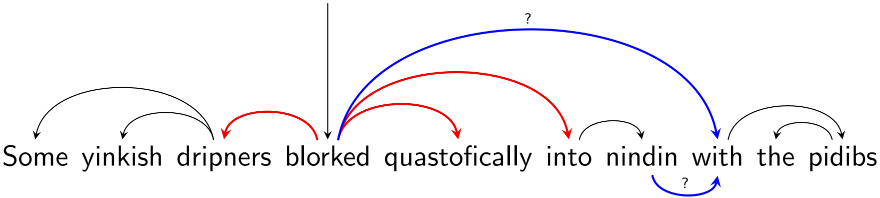
L90: Overview of Natural Language Processing

Lecture 5: Dependency Analysis and History-Based Parsing

Weiwei Sun

Department of Computer Science and Technology
University of Cambridge

Michaelmas 2020/21



words are linked together

Lecture 5: Dependency Analysis and History-Based Parsing

1. Dependency structures
2. Word order across languages
3. History-based models for structured prediction
4. Transition-based dependency parsing
5. Finite state $\rightarrow$ structured states

some slides are from Ann Copestake

Dependency Structures

In the words of Lucien Tesnière

- The sentence is an organized *whole*, the constituent elements of which are *words*.

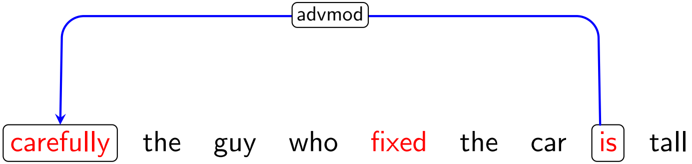
Representation

$\boxed{\text{carefully}}$ the guy who fixed the car $\boxed{\text{is}}$ tall

In the words of 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.

Representation

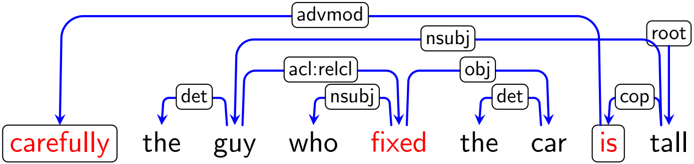


- Relate words to each other via labelled directed arcs (dependencies).

In the words of Lucien Tesnière

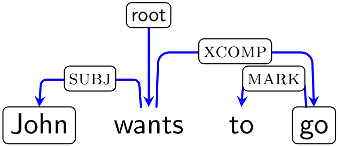
- The sentence is an organized *whole*, the constituent elements of which are *words*.
- Between the word and its neighbors, the mind perceives *connections*, the totality of which forms the structure of the sentence.
- The structural connections establish *dependency* relations between the words. Each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

Representation



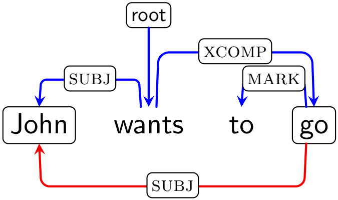
- Relate words to each other via labelled directed arcs (dependencies).
- **connected** **directed** labeled **graph**. lots of variants in NLP: usually trees.

Non-tree dependency structures



- XCOMP: open clausal complement
  cf. *she said that she would like to go*
- MARK: marker (semantically empty)

Non-tree dependency structures



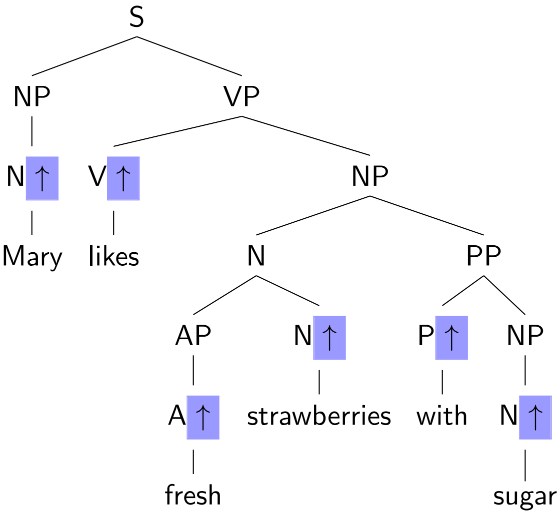
- XCOMP: open clausal complement
  cf. *she said that <u>she would like to go</u>*
- MARK: marker (semantically empty)

- But *John* is also the AGENT of *go*. And this kind of relation is systematic.

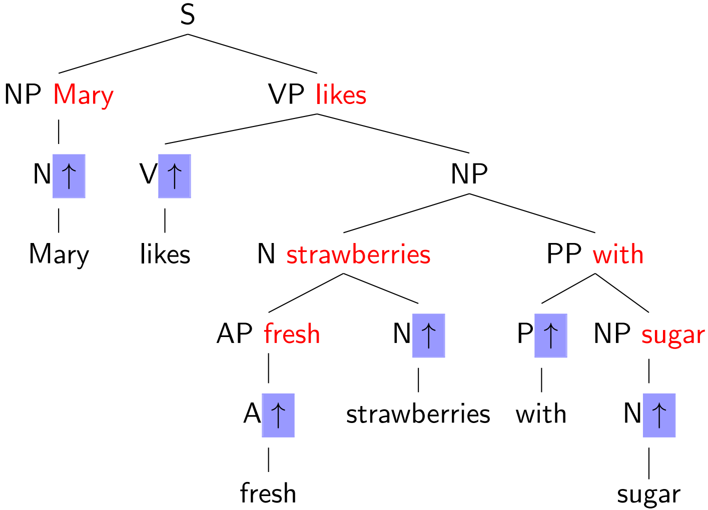
(2) a. *He* **wants** *to sleep in class.*

b. *He* **promises** *her not to sleep in class.*

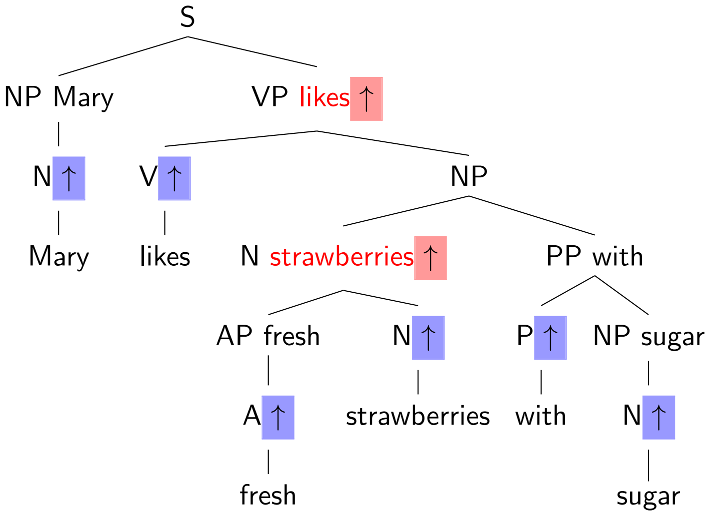
Projectivity (1)



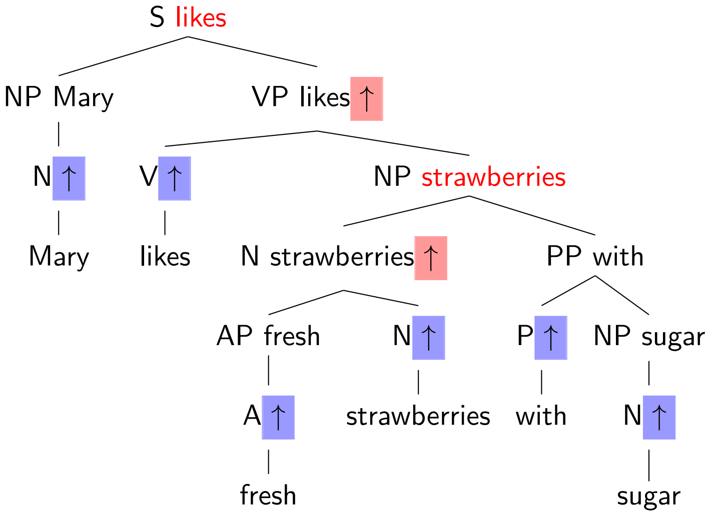
Projectivity (1)



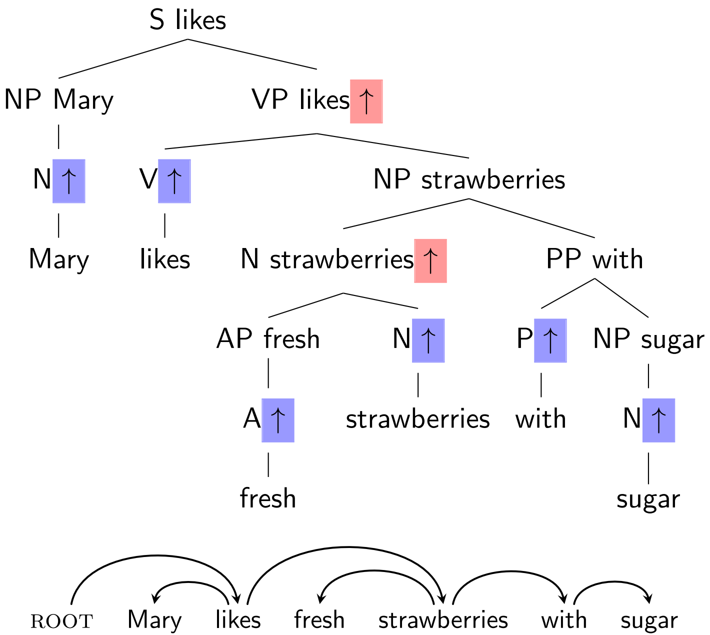
Projectivity (1)



Projectivity (1)



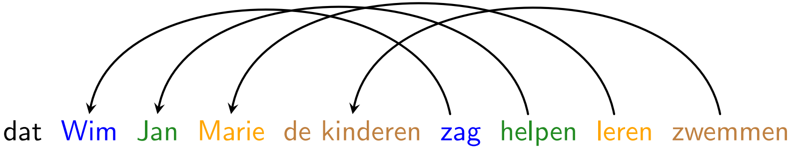
Projectivity (1)



Projectivity (2)

Cross-serial dependencies in Dutch

... dat Wim Jan Marie de kinderen zag helpen leren zwemmen
... that Wim Jan Marie the children saw help teach swim
... that Wim saw Jan help Marie teach the children to swim

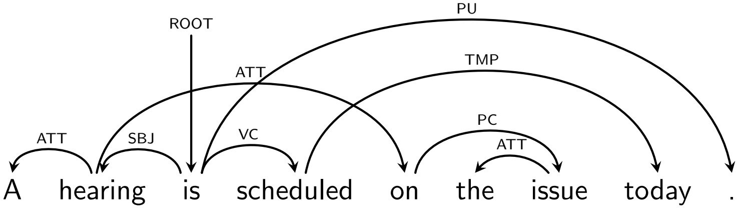


Projectivity (3)

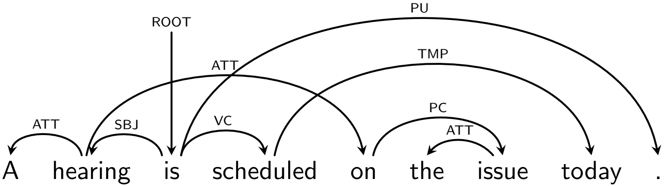
Projectivity

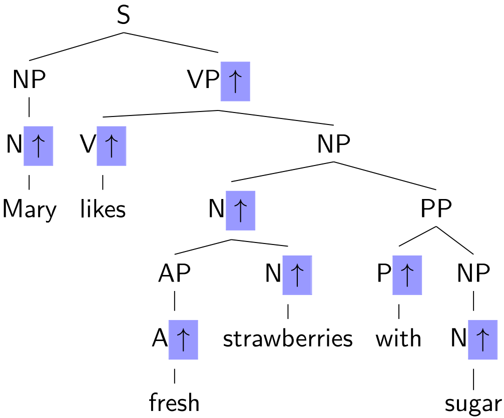
A dependency tree is projective: If $w_i \to w_j$, then $w_i \to \ldots \to w_k$, for any k such that w_k stands in between w_i and w_j.

A non-projective tree



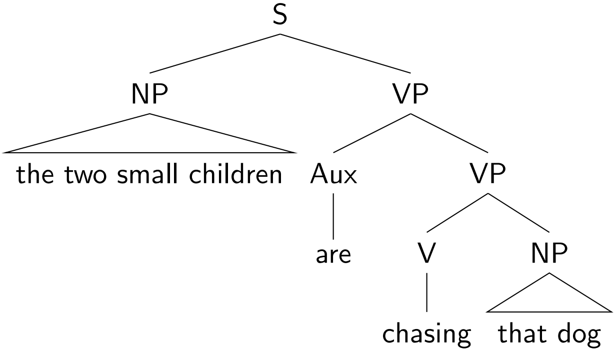
Why so many different lables?


ROOT
PU
TMP
ATT
PC
ATT
SBJ
VC
ATT
A hearing is scheduled on the issue today .


S
NP
VP ↑
N ↑
V ↑
NP
Mary
likes
N ↑
PP
AP
N ↑
P ↑
NP
A ↑
strawberries
with
N ↑
fresh
sugar

Word Order across Languages

Word groups or word shapes (1)



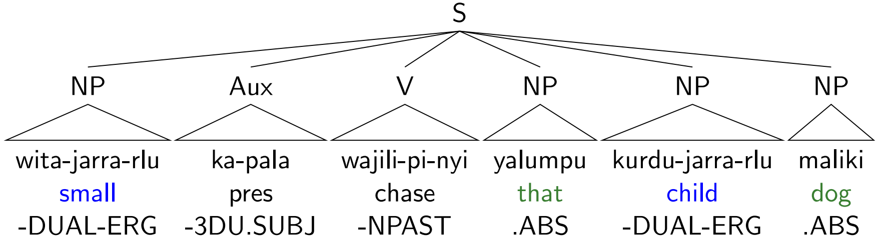
(3) a. *The two small are chasing that children dog.

b. *The two small are dog chasing children that.

c. *Chasing are the two small that dog children.

d. *That are children chasing the two small dog.

Word groups or word shapes (2)



Warlpiri

Every permutation of the words in the sentence is possible, so long as the auxiliary tense marker occurs in the second position.

morphology competes with syntax

Absolutive: Morphologicl case in ergative languages for indicating **subject of intransitive verbs** and **object of transitive verbs**. **Ergative**: Morphologicl case in ergative languages for indicating **agent of the transitive verbs** in the basic voice.

Dependency structures vs 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 are

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

History-Based Models

Incrementality in human language comprehension

Self-paced reading: you press a button for each word

I

Incrementality in human language comprehension

Self-paced reading: you press a button for each word

convinced

Incrementality in human language comprehension

Self-paced reading: you press a button for each word

her

Incrementality in human language comprehension

Self-paced reading: you press a button for each word

children

Incrementality in human language comprehension

Self-paced reading: you press a button for each word

are

Incrementality in human language comprehension

Self-paced reading: you press a button for each word

noisy.

Incrementality in human language comprehension

Self-paced reading: you press a button for each word

I convinced her children are noisy.

at which word, do you stop for a significantly longer time?

Incrementality in human language comprehension

Self-paced reading: you press a button for each word

I convinced her children are noisy.

at which word, do you stop for a significantly longer time?

Garden-path sentences

- The old man the boats.
- The florist sent the flowers was pleased.

Incrementality in human language comprehension

Self-paced reading: you press a button for each word

I convinced her children are noisy.

at which word, do you stop for a significantly longer time?

Garden-path sentences

- The old man the boats.
- The florist sent the flowers was pleased.

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 prediction

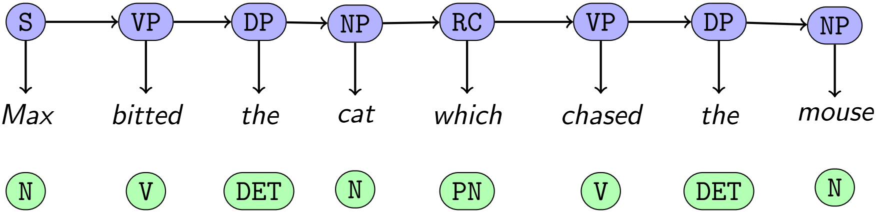
As a structured prediction problem

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

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

generate a structure step by step

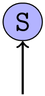
Sequential word tagging



Parsing by tagging

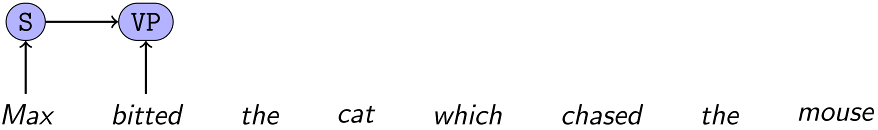
Max *bitted* *the* *cat* *which* *chased* *the* *mouse*

Parsing by tagging

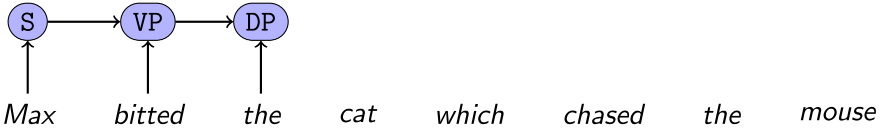


Max bitted the cat which chased the mouse

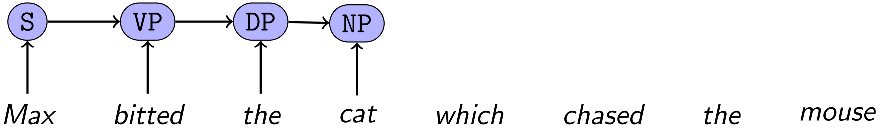
Parsing by tagging



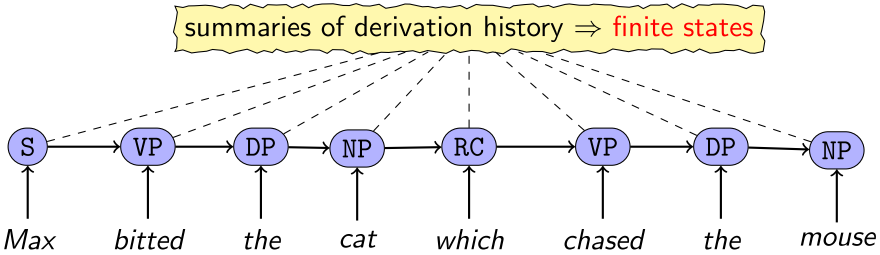
Parsing by tagging



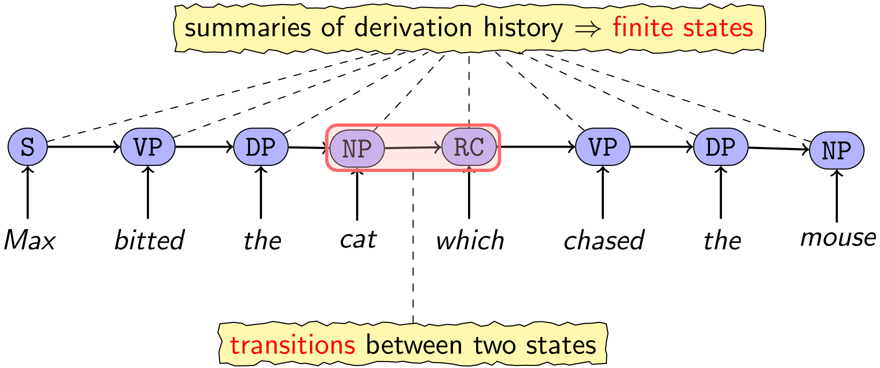
Parsing by tagging



Parsing by tagging



Parsing by tagging



Transition-Based Dependency Parsing

Architecture

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

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

Deterministic parsing

$\textsc{Parse}(x = (w_0, w_1, \ldots, w_n))$

$c \leftarrow c_s(x)$

while $c \notin C_t$

  $c = \textsc{Act}(c, \textsc{GetTransition}(c))$

return G_c

Oracle

- An oracle for a transition system $S = (\mathcal{C}, T, c_s, \mathcal{C}_t)$ is a function $o : \mathcal{C} \to T$.
- Given S and o, deterministic parsing is simple:

$\textsc{Parse}(x = (w_0, w_1, \ldots, w_n))$

$c \leftarrow c_s(x)$

while $c \notin \mathcal{C}_t$

$\qquad c = [o(c)](c)$

return G_c

Oracles can be approximated by a classifier

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

Perceptron, Deep Neural Networks, etc.

Transition-based parsing

$\textsc{Parse}(x = (w_0, w_1, \ldots, w_n))$

$c \leftarrow c_s(x)$

while $c \notin C_t$

$\quad c = \textsc{Act}(c, \textsc{GetTransition}(c))$

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 (viz. 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 $x = w_0, w_1, \ldots, w_n$ is a quadruple $c = (x, \sigma, \beta, \mathcal{A})$, where

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

A stack-based transition system is a quadruple $S = (\mathcal{C}, T, c_s, \mathcal{C}_t)$, where

1. $\mathcal{C}$ is the set of all stack-based configurations,
2. $c_s(x = w_0, w_1, \ldots w_n) = ([0], [1, \ldots, n], \emptyset)$,
3. T is a set of transitions, each of which is a function $t : \mathcal{C} \to \mathcal{C}$,
4. $\mathcal{C}_t = \{c \in \mathcal{C} | c = (\sigma, [\,], \mathcal{A})\}$.

Arc-standard algorithm

Transitions

- Shift:

  $$(\sigma, i|\beta, \mathcal{A}) \Rightarrow (\sigma|i, \beta, \mathcal{A})$$

- Left-Arc$_k$:

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

- Right-Arc$_k$:

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

Notation:

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

Arc-standard algorithm

Transitions

- Shift:

  $$(\sigma, i|\beta, \mathcal{A}) \Rightarrow (\sigma|i, \beta, \mathcal{A})$$

- Left-Arc$_k$:

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

- Right-Arc$_k$:

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

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, ..., 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, ..., 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

$\texttt{Left-Arc}_{advmod}$

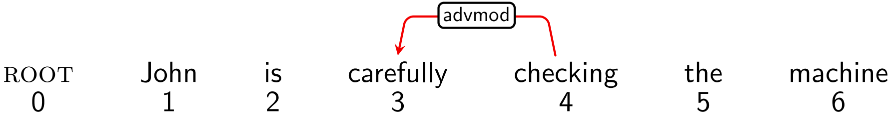
Stack

[ROOT, John, is, carefully]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm



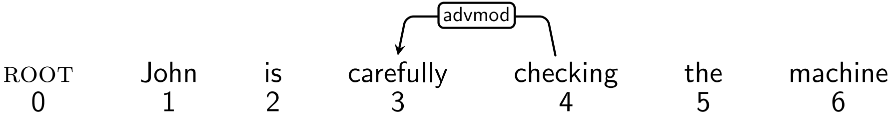
Stack

[ROOT, John, is]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm



$\texttt{Left-Arc}_{aux}$

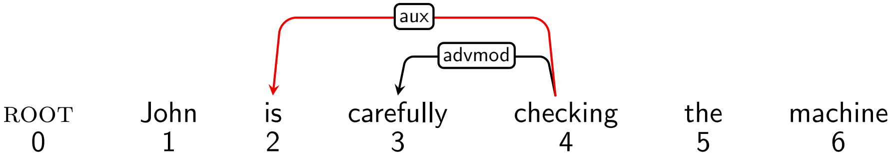
Stack

[ROOT, John, is]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm



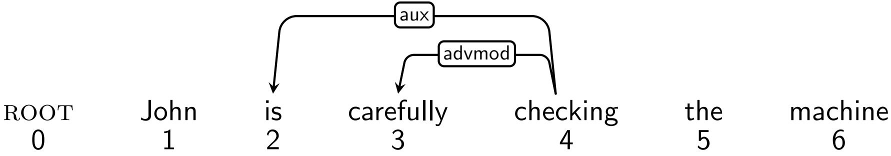
Stack

[ROOT, John]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm



`Left-Arc`$_{nsubj}$

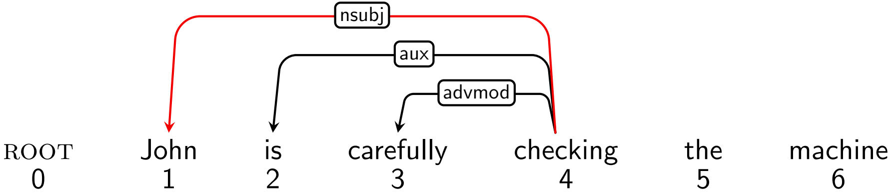
Stack

[ROOT, John]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm



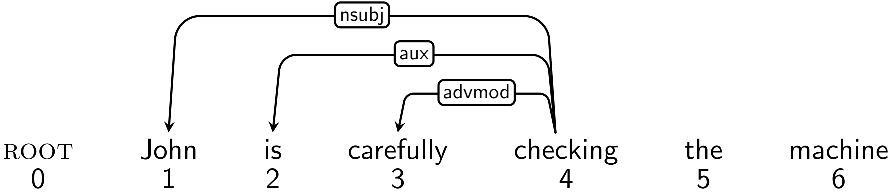
Stack

[ROOT]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm



Shift

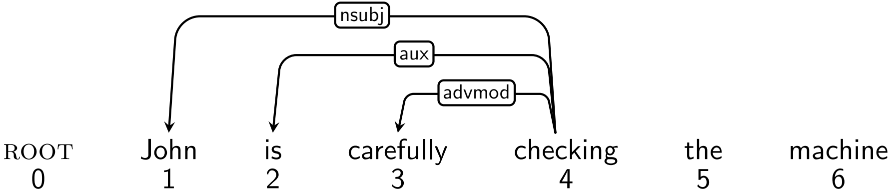
Stack

[ROOT]

Buffer/Queue

[checking, the, machine]

Example: Arc-standard algorithm



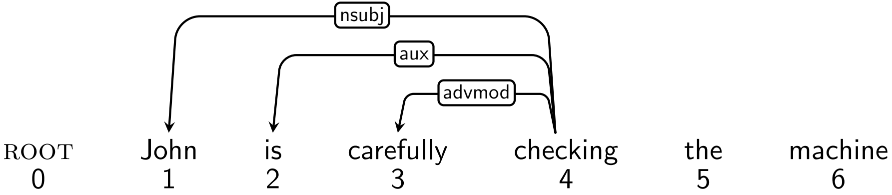
Stack

[ROOT, checking]

Buffer/Queue

[the, machine]

Example: Arc-standard algorithm



Shift

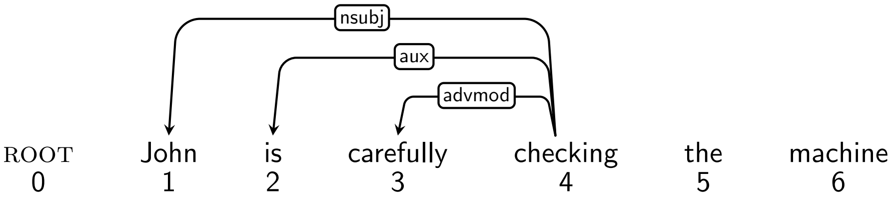
Stack

[ROOT, checking]

Buffer/Queue

[the, machine]

Example: Arc-standard algorithm



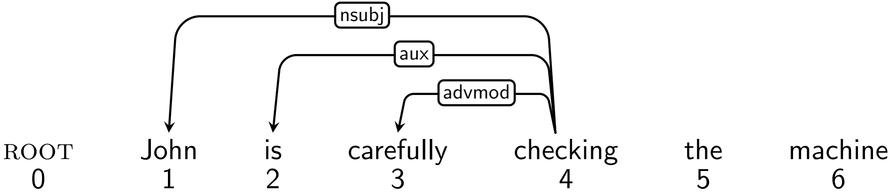
Stack

[ROOT, checking, the]

Buffer/Queue

[machine]

Example: Arc-standard algorithm



`Left-Arc`$_{det}$

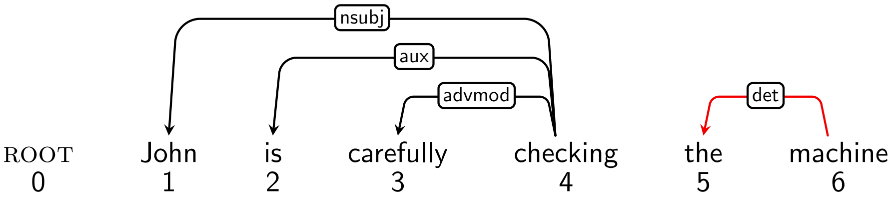
Stack

[ROOT, checking, the]

Buffer/Queue

[machine]

Example: Arc-standard algorithm



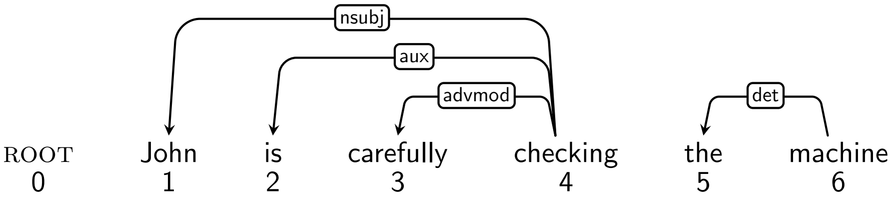
Stack

[ROOT, checking]

Buffer/Queue

[machine]

Example: Arc-standard algorithm



`Right-Arc`$_{obj}$

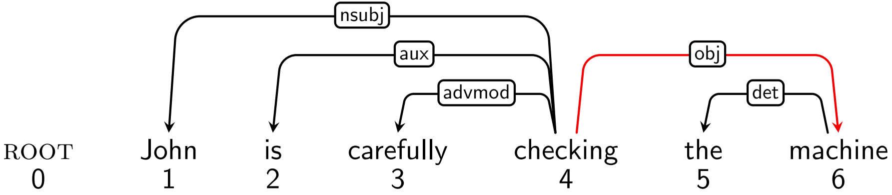
Stack

[ROOT, checking]

Buffer/Queue

[machine]

Example: Arc-standard algorithm



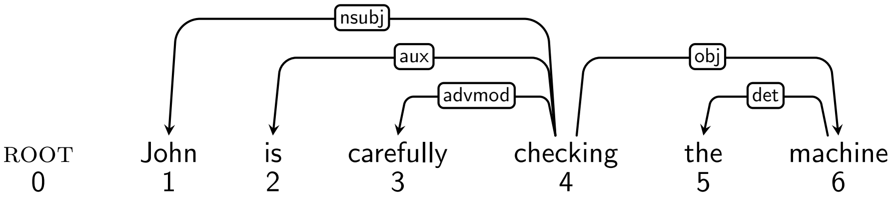
Stack

[ROOT]

Buffer/Queue

[checking]

Example: Arc-standard algorithm



`Right-Arc`$_{obj}$

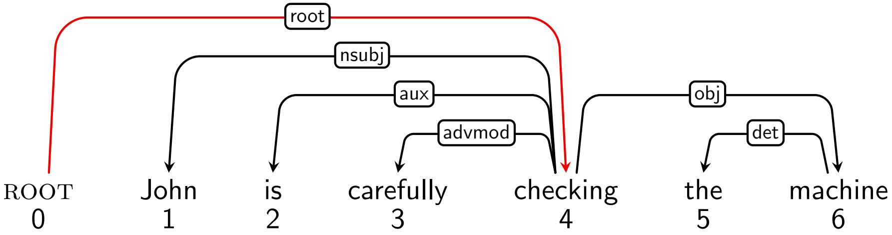
Stack

[ROOT]

Buffer/Queue

[checking]

Example: Arc-standard algorithm



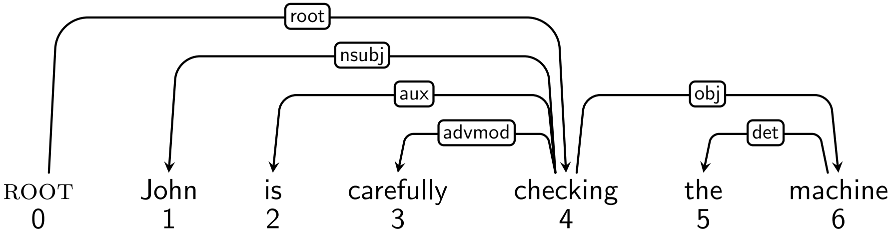
Stack

[]

Buffer/Queue

[ROOT]

Example: Arc-standard algorithm



Shift

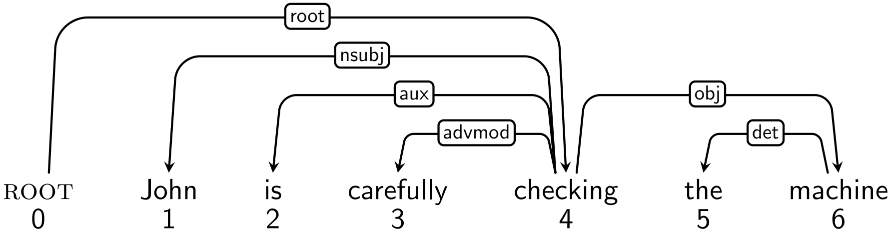
Stack

[]

Buffer/Queue

[ROOT]

Example: Arc-standard algorithm



Stack

[ROOT]

Buffer/Queue

[]

Complexity analysis

A naive idea: $\Theta(n^2)$

```
PARSE(x = (w_1, ..., w_n))
  for j = 1..n
      for k = j - 1..1
          Link(j, k)
```

The operation `Link` chooses between

1. adding the arc (i, j) or (j, i)
2. adding no arc at all.

Arc-standard system: $\Theta(n)/\textsc{shift}+\Theta(n)/\textsc{arc}$ (even better)

- Shift: $(\sigma, i|\beta, \mathcal{A}) \Rightarrow (\sigma|i, \beta, \mathcal{A})$
- Left-Arc$_k$: $(\sigma|i, j|\beta, \mathcal{A}) \Rightarrow (\sigma, j|\beta, \mathcal{A} \cup \{(j, i, k)\})$
- Right-Arc$_k$: $(\sigma|i, j|\beta, \mathcal{A}) \Rightarrow (\sigma, i|\beta, \mathcal{A} \cup \{(i, j, k)\})$

Expressiveness of a transition system

A transition sequence for a sentence $x = w_0, w_1, \ldots, w_n$ in S is a sequence $\mathbf{c}_{0,m} = (c_0, c_1, \ldots, c_m)$ of configurations, such that

1. $c_0 = c_s(x)$,
2. $c_m \in \mathcal{C}_t$,
3. for every i $(1 \leq i \leq m)$, $c_i = t(c_{i-1})$ for some $t \in T$.

Correctness

- S is **sound** for a class of parses $\mathcal{G}$ iff, for every sentence x and every transition sequence $\mathbf{c}_{0,m}$ for x in S, the parse $G_{c_m} \in \mathcal{G}$.
- S is **complete** for a class of parses $\mathcal{G}$ iff, for every sentence x and every parse G_x for x in $\mathcal{G}$, there is a transition sequence $\mathbf{c}_{0,m}$ such that $G_{c_m} = G_x$.

Expressiveness of a transition system

A transition sequence for a sentence $x = w_0, w_1, \ldots, w_n$ in S is a sequence $\mathbf{c}_{0,m} = (c_0, c_1, \ldots, c_m)$ of configurations, such that

1. $c_0 = c_s(x)$,
2. $c_m \in \mathcal{C}_t$,
3. for every i $(1 \leq i \leq m)$, $c_i = t(c_{i-1})$ for some $t \in T$.

Correctness

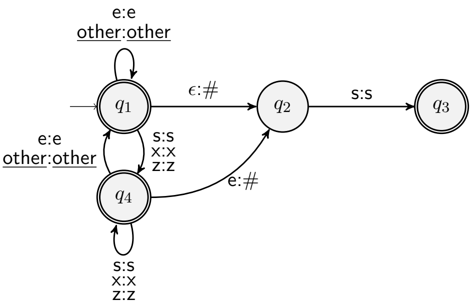
- S is **sound** for a class of parses $\mathcal{G}$ iff, for every sentence x and every transition sequence $\mathbf{c}_{0,m}$ for x in S, the parse $G_{c_m} \in \mathcal{G}$.
- S is **complete** for a class of parses $\mathcal{G}$ iff, for every sentence x and every parse G_x for x in $\mathcal{G}$, there is a transition sequence $\mathbf{c}_{0,m}$ such that $G_{c_m} = G_x$.

can arc-standard system generate a non-projective tree?

Finite States $\rightarrow$ Structured States

Comparing states

Shift
$(\sigma, i|\beta, \mathcal{A}) \Rightarrow (\sigma|i, \beta, \mathcal{A})$
Left-Arc$_k$
$(\sigma|i, j|\beta, \mathcal{A}) \Rightarrow (\sigma, j|\beta, \mathcal{A} \cup \{(j, i, k)\})$
Right-Arc$_k$
$(\sigma|i, j|\beta, \mathcal{A}) \Rightarrow (\sigma, i|\beta, \mathcal{A} \cup \{(i, j, k)\})$



- A symbolic system that can recognize or transform forms.
- An automaton remembers only a finite amount of information.
- Information is represented by its states.
- State changes in response to inputs and may trigger outputs.
- Transition rules define how the state changes in response to inputs.
- Given a sequence of input symbols, a recognition process starts in the start state and follow the transitions in turn. Input is accepted if this process ends up in an accepting state.

Transition-based dependency parsing

- History-based models, e.g. transition-based parsers, can be very fast.
  But GPU . . .
- 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
- *Den Mann beisst der Hund*
  choice between $\textsc{LeftArc}_{\mathsf{subj}}$ and $\textsc{LeftArc}_{\mathsf{obj}}$ conditioned on case of noun as well as position.

human brain inspired artificial intelligence

Universal dependencies

- Ongoing at tempt to define a set of dependencies which will work cross-linguistically
- `http://universaldependencies.org`
- Also 'universal' set of POS tags.
- UD dependency treebanks for over 50 languages (though most small).
- No single set of dependencies is useful cross-linguistically: tension between universality and meaningful dependencies.

Dependency annotation

- Balance between linguistically-motivated scheme, ease of human annotation, parsing efficiency and so on.
- Some vague 'catch all' classes in UD: e.g., MARK.
- Words like English infinitival *to* resist clean classification.
- Many linguistic generalizations can't be captured by dependencies.
- Semantic dependencies (not this time).

Readings

- Ann's lecture notes.
  `https://www.cl.cam.ac.uk/teaching/1920/NLP/materials.html`
- Chapter 14. Dependency Parsing. *Speech and Language Processing*.
  `https://web.stanford.edu/~jurafsky/slp3/15.pdf`
- Tutorials on dependency parsing
  `http://stp.lingfil.uu.se/~nivre/docs/ACLslides.pdf`
  `http://eacl2014.org/tutorial-dependency-parsing`