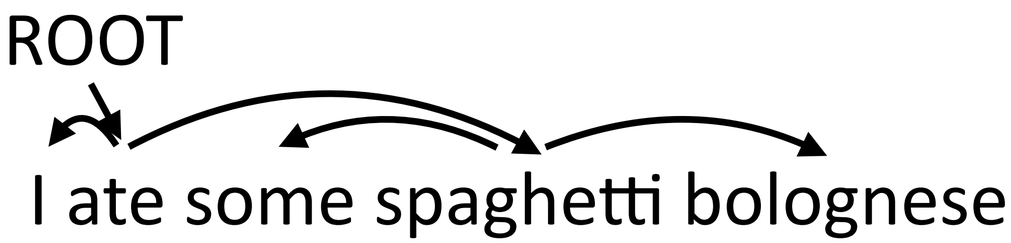
Transition-based Parsing

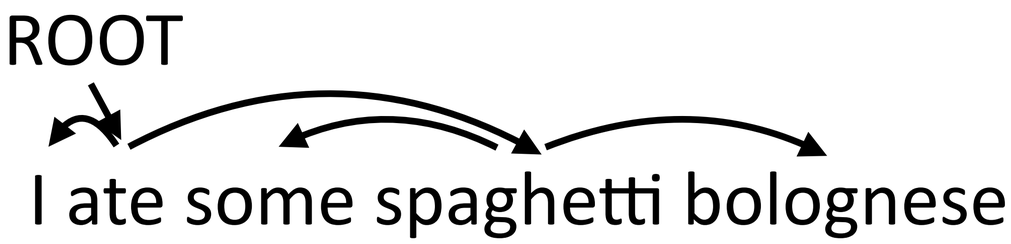
- We can build a dependency parser using a chart-based algorithm like CKY
  - Time: $O(n^3)$, but the algorithm is very tricky!
- *Transition-based*, or *shift-reduce*, is another style of parser; similar to deterministic parsing for compilers
- A tree is built from a sequence of incremental decisions 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 spaghetti 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]

Transition System



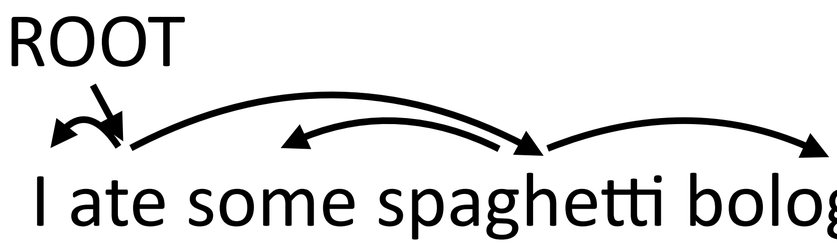
- State: Stack: [ROOT I ate] Buffer: [some spaghetti bolognese]

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

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

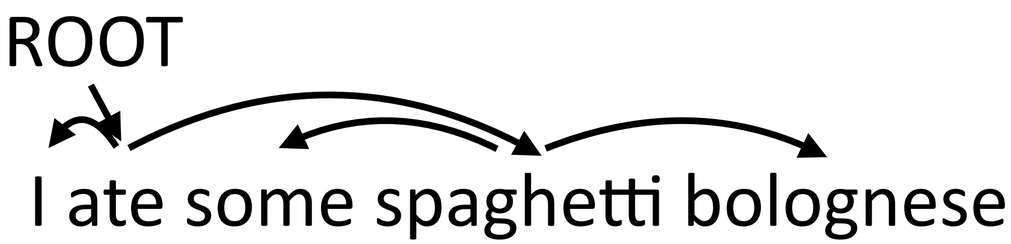
- State: Stack: [ROOT ate] Buffer: [some spaghetti bolognese]

Arc-Standard Parsing

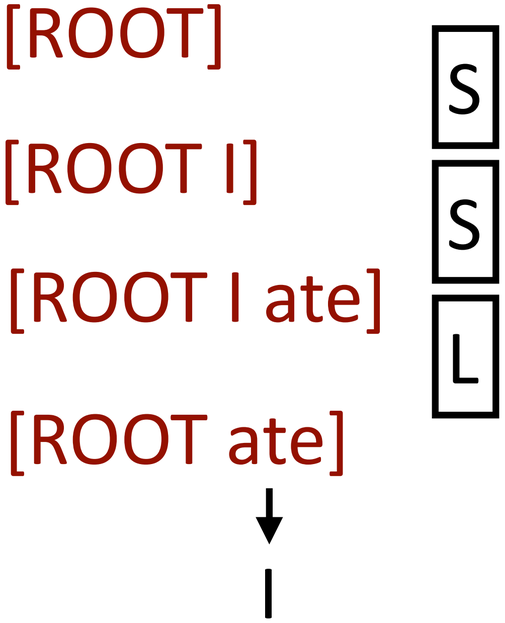


- Start: stack: [ROOT], buffer: [I ate some spaghetti bolognese]
- Arc-standard system: three operations
  - Shift: top of buffer -> top of stack
  - Left-Arc: $\boxed{\sigma|w_{-2}, w_{-1}} \rightarrow \boxed{\sigma|w_{-1}}$, w_{-2} is now a child of w_{-1}
  - Right-Arc $\boxed{\sigma|w_{-2}, w_{-1}} \rightarrow \boxed{\sigma|w_{-2}}$, w_{-1} is now a child of w_{-2}
- End: stack contains [ROOT], buffer is empty []
- How many transitions do we need if we have n words in a sentence?
- There are other transition systems, but we won't discuss these

Arc-Standard Parsing

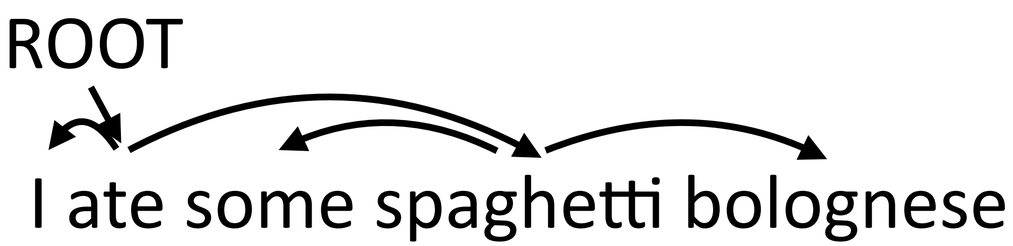


S	top of buffer -> top of stack
LA	pop two, left arc between them
RA	pop two, right arc between them

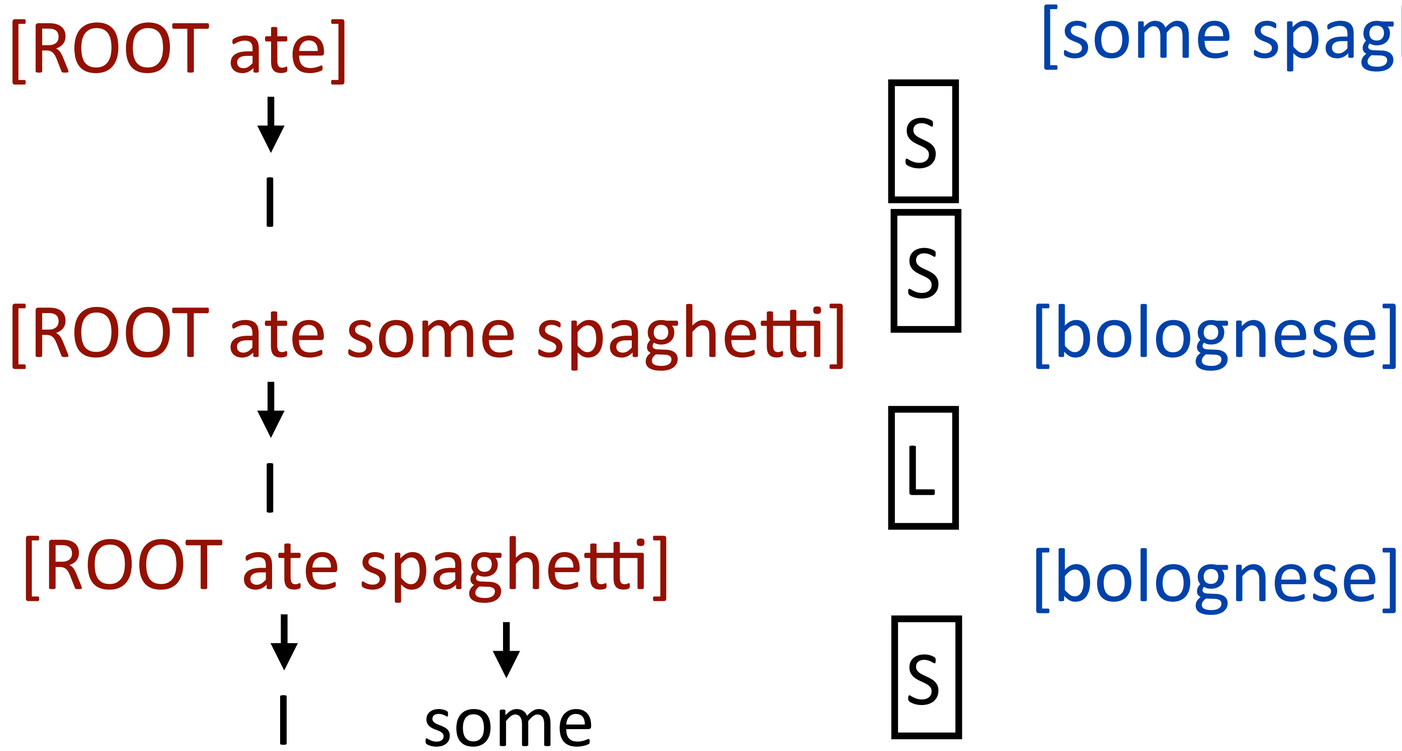


- Could do the left arc later! But no reason to wait
- Can’t attach ROOT <- ate yet even though this is a correct dependency!

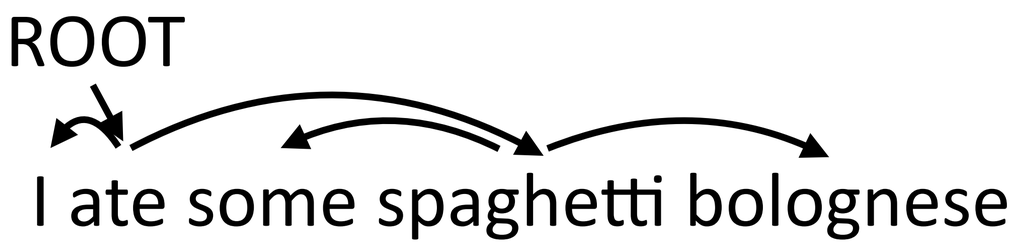
Arc-Standard Parsing



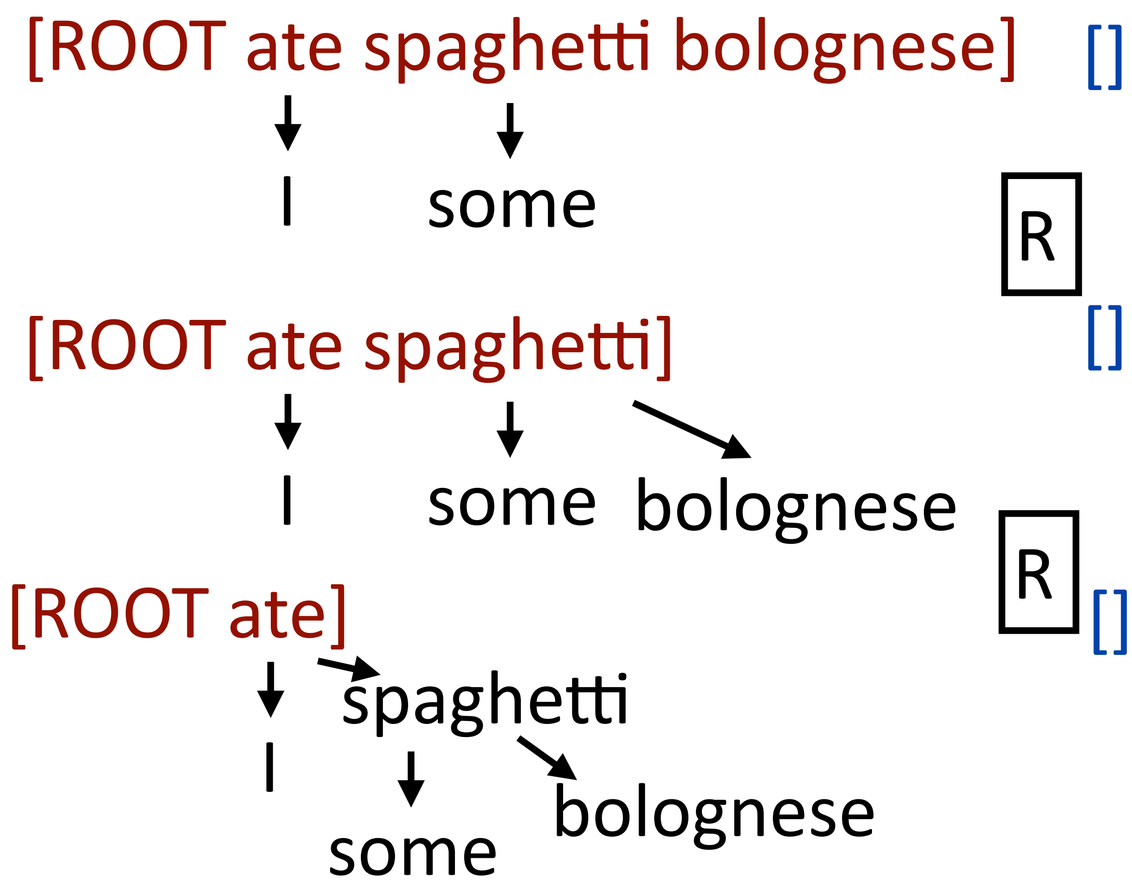
S	top of buffer -> top of stack
LA	pop two, left arc between them
RA	pop two, right arc between them



Arc-Standard Parsing

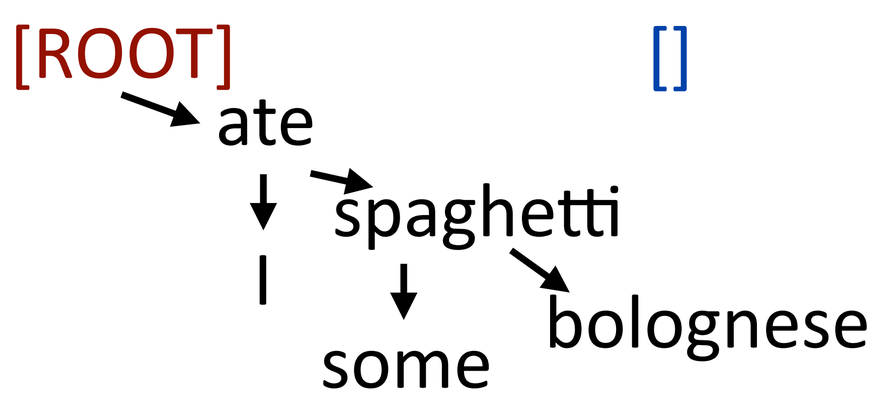


S top of buffer -> top of stack
LA pop two, left arc between them
RA pop two, right arc between them



Stack consists of all words that are still waiting for right children, end with a bunch of right-arc ops

Final state:



Building Transition-Based Parsers

[ROOT] [I ate some spaghetti bolognese]

- How do we make the right decision in this case?
- Only one legal move (shift)

[ROOT ate some spaghetti] [bolognese]

- How do we make the right decision in this case? (all three actions legal)
- Multi-way classification problem: shift, left-arc, or right-arc?

$$\text{argmax}_{a \in \{\text{S},\text{LA},\text{RA}\}} w^\top f(\text{stack}, \text{buffer}, a)$$

Features for Shift-Reduce Parsing

[ROOT ate some spaghetti] [bolognese]

- Features to know this should left-arc?
- One of the harder feature design tasks!
- In this case: the stack tag sequence VBD - DT - NN is pretty informative — looks like a verb taking a direct object which has a determiner in it
- Things to look at: top words/POS of buffer, top words/POS of stack, leftmost and rightmost children of top items on the stack

Training a Greedy Model

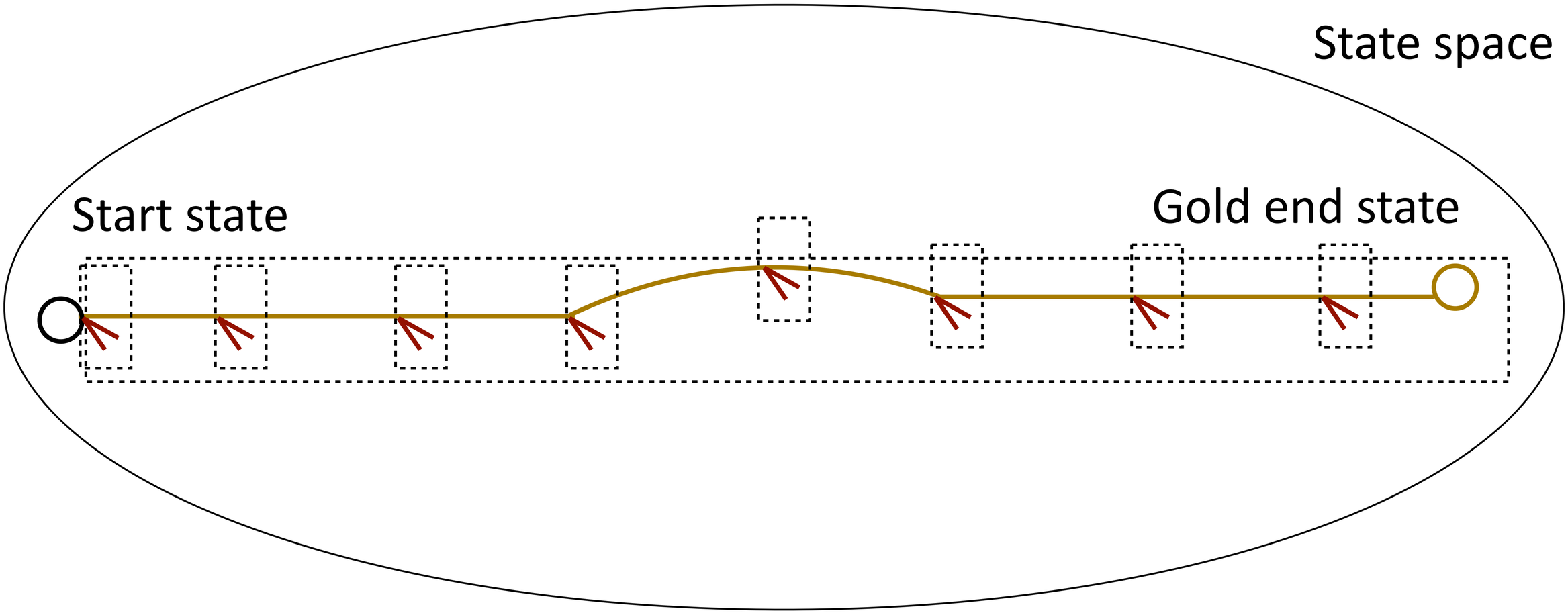
[ROOT ate some spaghetti] [bolognese]

$$\text{argmax}_{a \in \{\text{S},\text{LA},\text{RA}\}} w^\top f(\text{stack}, \text{buffer}, a)$$

- Can turn a tree into a decision sequence ***a*** by building an *oracle*
- Train a classifier to predict the right decision using these as training data

- Training data assumes you made correct decisions up to this point and teaches you to make the correct decision, but what if you screwed up…

Training a Greedy Model



- Greedy: 2n local training examples
- Non-gold states unobserved during training: consider making bad decisions but don't *condition* on bad decisions