Building a Parser

top-down parsers

- A **top-down** parser determines the structure of a parse tree by expanding it from the root node down
  - Expands the tree in *pre-order*
  - For each node in the parse tree, figure out what it expands to
- LL(1): Top-down derivation using 1 symbol of *lookahead*
- Common implementations:
  - Recursive descent: parser is a set of mutually-recursive functions
  - LL(1) parser: table-based parser that operates similarly to recursive-descent

context free grammars as functions

- Every nonterminal corresponds to a function:
  - X(): consume a prefix of the input to match X
  - B(): consume a prefix of the input to match B
- Think about writing a function to "match" a string to a non-terminal:

  Match X → a a B c against a a b b c
- If there is a terminal in the rule, match up the terminal against the string
  - Match X → a a B c against a a b b c
  - Match X → a a B c against a a b b c
- If there is a non-terminal in the rule, *call the function* for that non-terminal with the rest of the string and assume that it does its job:
  - Match X → a a B c against a a b b c
- When that function returns, keep matching the non-terminal
  - Match X → a a B c against a a b b c

X → a a B c

B → b b

how to match

- To match a non-terminal against a string, walk over the symbols of the right hand side of the rule
  - If it's a terminal, consume that token off the string
  - If it's a non-terminal, call the function for that non-terminal [which will consume characters off the string matching that non-terminal]
- Matching a rule may not consume all the tokens on a string
  - Just return the rest of the string from the function [think: what if this function was called recursively?]

- What if there are multiple rules for a non-terminal?

disambiguating multiple rules

- Suppose we call the function X() to match the non-terminal X in a string
- 3 choices! How do we know what tokens to match in the string?
- Idea:
  - Look at the **first** token on the string we're trying to match
  - What rule could generate that token?

$X \rightarrow a\ Y\ q$

$X \rightarrow b$

$X \rightarrow Y$

$Y \rightarrow c$

$Y \rightarrow d$

disambiguating multiple rules

- Suppose we call the function X() to match the non-terminal X in a string
- 3 choices! How do we know what tokens to match in the string?
- Idea:
  - Look at the **first** token on the string we're trying to match
  - What rule could generate that token?

Any string generated by this rule has to start with an 'a'

$X \rightarrow a\ Y\ q$

$X \rightarrow b$

$X \rightarrow Y$

$Y \rightarrow c$

$Y \rightarrow d$

disambiguating multiple rules

- Suppose we call the function X() to match the non-terminal X in a string
- 3 choices! How do we know what tokens to match in the string?

- Idea:
  - Look at the **first** token on the string we're trying to match
  - What rule could generate that token?

Any string generated by this rule has to start with a 'b'

$X \rightarrow a\ Y\ q$

$X \rightarrow b$

$X \rightarrow Y$

$Y \rightarrow c$

$Y \rightarrow d$

disambiguating multiple rules

- Suppose we call the function X() to match the non-terminal X in a string
- 3 choices! How do we know what tokens to match in the string?

- Idea:
  - Look at the **first** token on the string we're trying to match
  - What rule could generate that token?

What about this rule?

$X \rightarrow a\ Y\ q$

$X \rightarrow b$

$X \rightarrow Y$

$Y \rightarrow c$

$Y \rightarrow d$

disambiguating multiple rules

- Suppose we call the function X() to match the non-terminal X in a string
- 3 choices! How do we know what tokens to match in the string?
- Idea:
  - Look at the **first** token on the string we're trying to match
  - What rule could generate that token?

What about now?

$X \to a\ Y\ q$

$X \to b$

$X \to \lambda$

$Y \to c$

$Y \to d$

first and follow sets

- Figuring out which token to look for to match a given rule is complicated
- But we can simplify this by computing **first** and **follow** sets
  - **First(α)** = what terminals (or λ) might *start* any string you derive from α
    - If I start with α and apply rules, what terminals might the string start with?
  - **Follow(X)** = what terminals might *come after* the non-terminal X
    - If I start with the *start symbol* and apply rules, what terminals can I make come after X?

first and follow sets

Special symbol we put at the end of the start rule

- First sets defined for strings:
  - First(abX) = {a}
  - First(Y) = {λ, d}
  - First(S) = {a, b, d, $}
- Follow sets defined for non-terminals:
  - Follow(X) = {d, $}
  - Follow(Y) = {q, d, $}

$S \rightarrow X\ Y\ \$$

$X \rightarrow a\ Y\ q$

$X \rightarrow b$

$X \rightarrow Y$

$Y \rightarrow \lambda$

$Y \rightarrow d$

next: computing first and follow sets