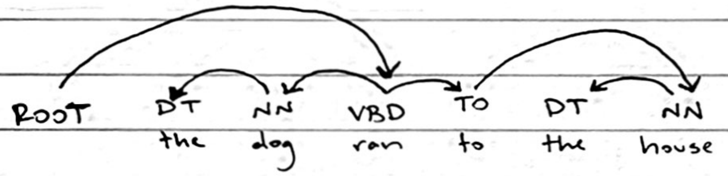
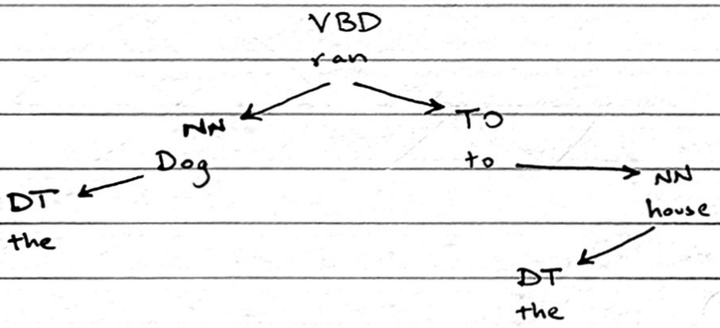
Dependencies

- **Dependency Syntax**: Syntactic structure is defined by these arcs
  - Head (parent, governor) connected to dependent (child, modifier)
  - Each word has exactly one parent except for root symbol
  - Dependencies must form acyclic graph (directed acyclic graph)



* POS tags same as before, often run tagger first as pre-processing

- Still a notion of hierarchy - subtrees often align with constituents



* Stanford Dependencies designed to be practical for relation extraction

* Universal dependencies project : Annotate dependencies w/ same representation in many languages. Dependencies are more portable cross-lingually : languages with free word order are not well handled by constituency parsers

* Projectivity :

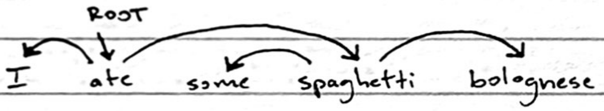
- Any subtree is contiguous span of the sentence ⟶ tree is projective
- Equivalent to drawing the structure and none of the arcs cross
- Many trees in other languages are non-projective

Transition-based Dependency Parsing

- We can build a dependency parser using chart-based algorithm like CKY, but time complexity is $O(n^3)$ and algorithm is very tricky

- Transition based or Shift Reduced is another style of parser; similar to deterministic parsing for compilers
    - A tree is built from sequence of incremental decisions moving left to right through the sentence
    - Stack contains partially built tree, Buffer contains rest of sentence

- Transition System



- Initial State : Stack - [ROOT] Buffer - [I ate some sphaghetti bolognese]
- Shift : Top of buffer → top of stack
    - Shift 1 : Stack - [ROOT I] Buffer - [ate some spaghetti bolognese]
    - Shift 2 : Stack - [ROOT I ate] Buffer - [some spaghetti bolognese]

- Left-arc (reduce) : Let σ denote stack, $\sigma | w_{-1}$ = stack ending in w_{-1}
    - Pop 2 elements, add an arc, put them back on stack

$\sigma | w_{-2}, w_{-1} \longrightarrow \sigma | w_{-1}$ w_{-2} is now a child of w_{-1}

state: Stack - [ROOT $\underset{\text{I}}{\overset{}{\text{ate}\downarrow}}$] Buffer - [some spaghetti bolognese]

- Right-arc : $\sigma | w_{-2}, w_{-1} \longrightarrow \sigma | w_{-2}$ w_{-1} is now child of w_{-2}
- End State : Stack contains [ROOT], Buffer is empty [ ]
    ↪ words are children of root

* How many transitions for a sentence w/ n words? → $\boxed{2n}$

* Full Algorithm walkthrough on "seg-39.pdf"

- When building these parsers :
  - How do we know which operation to use? (S, LA, RA)
  - In some cases, all 3 actions are legal
  - Multiway classification problem : S, LA, or RA?

$$\underset{a \in [S, LA, RA]}{\text{argmax}} \; w^T f(\text{stack}, \text{buffer}, a)$$

- <u>Features for Shift-Reduce Parsing</u>

$$[\text{ROOT} \;\; \underset{\text{I}}{\overset{}{\text{ate}\downarrow}} \;\; \text{some} \;\; \text{spaghetti}] \qquad\qquad [\text{Bolognese}]$$

  - Features to know this should be LA?
  - <u>In this case</u> : the stack tag sequence VBD - DT - NN is informative - looks like a verb taking a direct object which has determiner
  - <u>Things to look at</u> : top words/POS of Buffer, top words/POS of stack, leftmost and rightmost children of top items on stack

- <u>Training a Greedy Model</u>

$$\underset{a \in [S, LA, RA]}{\text{argmax}} \; w^T f(\text{stack}, \text{buffer}, a)$$

  - Con turn tree into decision sequence "a" by building an <u>oracle</u>
  - Train a classifier to predict the right decision using these as training data
  - 2n local training examples