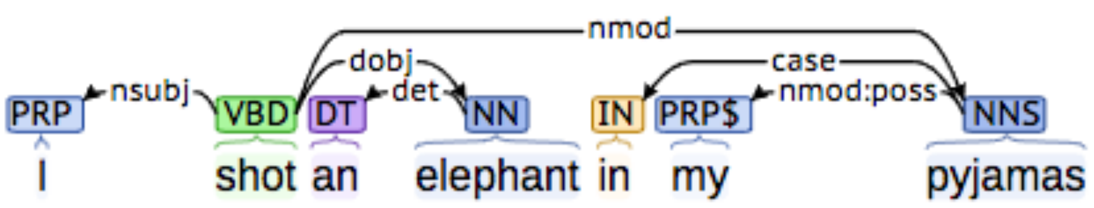
Dependency Grammar & Parsing

COMP90042 Lecture 19

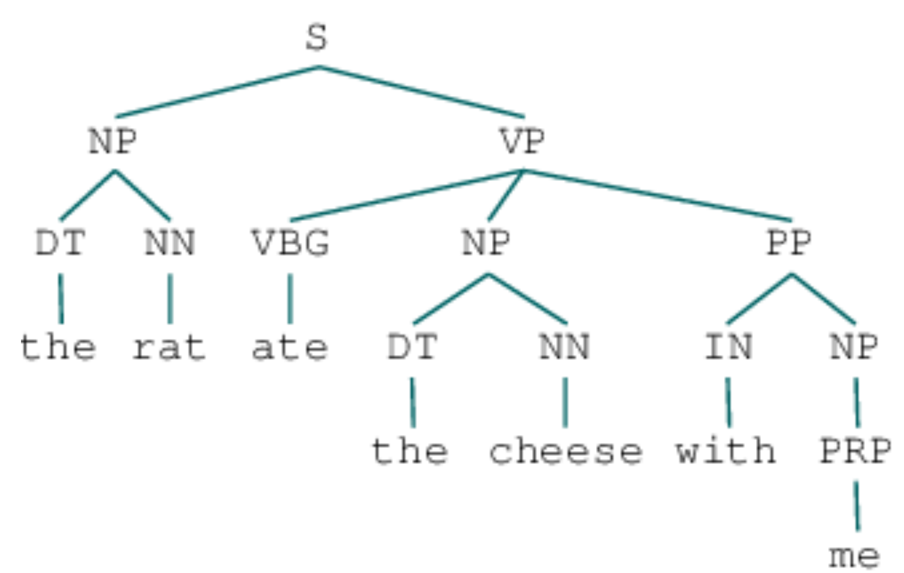


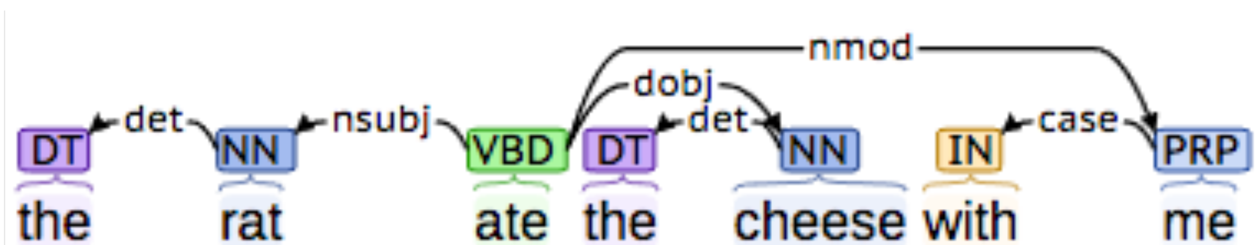
Outline

- Dependency grammars
- Projectivity
- Parsing methods
  - transition-based parsing
  - graph-based

Dependency G vs. Phrase-Structure G

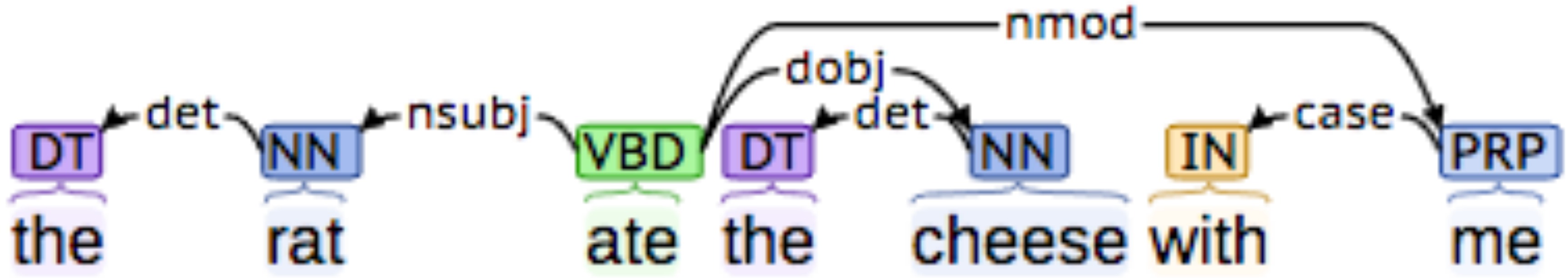
- *phrase-structure grammars* assume a *constituency tree* which identifies the *phrases* in a sentence
  - based on idea that these phrases are interchangable (e.g., swap an NP for another NP) and maintain grammaticality
- *dependency grammar* offers a simpler approach
  - describe binary relations between pairs of words
  - namely, between *heads* and *dependents*
- Building on notion of *head* as seen in phrase-structure parsers…





What is a Dependency?

- Links between a *head* word and its *dependent* words in the sentence: either *syntactic roles* or *modifier relations*



- *Several types of dependency, e.g.,*
  - *argument* of a predicate, e.g., *ate(rat, cheese)*
    - *rat* is the *subject* of verb *ate* (thing doing the eating)
    - *cheese* is the *direct object* of verb ate (thing being eaten)

What is a Dependency II

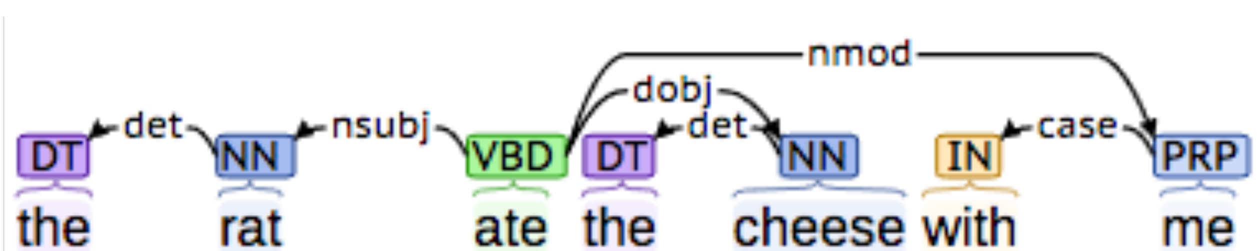
- Other types of dependencies include
  - a *modifier* which is typically optional (aka *adjunct*)
    - [(with) *me*] modifies the act of (the rat) *eating*
  - *specifiers*, e.g., *the rat, the cheese, with me*
    - help to specify the referent (which rat?), the head's relation, etc.
- Head and *type* of relation will affect dependents
  - Case, verb-subject agreement: *I talk to myself*, vs **me talks to I*
  - agreement for number, gender and case

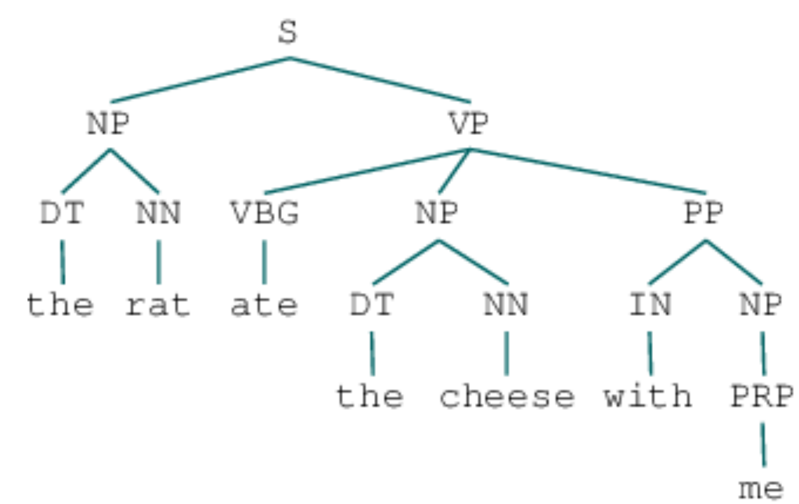
Dependency types

- Edges labelled with the dependency *type, e.g., Stanford types, e.g.,* sample types (key: *head,* **dependent**)
  - NSUBJ (nominal subject) — **Daniel** *speaks* Brazilian Portuguese
  - DOBJ (direct object) — Trevor *presented* a **lecture** in English
  - IOBJ (indirect object) — Morpheus *gave* **Neo** the red pill
  - APPOS (appositive) — *Neo*, the main **character**, swallowed the pill
- See reading for more!

Why dependencies?

- Dependency tree more directly represents the core of the sentence: *who did what to whom?*
  - captured by the links incident on verb nodes, e.g., NSUBJ, DOBJ etc; easier to answer questions like:
    - what was the main thing being expressed in the sentence (*eating* = root)

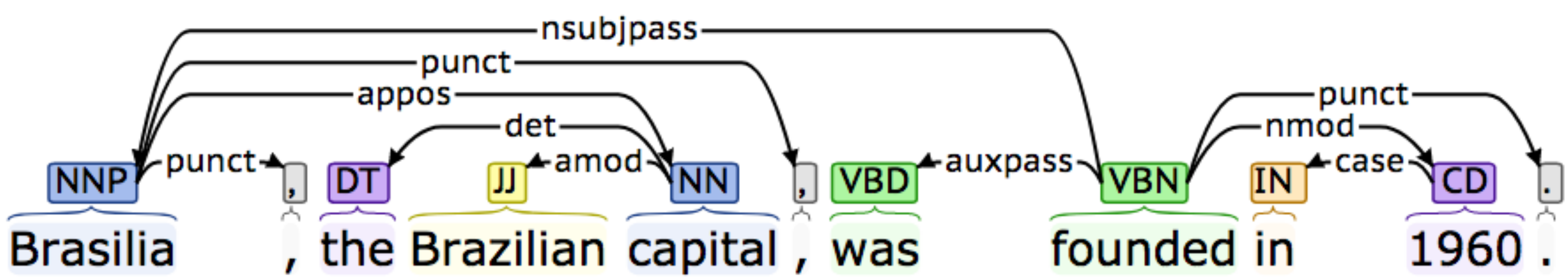




  - more minor details are buried deeper in the tree (e.g., adjectives, determiners etc)

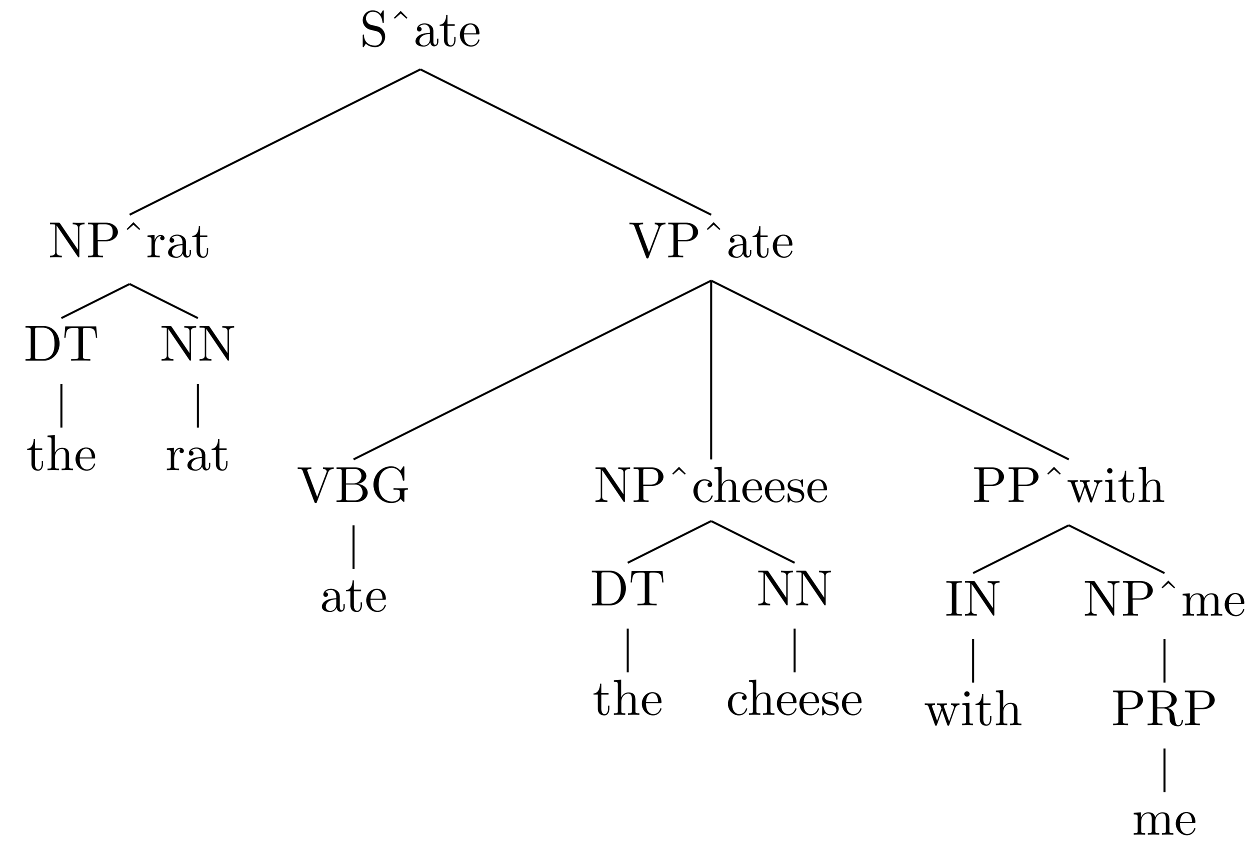
Dependencies in NLP models

- What can we do with dependency trees?
  - use as features for other tasks, e.g., sentiment, relation extraction, QA, various semantic tasks.
- E.g., relation extraction
  - "Brasilia, the Brazilian capital, was founded in 1960."
    → capital(Brazil, Brasilia); founded(Brasilia, 1960)
  - parts of the tree capture relations in succinctly and in a generalisable way



Dependency vs head

- Close similarity with 'head' in phrase-structure grammars
  - the 'head' of an XP is (mostly) an X, i.e., noun in a NP, verb in a VP etc.
  - main dependency edges captured in rewrite rules
    - S^ate -> NP^rat VP^ate captures dependency rat ← ate



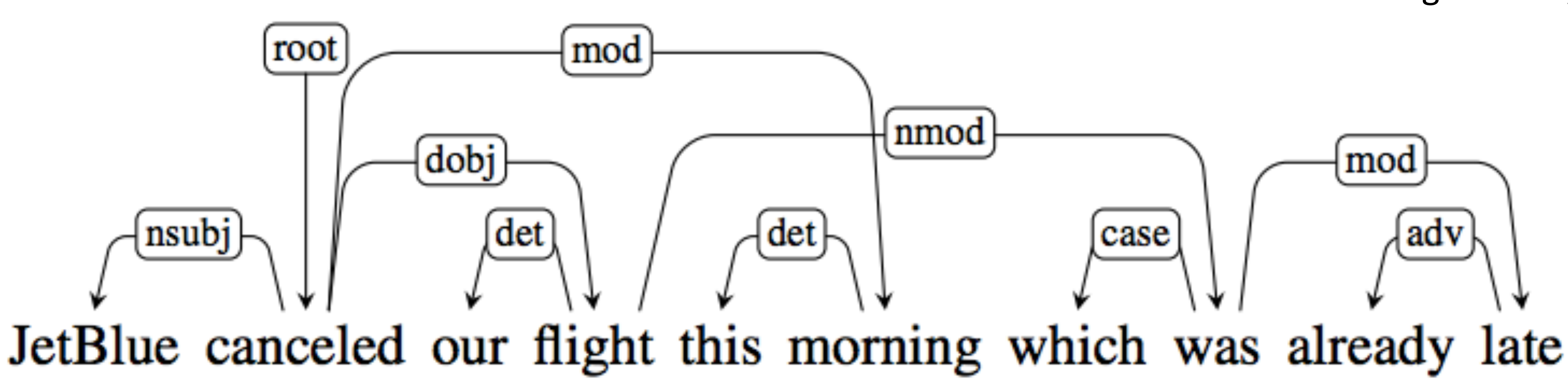
Dependency *tree*

- Dependency edges form a *tree*
  - each node is a *word token*
  - one node is chosen as the *root*
  - directed edges link heads and their dependents
- Cf. phrase-structure grammars
  - forms a hierarchical tree
  - word tokens are the *leaves*
  - internal nodes are ‘constituent phrases’ e.g., NP, VP etc
- Both use part-of-speech

Projectivity

- A tree is ***projective*** if, for all arcs from head to dependent
  - there is a path from the head to every word that lies between the head and the dependent
  - I.e., the tree can be drawn on a plane without any arcs crossing
- Most sentences are projective, however exceptions exist (fairly common in other languages)

Figure JM3, Ch 13



Dependency grammar

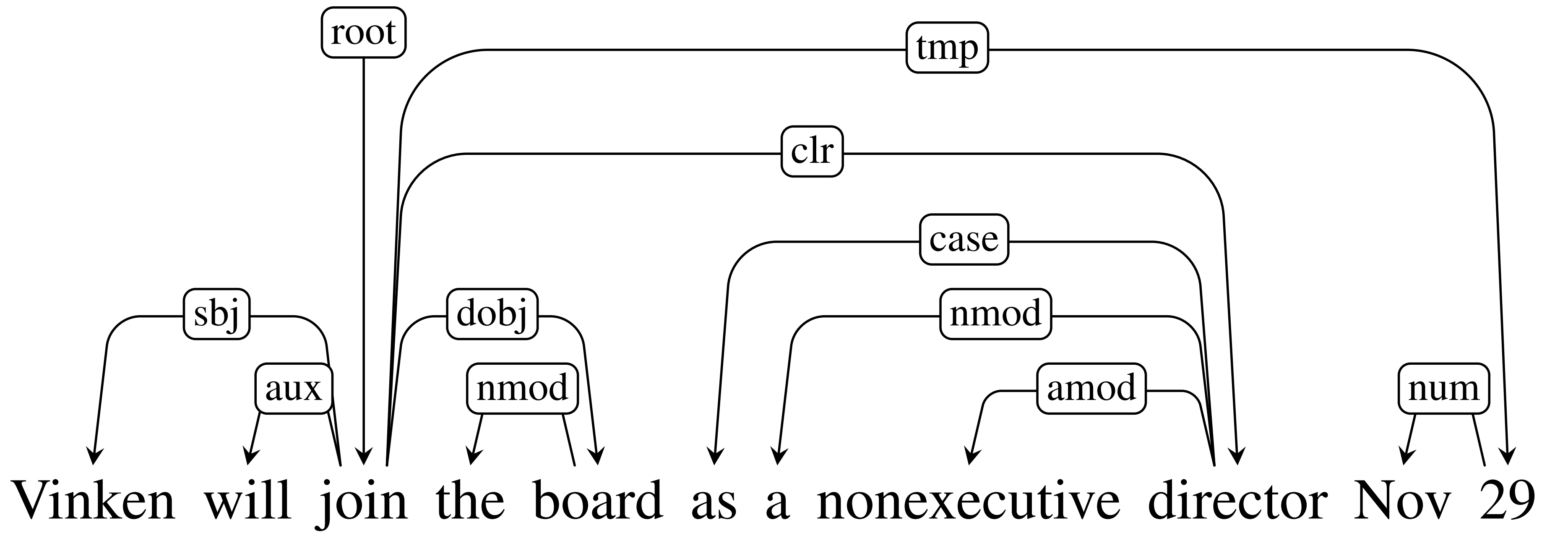
- Not really a grammar, in sense of a '*generative grammar*'
  - cannot be said to define a language, unlike a context free grammar
  - any structure is valid, job of *probabilistic model* to differentiate between poor and good alternatives
- However, very practical and closely matches what we want from a parser (most often predicates & arguments)

Dependency treebanks

- A few dependency treebanks
  - Czech, Arabic, Danish, Dutch, Greek, Turkish …
- Many more phrase-structure treebanks, which can be *converted* into dependencies
- More recently, *Universal Dependency Treebank*
  - collates >100 treebanks, >60 languages
  - unified part-of-speech, morphology labels, relation types
  - consistent handling of conjunctions and other tricky cases
- http://universaldependencies.org/

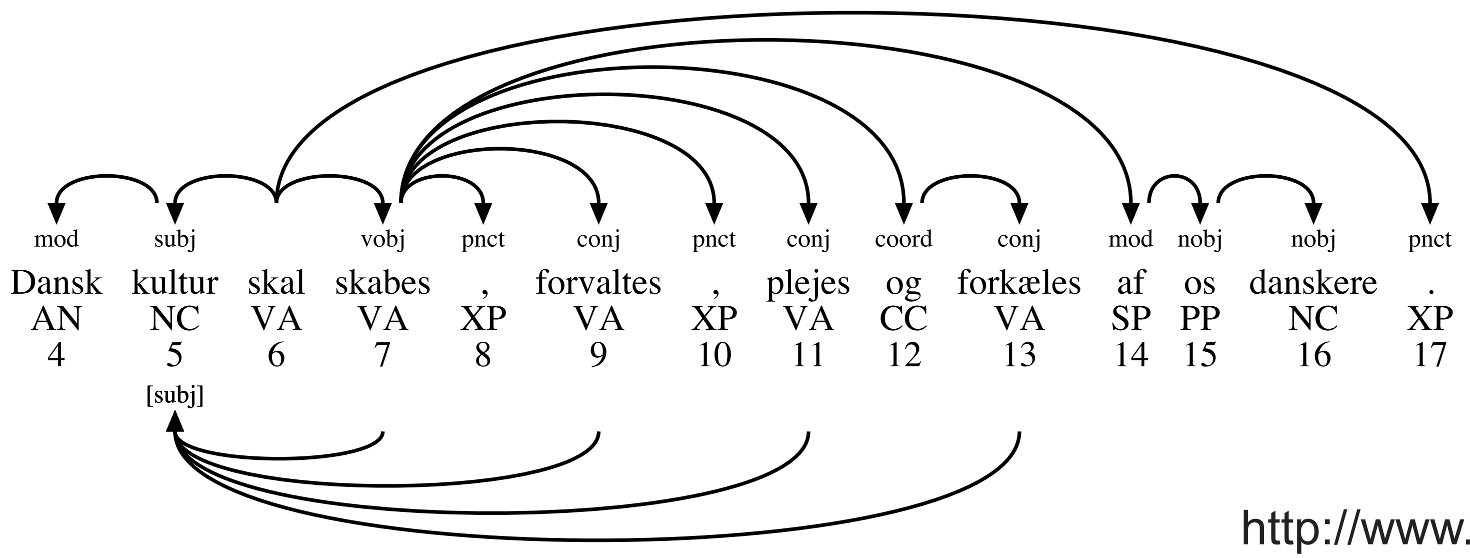
Treebank conversion

- Some treebanks automatically converted into dependencies
  - using various heuristics, e.g., head-finding rules
  - often with manual correction



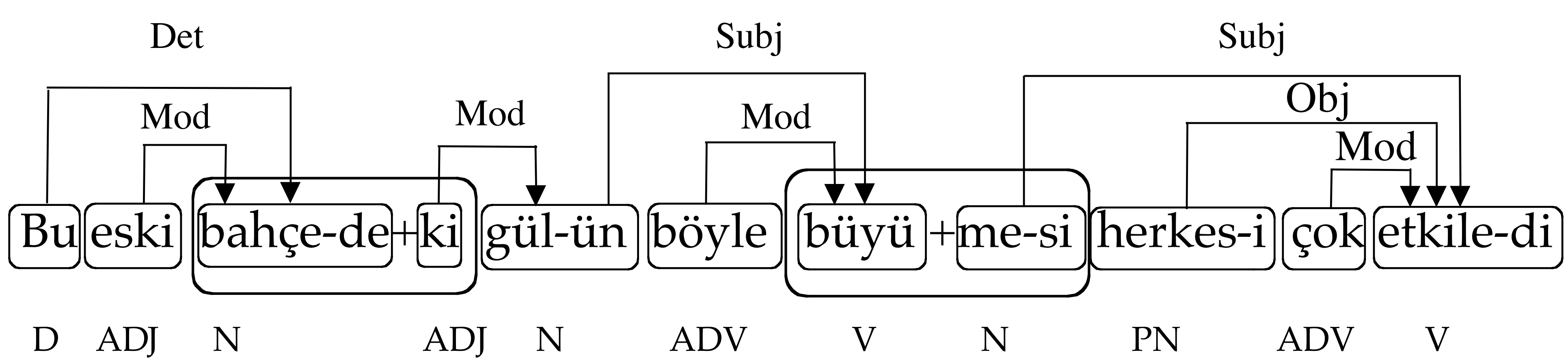
Examples from treebanks

- Danish DDT includes additional 'subject' link for verbs



http://www.buch-kromann.dk/matthias/ddt1.0/

- METU-Sabancı Turkish treebank
  - edges between morphological units, not just words (-,+)



Oflazer, Kemal, et al. "Building a Turkish treebank." *Treebanks*. Springer, 2003. 261-277.

Dependency parsing

- Parsing: task of finding the *best* structure for a given input sentence
  - i.e., $\textit{arg max}_{\boldsymbol{t}}\ \text{score}(\boldsymbol{t}|\boldsymbol{w})$
- Two main approaches:
  - *graph-based*: uses *chart* over possible parses, and dynamic programming to solve for the maximum
  - *transition-based*: treats problem as incremental sequence of decisions over next action in a state machine

Transition based parsing

- Frames parsing as sequence of simple parsing transitions
  - maintain two data structures
    - *buffer* = input words yet to be processed
    - *stack* = head words currently being processed
  - two types of transitions
    - *shift* = move word from buffer on to top of stack
    - *arc* = add arc (left/right) between top two items on stack (and *remove* dependent from stack)

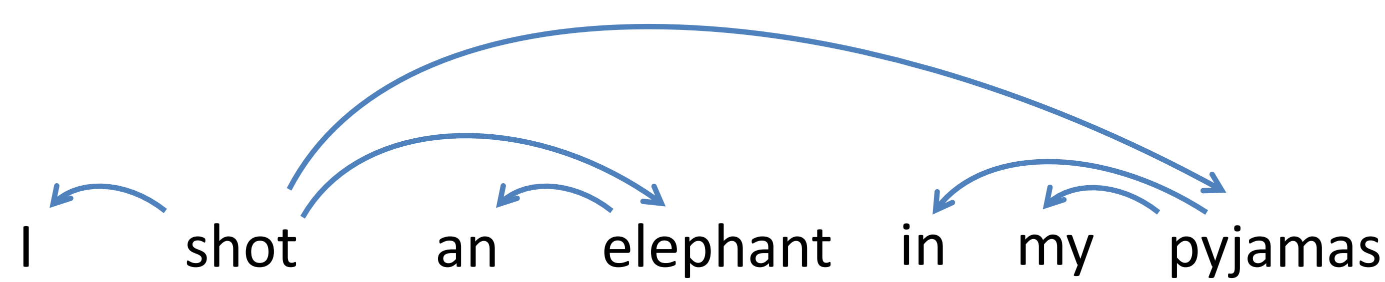
Transition based parsing algorithm

- For each word in input (buffer)
  - *shift* current word from buffer onto stack
  - while there are 2 or more items on stack:
    - either:
      - a) add an *arc (left or right)* between top two items, and remove the dependent; or
      - b) continue to outer loop
- Finished when buffer empty & stack has only 1 item
- Always results in a *projective tree*

Example

Buffer	Stack	Action
I shot an elephant in my pyjamas		Shift
shot an elephant in my pyjamas	I	Shift
an elephant in my pyjamas	I, shot	Arc-left
an elephant in my pyjamas	shot	Shift
elephant in my pyjamas	shot, an	Shift
in my pyjamas	shot, an, elephant	Arc-left
in my pyjamas	shot, elephant	Arc-right
in my pyjamas	shot	Shift
…	…	…
	shot	<done>

Generated parse:



Transition based parsing models

- How do we know when to *arc* and whether to add *left* or *right* facing arcs?
- Use a scoring function, *score(buffer, stack, transition),* based on the state, i.e.,
  - the next word(s) in the buffer
  - the contents of the stack, particularly the top two items
  - the transition type, one of *{continue, arc-left, arc-right}*
- Then select the *transition* with the highest score (greedy search)

Transition based scoring

- Form a feature representation for the state
- Example features, ϕ
  - [stack top has tag NN & next in stack has tag DT & transition = arc-left]
  - [stack top has tag NN & next in stack has tag DT & transition = arc-right]
  - [stack top has tag NN & next in stack is “has” & transition = arc-right]
  - [stack top has tag JJ & next in stack has tag DT & transition = shift]
- Have a *weight* for each feature, ***w***
  - such that the parser can choose between the possible transitions (e.g., arc-left, arc-right, shift)

Fig from Goldberg & Nivre (2012)

Training a Transition-based Dep Parser

- How to learn the feature weights from data? Perceptron training (Goldberg & Nivre, COLING 2012)
  - uses an "oracle" sequence of parser actions
  - predict next action in sequence, and update when model disagrees with gold action

Algorithm 2 Online training with a static oracle

1: $\mathbf{w} \leftarrow 0$
2: **for** $I = 1 \rightarrow$ ITERATIONS **do**
3:     **for** sentence x with gold tree G_{gold} in corpus **do**
4:         $c \leftarrow c_s(x)$
5:         **while** c is not terminal **do**
6:             $t_p \leftarrow \arg\max_t \mathbf{w} \cdot \phi(c, t)$
7:             $t_o \leftarrow o(c, G_{\text{gold}})$
8:             **if** $t_p \neq t_o$ **then**
9:                 $\mathbf{w} \leftarrow \mathbf{w} + \phi(c, t_o) - \phi(c, t_p)$
10:             $c \leftarrow t_o(c)$
11: **return w**

Graph based parsing

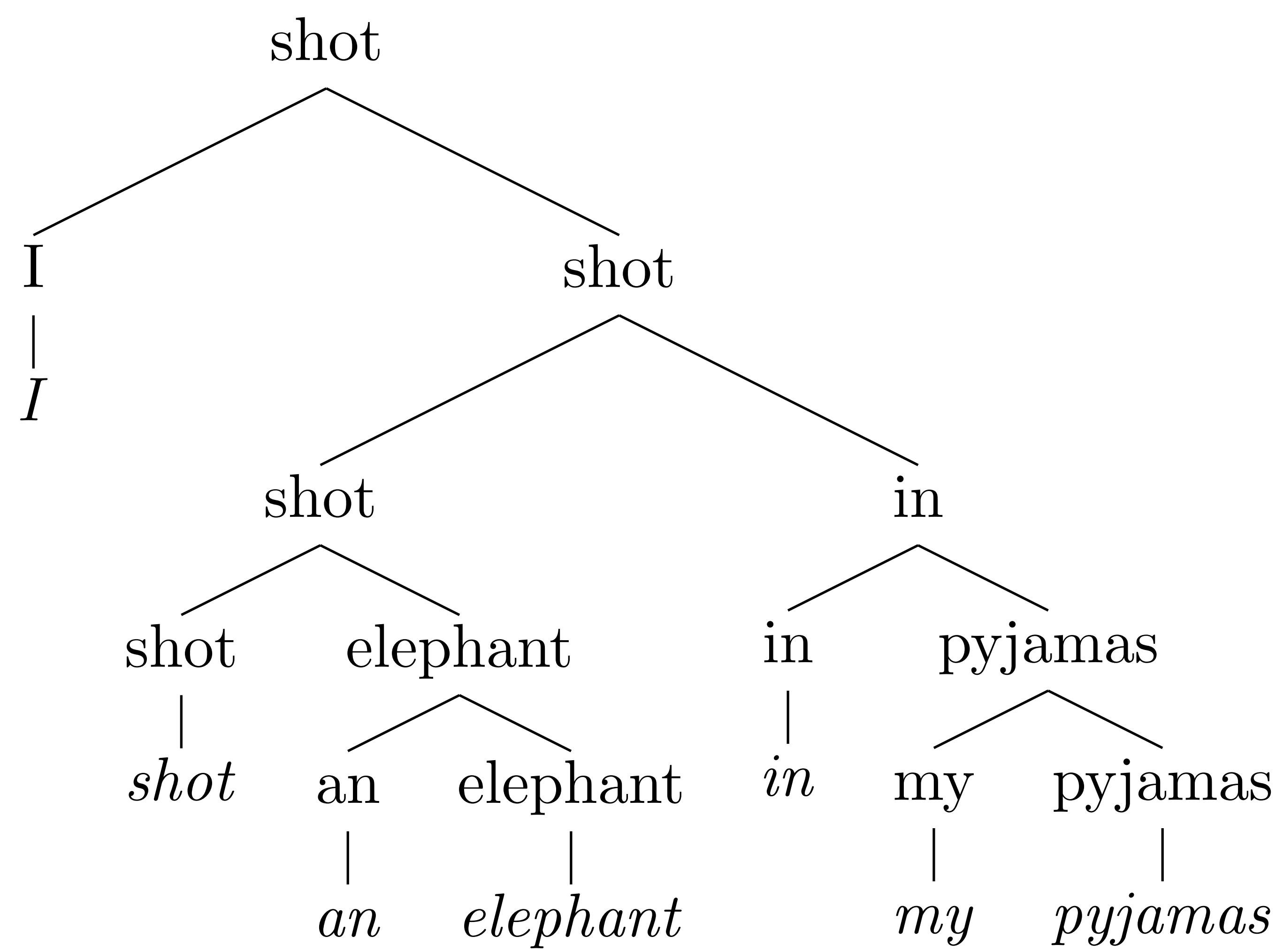
- Dependency parsing using dynamic programming...
  - Can consider as a CFG, where lexical items (heads) are non-terminals

E.g., production

shot → shot in

means *arc-right from "shot" to "in"*

The head is carried up in the tree.



  - Score of parse assumed to decompose into pairwise dependencies

Graph based parsing

- Naïve method for using CYK inefficient
  - Parsing complexity $O(n^5)$
  - split encoding allows processing of left or right dependents separately, leading to $O(n^3)$ runtime (Johnson, 2017)
- Alternatively can use Chiu-Liu-Edmond's algorithm
  - minimum cost arborescence (spanning tree)

A final word

- Dependency parsing a compelling, alterative, formulation to constituency parsing
  * structures based on words as internal nodes
  * edges encode word-word syntactic and semantic relations
  * often this is the information we need for other tasks!
- Transition-based parsing algorithm
  * as sequence of shift and arc actions
- Graph-based parsing
  * uses classic dynamic programming methods (similar to CYK)

Required Reading

- J&M3 Ch. 13