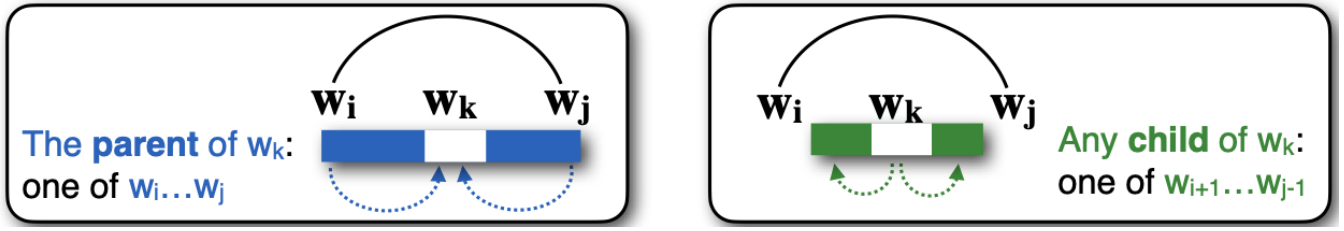
Projective = No crossing dependencies!

Projective 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: $i<m<j$



TRANSITION-BASED PARSING

- The parser processes the sentence **S = w0w1...wn** from left to right (“**incremental parsing**”)
- The parser uses **three data structures**:
- **σ**: a **stack** of partially processed words wi TS
  β: a **buffer** of remaining input words wi TS
  A: a **set of dependency arcs** (wi, l, wj) TS × L ×TS
- w0 is a special ROOT token.
  TS = {w0, w1, ..., wn} are the tokens in the input sentence L is a predefined set of dependency relation labels
- (wi, l, wj) is a dependency with label l from head wi to wj

PARSER CONFIGURATIONS (σ, β, A)

- The **stack σ** is a list of **partially processed** words
- We can *shift* the top word of **β** onto the top of **σ** (grow the stack by one) or remove one of the top two words from **σ** by attaching it to the other top word (this *reduces* the stack by one element)
- **σ**|wj or **σ**|wiwj: wj is the topmost word on the stack. **σ**|wiwj: wi is the second top word on the stack.
- The **buffer β** is the **remaining input** words
  We read words from **β** (left-to-right) and push (‘*shift*’) them onto **σ** w|**β**: w is on top of the buffer. w is the next word to be shifted onto **σ**
- The **set of arcs A** defines the current tree.
  We add a new arc to **A** by attaching the first word on top of the
- stack to the second word on top of the stack or vice versa

PARSER ACTIONS (TRANSITIONS)

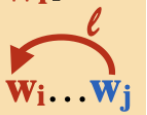
In any configuration (σ, β, A), take one of these actions:

1) **Shift** w_k from buffer β to stack σ:

Shift: $(\sigma, w_k|\beta, A) \Rightarrow (\sigma|w_k, \beta, A)$

2) Add a ***leftwards*** **dependency arc with label** ℓ from w_j to w_i:

LeftArc-ℓ : $(\sigma|w_i w_j, \beta, A) \Rightarrow (\sigma|w_j, \beta, A \cup \{(w_j, \ell, w_i)\})$



w_i (2nd on stack) is a dependent (with label ℓ) of head w_j (1st on stack)

w_i is removed from the stack σ: only do this if w_i has no further children to be attached

3) Add a ***rightwards*** **dependency arc with label** ℓ from w_i to w_j:

Right Arc-ℓ: $(\sigma|w_i w_j, w_k|\beta, A) \Rightarrow (\sigma|w_i, \beta, A \cup \{(w_i, \ell, w_j)\}$

w_j (1st on stack) is a dependent (with label ℓ) of head w_i (2nd on stack).

w_i is removed from the stack σ: only do this if w_i has no further children to be attached

INTERPRETING CONFIGURATION (σ, β, A)

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

w_i and w_j are the top two elements of the stack.
Each may already have some dependents of their own

w_k (top of the buffer) does not have any dependents yet

w_i (2nd on stack) precedes w_j (top of stack): $i < j$
w_j (top of stack) precedes w_k (top of buffer): $j < k$

We have to either **attach** w_i **to** w_j (add a LeftArc),
attach w_j **to** w_i (add a RightArc), or **shift** w_k onto the stack

We can only reach ($\sigma|w_j$, $w_k|\beta$, A) if all words w_l with $j < l < k$
have already been attached to their parent w_m with $j \leq m < k$

PARSER CONFIGURATIONS (σ, β, A)

- We start in the **initial configuration** (**[w0]**, **[w1,..., wn]**, **{}**)
- (**Root token**, **Input Sentence**, **No tree**)
- In the initial configuration, we can only **push w1 onto the stack.**
- We want to end in a **terminal configuration** (**[w0]**, **[]**, **A**) (**Root token**, **Empty buffer**, **Complete tree**)
- In a terminal configuration, we have **read all of the input words** (empty buffer) and we have **attached all input words**.
- (The root **w0** is the only token that can't get attached to any other word)

TRANSITION-BASED PARSING IN PRACTICE

Which action should the parser take

in 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 l: LEFTARC-l, or RIGHT-ARC-l

– **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.

A NEURAL DEPENDENCY PARSER

(Chen and Manning, 2014)

- https://www.aclweb.org/anthology/D14-1082.pdf

Predict the next action in a transition-based parser with a **feedforward network** (with one hidden layer)

Input: Parser configurations (stack, buffer, arcs) represented as a (fixed-sized) list of features.

Output: With L dependency labels, softmax over $(1+2L)$ actions

- (SHIFT, plus 2 actions per label $l \in L$: LEFTARC-l, RIGHTARC-l)