FNLP
Syntactic Analysis II

Yansong Feng
fengyansong@pku.edu.cn

(mainly from slides of Dr. W. Sun)
Wangxuan Institute of Computer Technology
Peking University

April 3, 2025

Outline

1. **Dependency Structures**
   - Constituent and Dependency
   - Dependency Structures
   - Dependency Relations
   - Dependency Graph

2. **Dependency Parsing**
   - Transition-Based Dependency Parsing
   - The Arc-standard Transition Algorithm
   - Evaluation

Outline

Outline

Structures

- the guy who fixed the car carefully packed his tools
- carefully the guy who fixed the car packed his tools
- carefully the guy who fixed the car is tall

Structures

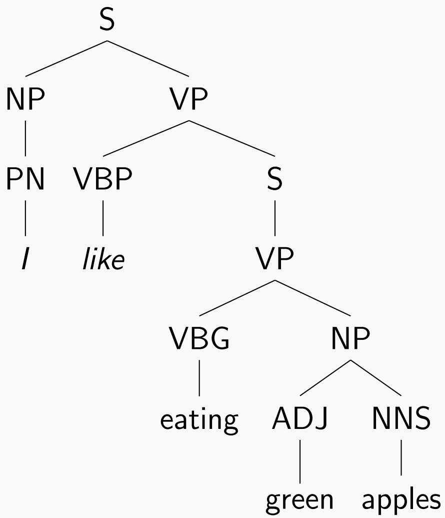
- the guy who fixed the car carefully packed his tools
- carefully the guy who fixed the car packed his tools
- carefully the guy who fixed the car is tall

I think the deepest property of language and puzzling property that's been discovered is what is sometimes called structure dependence. [...] Linear closeness is an easy computation, but here yousre doing a much more, what looks like a more complex computation.
–Noam Chomsky

Structures

I like eating green apples

- *green apples*
- *eating green apples*
- *like eating green apples*
- ...

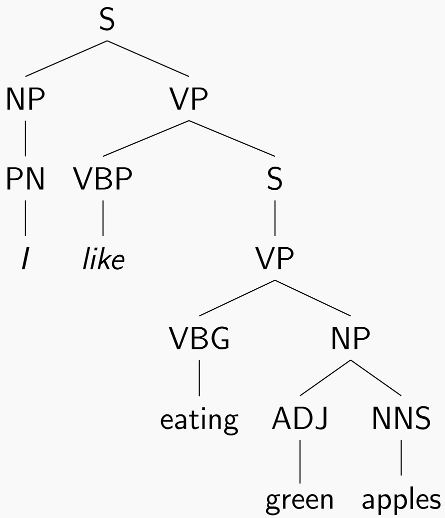


Structures

I like eating green apples

- *green apples*
- *eating green apples*
- *like eating green apples*
- ...

- *I – like*
- *like – apples*
- *apples – eating*
- *apples – green*



Structures

- 您教的都是没用的东西
- 您教的都是没用的东西

Structures

Event: 教

- 您教的都是没用的东西
- 您教的都是没用的东西

Structures

- 您教的都是没用的东西
- 您教的都是没用的东西

Event: 教

- Someone who teaches

Structures

Event: 教

- Someone who teaches
- Someone whom is taught

- 您教的都是没用的东西
- 您教的都是没用的东西

Structures

Event: 教

- Someone who teaches
- Someone whom is taught
- Something that is taught

- 您教的都是没用的东西
- 您教的都是没用的东西

Structures

Event: 教

- Someone who teaches
- Someone whom is taught
- Something that is taught

Predicate-Argument Structures

- 您教的都是没用的东西
- 您教的都是没用的东西

Structures

I like eating green apples

- S, NP, VP, PP, ...
- *green apples*
- *eating green apples*
- *like eating green apples*
- ...

- Predicate-argument or head-dependent, ...
- *I – like*
- *like – apples*
- *apples – eating*
- *apples – green*

Structures

I like eating green apples

- S, NP, VP, PP, ...
- *green apples*
- *eating green apples*
- *like eating green apples*
- ...
- ⇒ Constituents

- Predicate-argument or head-dependent, ...
- *I – like*
- *like – apples*
- *apples – eating*
- *apples – green*

Structures

I like eating green apples

- S, NP, VP, PP, ...
- *green apples*
- *eating green apples*
- *like eating green apples*
- ...
- ⇒ Constituents

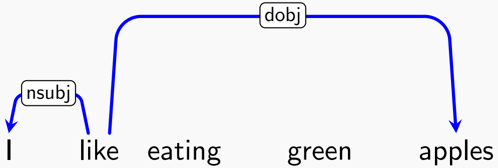
- Predicate-argument or head-dependent, ...
- *I – like*
- *like – apples*
- *apples – eating*
- *apples – green*
- ⇒ Dependency

In the words by Lucien Tesnière

- The sentence is an organized *whole*, the constituent elements of which are *words*.
- Between the word and its neighbors, the mind perceives *connections*, the totality of which forms the structure of the sentence.
- The structural connections establish *dependency relations* between the words. Each connection in principle unites a *superior* term and an *inferior* term.

In the words by Lucien Tesnière

- The sentence is an organized *whole*, the constituent elements of which are *words*.
- Between the word and its neighbors, the mind perceives *connections*, the totality of which forms the structure of the sentence.
- The structural connections establish *dependency relations* between the words. Each connection in principle unites a *superior* term and an *inferior* term.



- lexical items are linked by binary asymmetric relations
- they are called as dependencies

Outline

Dependency Structures

Economic news had little effect on financial markets .

Dependency Structures

Economic news had little effect on financial markets .

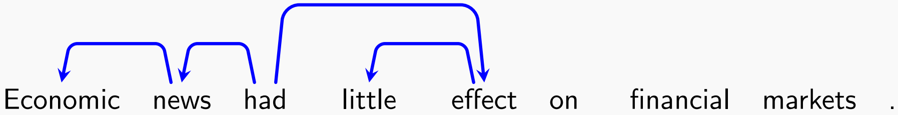
Dependency Structures

Economic news had little effect on financial markets .

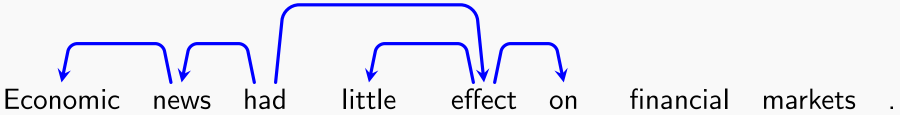
Dependency Structures

Economic news had little effect on financial markets .

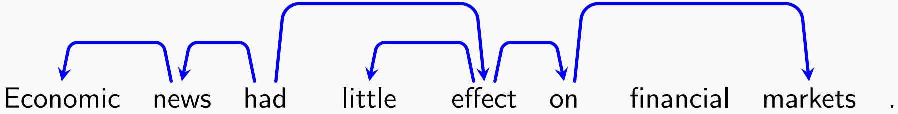
Dependency Structures



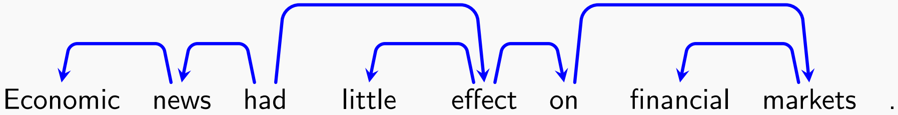
Dependency Structures



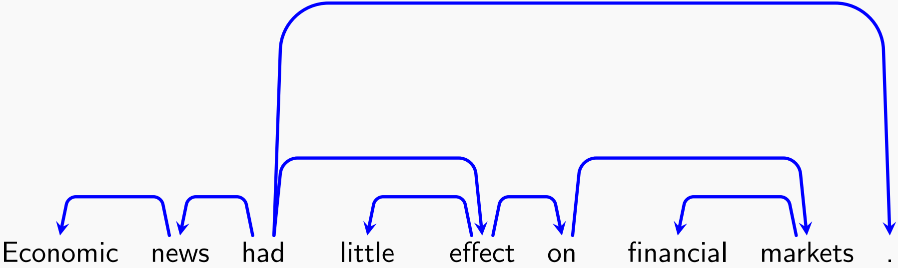
Dependency Structures



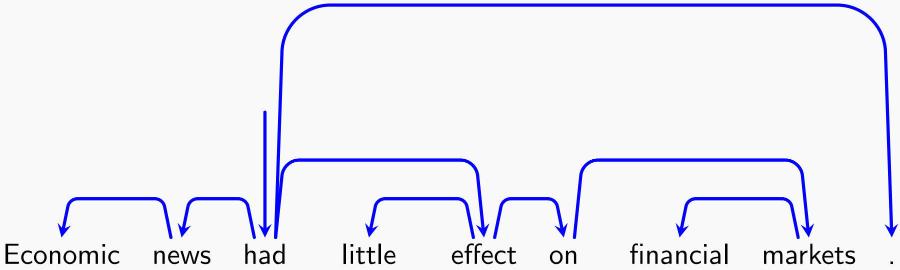
Dependency Structures



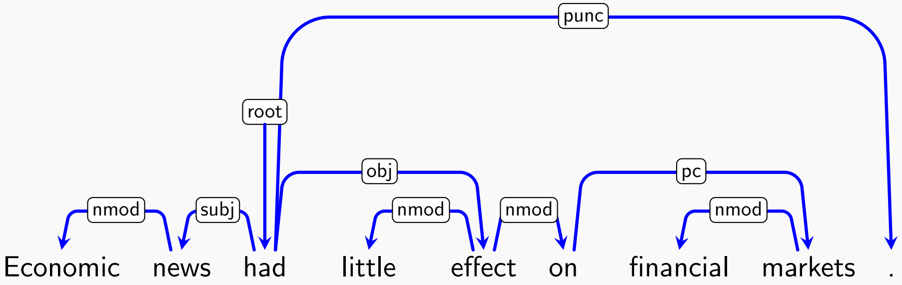
Dependency Structures



Dependency Structures

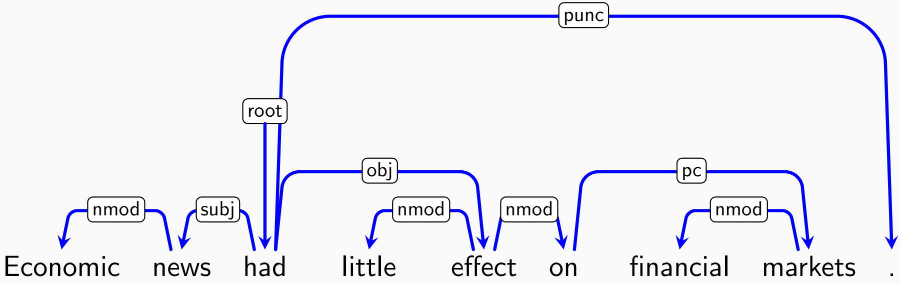


Dependency Structures



Terminology

- Superior: Head/Governor
- Inferior: Dependent/Modifier
- Dependency Relations: Grammatical Relations



Terminology

- Superior: Head/Governor
- Inferior: Dependent/Modifier
- Dependency Relations: Grammatical Relations

Criteria for a syntactic relation between a head H and a dependent D in a construction C:

- H determines the syntactic category of C; H can replace C.
- H determines the semantic category of C; D specifies H.
- H is obligatory; D may be optional.
- H selects D and determines whether D is obligatory.
- The form of D depends on H (agreement or government).
- The linear position of D is specified with reference to H.

Terminology

- Superior: Head/Governor
- Inferior: Dependent/Modifier
- Dependency Relations: Grammatical Relations

Criteria for a syntactic relation between a head H and a dependent D in a construction C:

- H determines the syntactic category of C; H can replace C.
- H determines the semantic category of C; D specifies H.
- H is obligatory; D may be optional.
- H selects D and determines whether D is obligatory.
- The form of D depends on H (agreement or government).
- The linear position of D is specified with reference to H.

Or, see Michael Collins (1999)

Outline

Dependency Relations

Relations to characterize the dependency structures

- binary relations between a **head** and a **dependent**
- **head**: the central organizing word
- **dependent**: often act as a modifier

Dependency Relations

Relations to characterize the dependency structures

- binary relations between a **head** and a **dependent**
- **head**: the central organizing word
- **dependent**: often act as a modifier

Or, indicate

- the grammatical function that a **dependent** plays regarding to its **head**
- *I* is the subject of *like*
- *apple* is the direct object of *like*

Dependency Relations

Relations to characterize the dependency structures

- binary relations between a **head** and a **dependent**
- **head**: the central organizing word
- **dependent**: often act as a modifier

Or, indicate

- the grammatical function that a **dependent** plays regarding to its **head**
- *I* is the subject of *like*
- *apple* is the direct object of *like*

There are many different systems of dependency relations.

Grammatical Relations

the Universal Dependencies (UD) project (de Marneffe et al., 2021)

- 37 types

Clausal Argument Relations	Description
NSUBJ	Nominal subject
OBJ	Direct object
IOBJ	Indirect object
CCOMP	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

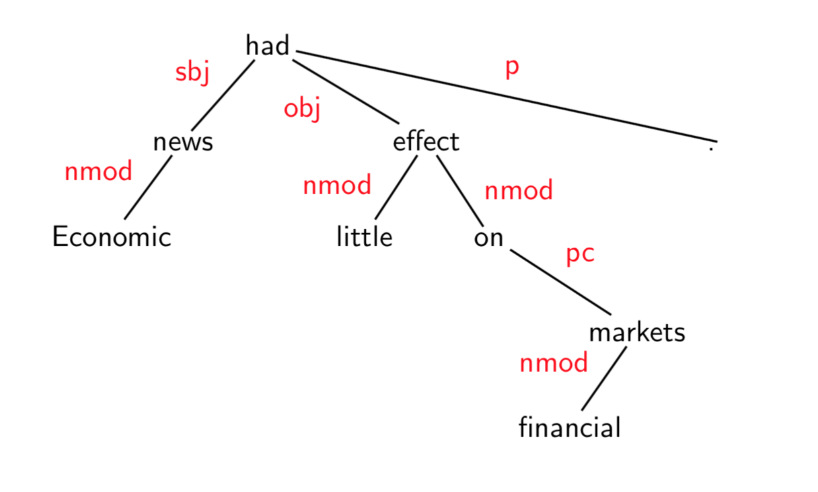
Grammatical Relations

the Universal Dependencies (UD) project (de Marneffe et al., 2021)

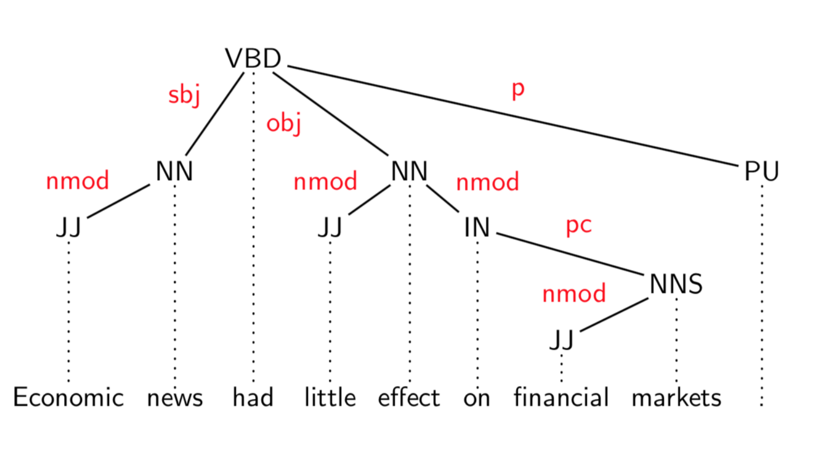
- 37 types

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
OBJ	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*.

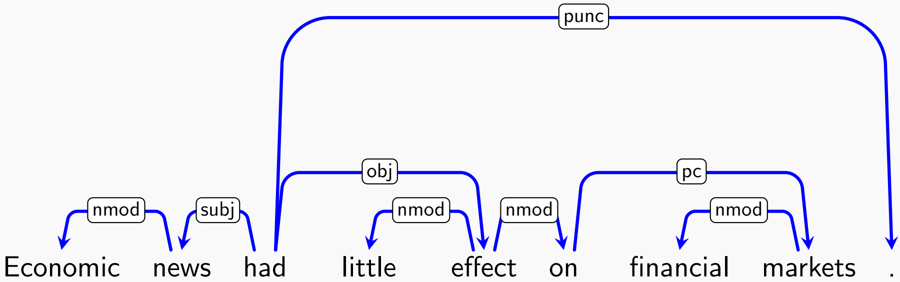
Notations


had
sbj
obj
p
news
effect
.
nmod
nmod
nmod
Economic
little
on
pc
markets
nmod
financial

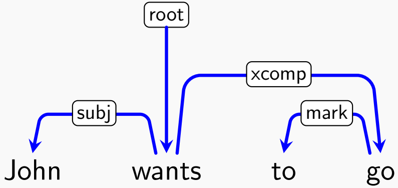
Notations


VBD
sbj
obj
p
NN
NN
PU
nmod
nmod
nmod
JJ
JJ
IN
pc
NNS
nmod
JJ
Economic news had little effect on financial markets .

Notations

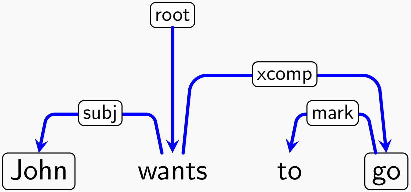

punc
obj
pc
nmod
subj
nmod
nmod
nmod
Economic news had little effect on financial markets .

Are They Always Trees?



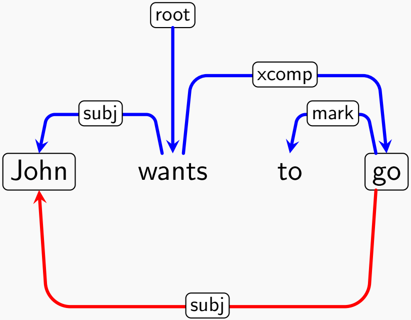
- `xcomp`: open clausal complement
- `mark`: marker (semantically empty)

Are They Always Trees?



- `xcomp`: open clausal complement
- `mark`: marker (semantically empty)

Are They Always Trees?



- `xcomp`: open clausal complement
- `mark`: marker (semantically empty)
- But *John* is also the agent of *go*. And this kind of relation is systematic.
  - *He wants to sleep in class*.
  - *He promises her not to sleep in class*.

Outline

Dependency Graph

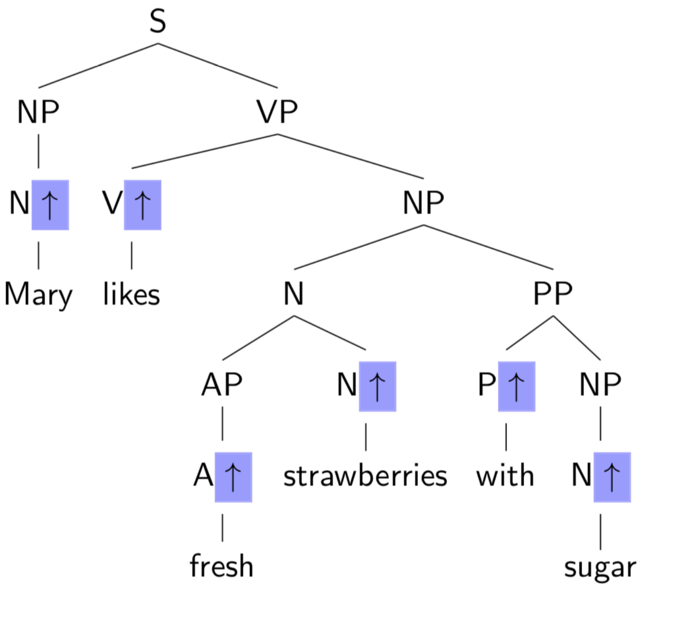
- A dependency structure can be defined as a directed graph, consisting of
  - a set of nodes
  - a set of arcs (edges)
  - a linear precedence order on the node set
- Labeled graphs:
  - Nodes are labeled with word forms (and annotation)
  - Arcs are labeled with dependency types.

Dependency Graph

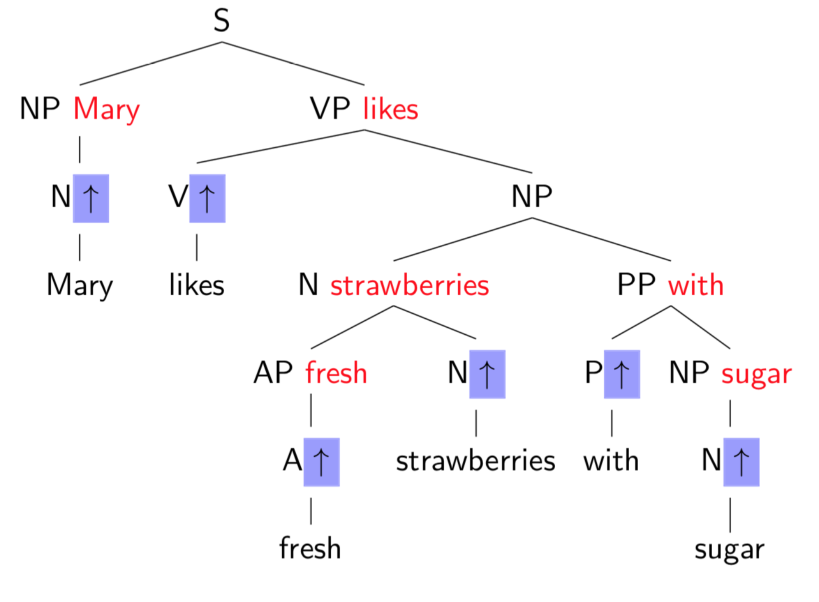
We hope the dependency graph G:

- G is (weakly) connected:
  - for every node i there is a node j such that $i \to j$ or $j \to i$.
- G is acyclic:
  - if $i \to j$ then not $j \to^* i$.
- G obeys the single-head constraint:
  - if $i \to j$,then not $k \to j$,for any $k \neq i$.
- G is projective:
  - if $i \to j$ then $i \to^* k$,for any k such that $i < k < j$ or $j < k < i$.

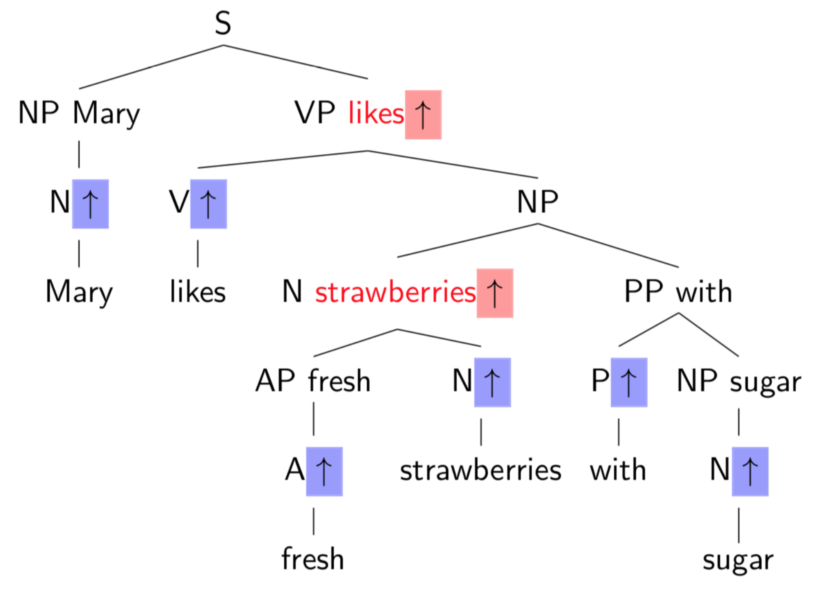
Projectivity



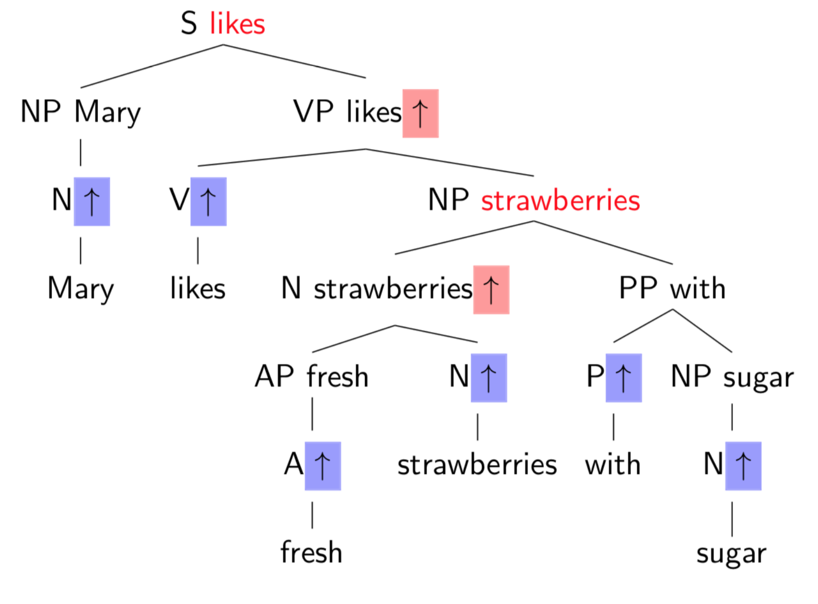
Projectivity



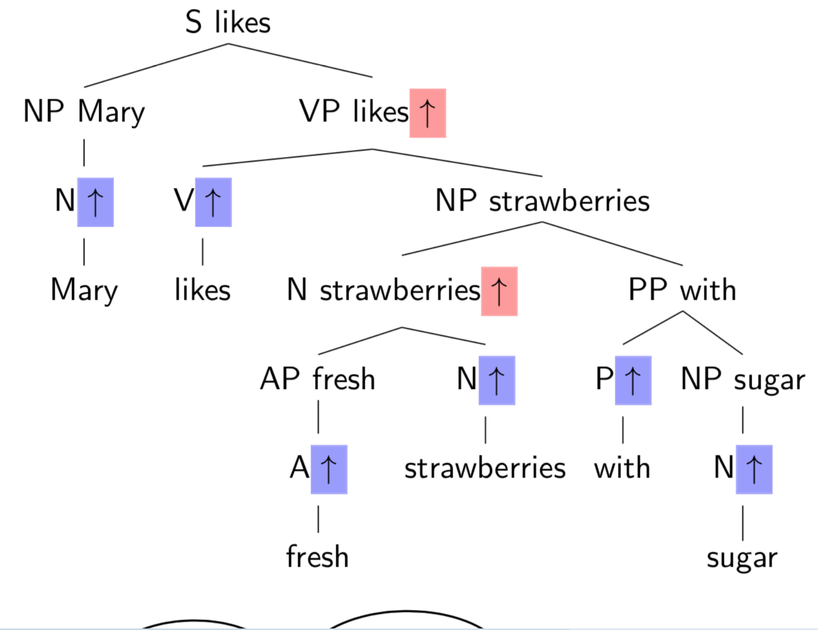
Projectivity



Projectivity



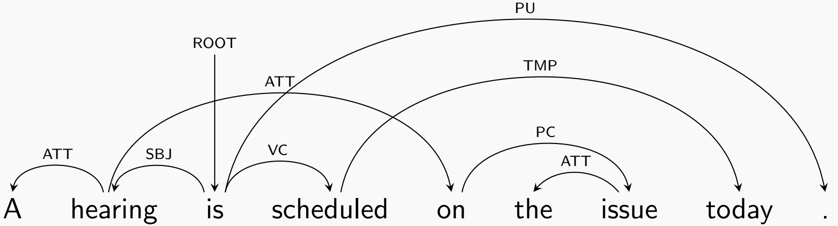
Projectivity



Projectivity

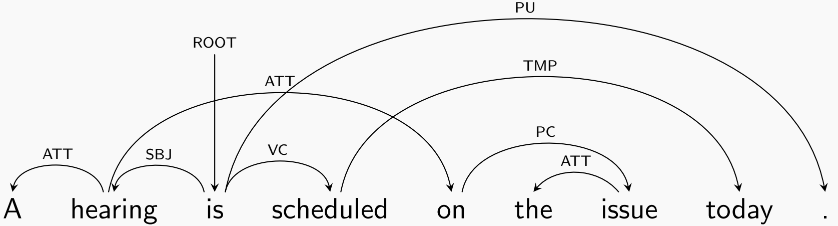
A dependency tree is projective: If $w_i \rightarrow w_j$, then $w_i \rightarrow ... \rightarrow w_k$, for any k such that w_k stands in between w_i and w_j.

- The is a non-projective dependency tree:



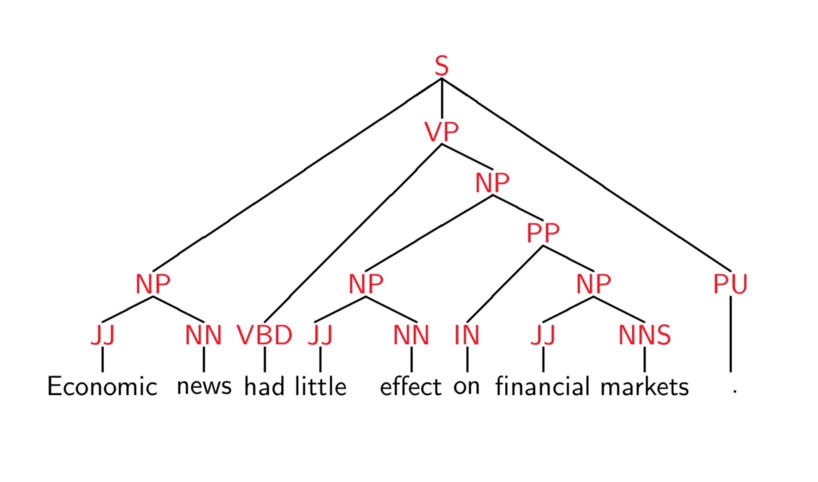
Projectivity

- Most theoretical frameworks do not assume projectivity.
- Non-projective structures are needed to account for
  - long-distance dependencies,
  - free word order.



Phrase Structures

Dealing with constituents



Dependency Structures v.s. Phrase Structures

Dependency structures explicitly represent

- head-dependent relations (directed arcs),
- functional categories (arc labels),
- possibly some structural categories (parts-of-speech)

Phrase structures explicitly represent

- phrases (nonterminal nodes),
- structural categories (nonterminal labels),
- possibly some functional categories (grammatical functions).

Dependency Structures v.s. Phrase Structures

Dependency structures explicitly represent

- head-dependent relations (directed arcs),
- functional categories (arc labels),
- possibly some structural categories (parts-of-speech)

Dependency structures are

- intuitively closer to meaning
- more neutral to word order variations.

Phrase structures explicitly represent

- phrases (nonterminal nodes),
- structural categories (nonterminal labels),
- possibly some functional categories (grammatical functions).

Dependency Structures v.s. Phrase Structures

Dependency structures explicitly represent

- head-dependent relations (directed arcs),
- functional categories (arc labels),
- possibly some structural categories (parts-of-speech)

Dependency structures are

- intuitively closer to meaning
- more neutral to word order variations.

Phrase structures explicitly represent

- phrases (nonterminal nodes),
- structural categories (nonterminal labels),
- possibly some functional categories (grammatical functions).

A proxy for the semantic relationships between predicates and arguments

Dependency Treebanks

- Many previous dependency structure annotations are automatically transformed from phrase structures, e.g., Penn Treebanks
- Still, the Universal Dependencies (UD) project (de Marneffe et al., 2021)
  - 200 treebanks in more than 100 languages!

Outline

Now, Parsing Algorithms

How to obtain such structures?

Incrementality in Human Language Comprehension

Self-paced Reading

press a button for each word

Incrementality in Human Language Comprehension

Self-paced Reading

press a button for each word

convinced

Incrementality in Human Language Comprehension

Self-paced Reading

press a button for each word

her

Incrementality in Human Language Comprehension

Self-paced Reading

press a button for each word

children

Incrementality in Human Language Comprehension

Self-paced Reading

press a button for each word

are

Incrementality in Human Language Comprehension

Self-paced Reading

press a button for each word

noisy.

Incrementality in Human Language Comprehension

Self-paced Reading

press a button for each word

I convinced her children are noisy.

Incrementality in Human Language Comprehension

Self-paced Reading

press a button for each word

I convinced her children are noisy.

Garden-path Sentences:

Incrementality in Human Language Comprehension

Self-paced Reading

press a button for each word

I convinced her children are noisy.

Garden-path Sentences:

A garden-path sentence is a grammatically correct sentence that starts in such a way that a reader's most likely interpretation will be incorrect; the reader is lured into a parse that turns out to be a dead end or yields a clearly unintended meaning.

- *The old man the boats.*
- *The man who hunts ducks out on weekends.*

Incrementality in Human Language Comprehension

Self-paced Reading

press a button for each word

I convinced her children are noisy.

`Garden-path Sentences`:

- *The old man the boats.*
- *The man who hunts ducks out on weekends.*

Linguistic performance

- Left-to-right, word-by-word
- Partially parsed results (history) constrain parsing of subsequent words
- Usually, perform greedy search to get a good parse.

Linguistic Structure Predictions

As a structured prediction problem

- word: single classification
- word sequence: a linear chain of classifications
- trees, graphs, ...: searching a structure with many classifications

Linguistic Structure Predictions

As a structured prediction problem

- word: single classification
- word sequence: a linear chain of classifications
- trees, graphs, ...: searching a structure with many classifications

Two views for structured prediction

- discrete optimization: define a scoring function and seek the structure with the highest score
- incremental search: the state of the search is the partial structure built so far; each action incrementally extends the structure

Linguistic Structure Predictions

As a structured prediction problem

- word: single classification
- word sequence: a linear chain of classifications
- trees, graphs, ...: searching a structure with many classifications

Two views for structured prediction

- discrete optimization: define a scoring function and seek the structure with the highest score
  - the Viterbit Algorithm
  - the CKY Algorithm
- incremental search: the state of the search is the partial structure built so far; each action incrementally extends the structure
  - often, greedy search, with a classifier deciding what action to take in every state
  - sometimes, improved with beam search

Linguistic Structure Predictions

As a structured prediction problem

- Search space: Is this analysis possible?
- Measurement: Is this analysis good?
- Decoding: find the analysis that obtains the highest score
- Parameter estimation: find good parameters

$$y^*(x;\theta) = \arg\max_{y \in \mathcal{Y}(x)} Score(x, y)$$

Linguistic Structure Predictions

As a structured prediction problem

- Search space: Is this analysis possible?
- Measurement: Is this analysis good?
- Decoding: find the analysis that obtains the highest score
- Parameter estimation: find good parameters

$$y^*(x;\theta) = \arg\max_{y \in \mathcal{Y}(x)} Score(x, y)$$

generate a structure step by step

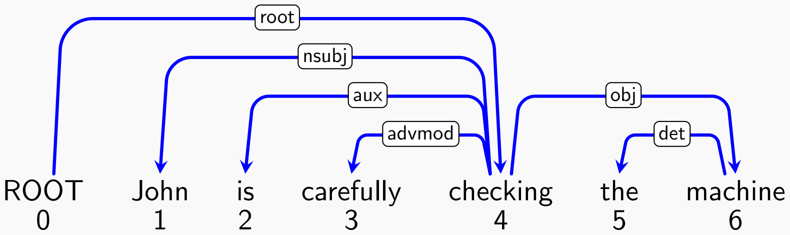
Outline

The Task

from

John is carefully checking the machine

to



Transition-Based Dependency Parsing

A transition system for parsing is a quadruple $S = (C, T, c_s, C_t)$, where

- C is a set of configurations, each of which represents a parser state.
- T is a set of transitions, each of which represents a parsing action,
- c_s initializes S by mapping a sentence x to a particular configuration,
- $C_t \subseteq C$ is a set of terminal configurations.

Deterministic parsing

$Parse(x = (w_0, w_1, ..., w_n))$

    $c \leftarrow c_s(x)$
    while $c \notin C_t$
        $c = Act(c, GetTransition(c))$
    return G_c

Oracle

- An oracle for a transition system $S = (C, T, c_s, C_t)$ is a function $o : C \rightarrow T$.
- Given S and o, deterministic parsing is simple:

$Parse(x = (w_0, w_1, ..., w_n))$

1 $c \leftarrow c_s(x)$

2 while $c \notin C_t$

3 $c = [o(c)](c)$

4 return G_c

Oracles can be approximated by a classifier

$$o(c) = \arg\max_t ScoreTransition(c, t; \theta)$$

You can use whatever classifiers, perceptron, loglinear model, SVM, Neural Networks, etc.

Transition-Based Parsing

Deterministic parsing

$Parse(x = (w_0, w_1, ..., w_n))$

```
c ← c_s(x)
while c ∉ C_t
    c = Act(c, GetTransition(c))
return G_c
```

Basic idea

- Define a transition system (state machine) for mapping a sentence to its parse.
- Learning: Induce a model for predicting the next action (state transition), given the current state.
- Parsing: Construct the optimal transition sequence, given the induced model.

Stack-based Transition Systems

A stack-based configuration for a sentence $x = w_0, w_1, ..., w_n$ is a quadruple $c = (x, \sigma, \beta, A)$, where

- σ is a stack of tokens $i \leq m$ (for some $m \leq n$),
- β is a buffer of tokens $j > m$,
- A is a set of dependency arcs such that $G = (0, 1, ..., n, A)$ is a dependency graph for x.

A stack-based transition system is a quadruple $S = (C, T, c_s, C_t)$, where

- C is the set of all stack-based configurations,
- $c_s(x = w_0, w_1, ...w_n) = ([0], [1, ..., n], \varnothing)$,
- T is a set of transitions, each of which is a function $t : C \rightarrow C$,
- $C_t = c \in C | c = (\sigma, [], A)$.

Outline

Arc-standard algorithm

Transitions

- Shift

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

- Left-Arc$_k$

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i, k)\})$$

- Right-Arc$_k$

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, i|\beta, A \cup \{(i, j, k)\})$$

Notation:

- $\sigma|i =$ stack with top i
- $i|\beta =$ buffer with next token i

Arc-standard algorithm

Transitions

- Shift

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

- Left-Arc$_k$

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i, k)\})$$

- Right-Arc$_k$

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, i|\beta, A \cup \{(i, j, k)\})$$

Notation:

- $\sigma|i =$ stack with top i
- $i|\beta =$ buffer with next token i

configurations are structured states

Example: Arc-standard algorithm

ROOT	John	is	carefully	checking	the	machine
0	1	2	3	4	5	6

Stack

[ROOT]

Buffer/Queue

[John, is, carefully, checking, the, machine]

Example: Arc-standard algorithm

ROOT	John	is	carefully	checking	the	machine
0	1	2	3	4	5	6

Shift

Stack

[ROOT]

Buffer/Queue

[John, is, carefully, checking, the, machine]

Example: Arc-standard algorithm

ROOT	John	is	carefully	checking	the	machine
0	1	2	3	4	5	6

Stack

[ROOT, John]

Buffer/Queue

[is, carefully, checking, the, machine]

Example: Arc-standard algorithm

ROOT	John	is	carefully	checking	the	machine
0	1	2	3	4	5	6

Shift

Stack

[ROOT, John]

Buffer/Queue

[is, carefully, checking, the, machine]

Example: Arc-standard algorithm

ROOT	John	is	carefully	checking	the	machine
0	1	2	3	4	5	6

Stack

[ROOT, John, is]

Buffer/Queue

[carefully, checking, the, machine]

Example: Arc-standard algorithm

ROOT	John	is	carefully	checking	the	machine
0	1	2	3	4	5	6

Shift

Stack

[ROOT, John, is]

Buffer/Queue

[carefully, checking, the, machine]

Example: Arc-standard algorithm

ROOT	John	is	carefully	checking	the	machine
0	1	2	3	4	5	6

Stack

[ROOT, John, is, carefully]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm

ROOT	John	is	carefully	checking	the	machine
0	1	2	3	4	5	6

Left-Arc$_{advmod}$

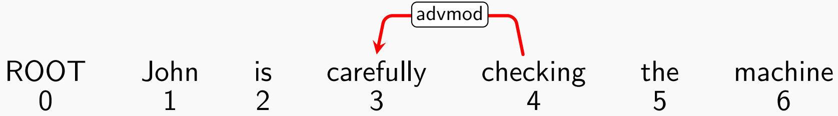
Stack

[ROOT, John, is, carefully]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm



Stack

[ROOT, John, is]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm

advmod

ROOT	John	is	carefully	checking	the	machine
0	1	2	3	4	5	6

Left-Arc$_{aux}$

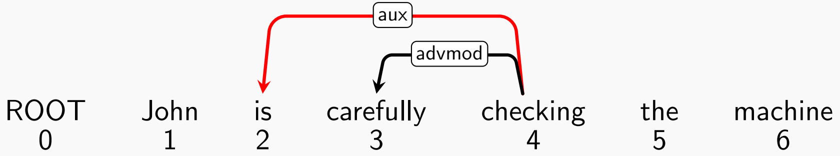
Stack

[ROOT, John, is]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm



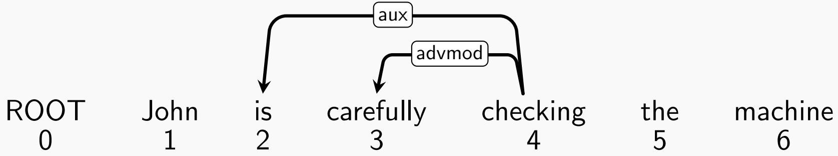
Stack

[ROOT, John]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm



Left-Arc$_{nsubj}$

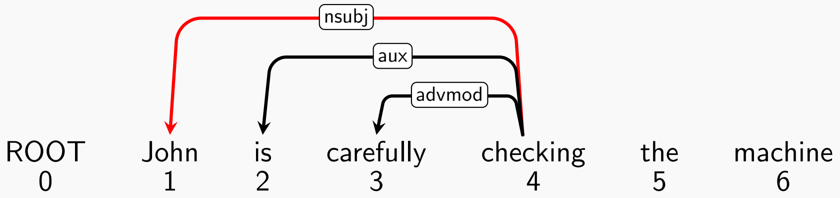
Stack

[ROOT, John]

Buffer/Queue

[checking, the, machine]

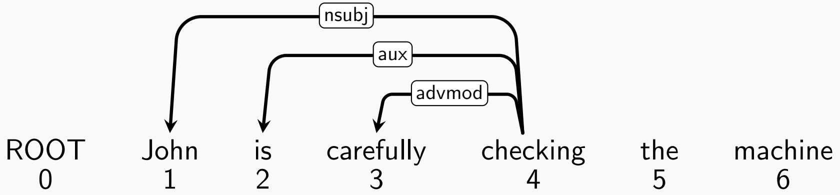
Example: Arc-standard algorithm



Stack
[ROOT]

Buffer/Queue
[checking, the, machine]

Example: Arc-standard algorithm



Shift

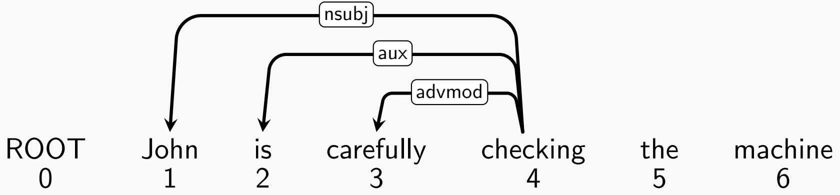
Stack

[ROOT]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm



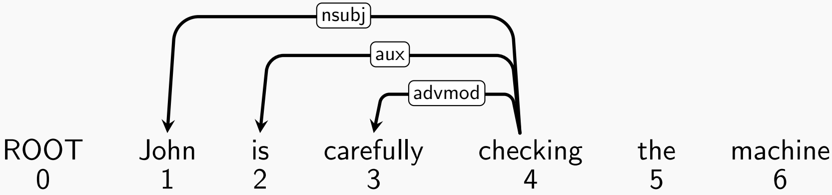
Stack

[ROOT, checking]

Buffer/Queue

[the, machine]

Example: Arc-standard algorithm



Shift

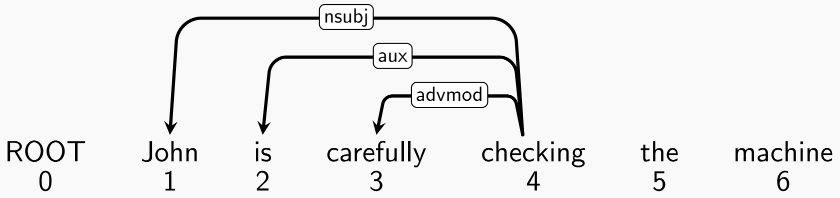
Stack

[ROOT, checking]

Buffer/Queue

[the, machine]

Example: Arc-standard algorithm



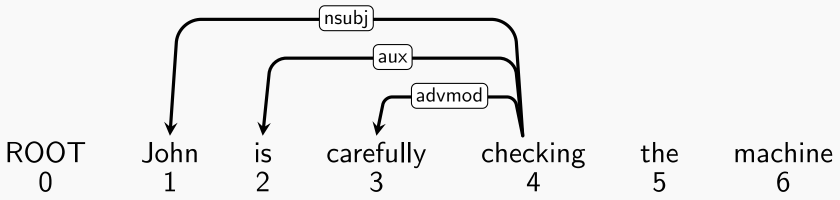
Stack

[ROOT, checking, the]

Buffer/Queue

[machine]

Example: Arc-standard algorithm



Left-Arc$_{det}$

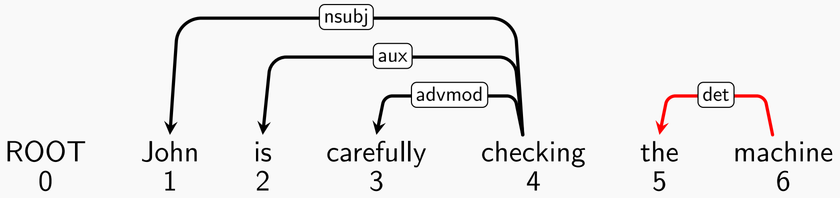
Stack

[ROOT, checking, the]

Buffer/Queue

[machine]

Example: Arc-standard algorithm



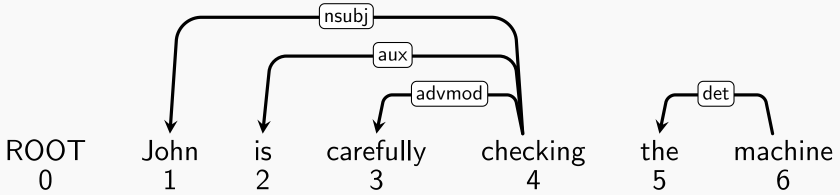
Stack

[ROOT, checking]

Buffer/Queue

[machine]

Example: Arc-standard algorithm



Right-Arc$_{obj}$

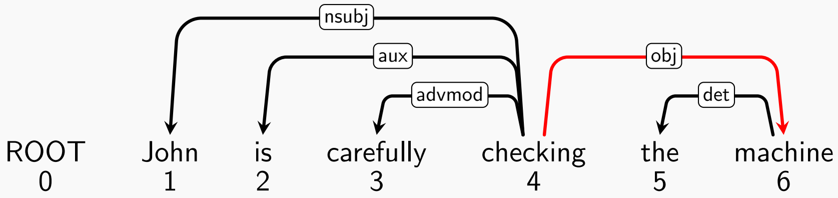
Stack

[ROOT, checking]

Buffer/Queue

[machine]

Example: Arc-standard algorithm



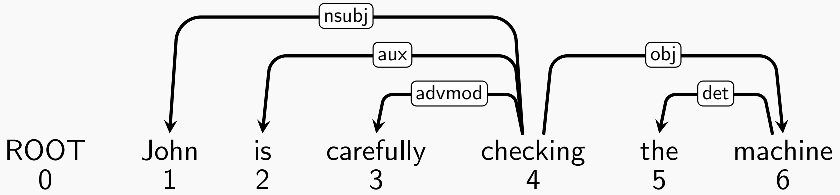
Stack

[ROOT]

Buffer/Queue

[checking]

Example: Arc-standard algorithm



Right-Arc$_{obj}$

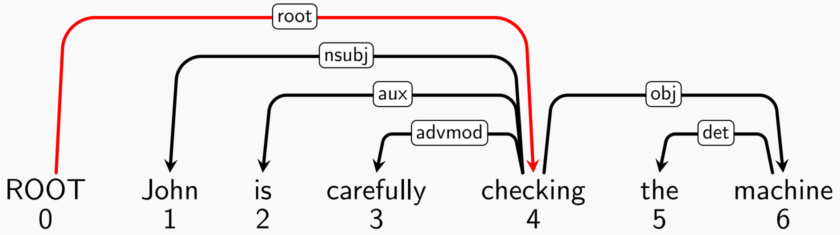
Stack

[ROOT]

Buffer/Queue

[checking]

Example: Arc-standard algorithm



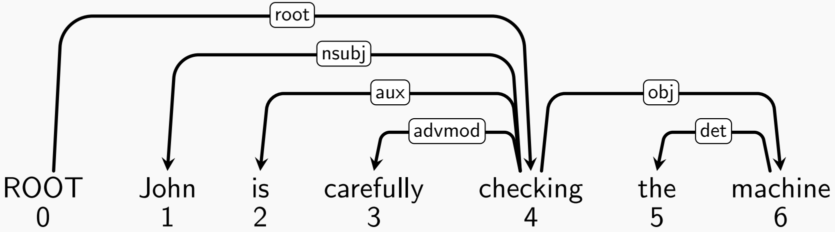
Stack

[]

Buffer/Queue

[ROOT]

Example: Arc-standard algorithm



Shift

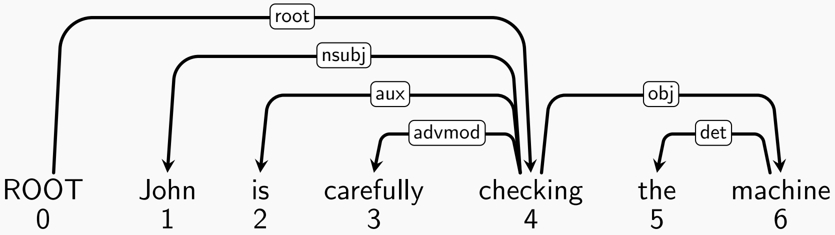
Stack

[]

Buffer/Queue

[ROOT]

Example: Arc-standard algorithm



Stack

[ROOT]

Buffer/Queue

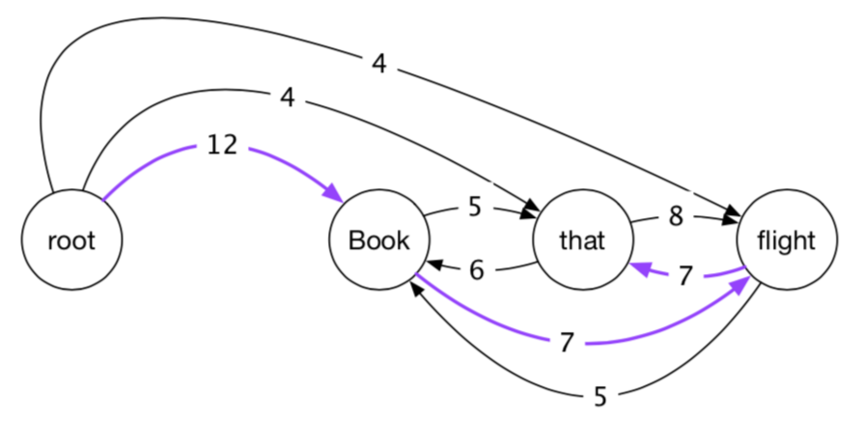
[]

Transition-based dependency parsing

- History-based models, e.g. transition-based parsers, can be very fast.
- Greedy algorithm can go wrong, but usually reasonable accuracy (Note that humans process language incrementally and (mostly) deterministically.)
- No notion of grammaticality (so robust to typos).
- Decisions sensitive to case, agreement etc via features

Graph-Based Dependency Parsing

Parsing via finding the maximum spanning tree

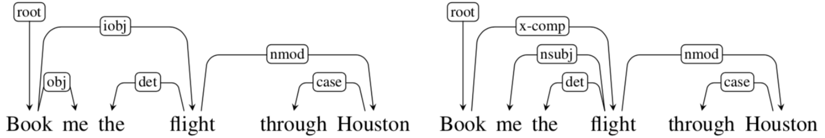


Outline

Evaluation

Evaluate a whole sentence? or, the prediction pieces?

- **Exact match**: how many sentences are parsed correctly ?
- **or**, the percentage of words in an input that are assigned the correct head with the correct relation
  - Labeled Attachment Score (LAS)
  - Unlabeled Attachment Score (UAS)



(a) Reference *(b) System*

Reading

- Chapter 19. Dependency Parsing. Speech and Language Processing.
  `https://web.stanford.edu/~jurafsky/slp3/19.pdf`
- Tutorials on dependency parsing
  `http://stp.lingfil.uu.se/~nivre/docs/ACLslides.pdf`
  `http://eacl2014.org/tutorial-dependency-parsing`