When LL(1) Fails

recall: how lookahead works

- Build the function for each non-terminal:
    - Switch on the lookahead token in the string, pick rule to expand based on predict sets of the rules
- What if no rule matches the lookahead token?
  String not part of the language

$S \rightarrow X\ Y\ \$$

$X \rightarrow a\ Y\ q$

$X \rightarrow b$

$X \rightarrow Yq$

$Y \rightarrow \lambda$ $\{q, \$\}$

$Y \rightarrow d$ $\{d\}$

how can this go wrong?

- What if more than one rule matches the lookahead token? Grammar is not LL(1) — cannot be parsed top-down with one token of lookahead

$S \rightarrow X\ Y\ \$$

$X \rightarrow a\ Y\ q$

$X \rightarrow b$

$X \rightarrow \mathbf{Y}$

$Y \rightarrow \lambda$

$Y \rightarrow d$

First(S) = {a, b, d, \$}	Follow(S) = { }
First(X) = {a, b, d, λ}	Follow(X) = {d, \$}
First(Y) = {d, λ}	Follow(Y) = {d, q, \$}

how can this go wrong?

- What if more than one rule matches the lookahead token? Grammar is not LL(1) — cannot be parsed top-down with one token of lookahead

$S \rightarrow X\ Y\ \$$

$X \rightarrow a\ Y\ q$

$X \rightarrow b$

$X \rightarrow \mathbf{Y}$

$Y \rightarrow \lambda$ $\{\boldsymbol{d}, \boldsymbol{q}, \$\}$

$Y \rightarrow d$ $\{\boldsymbol{d}\}$

First(S) = {a, b, d, \$}	Follow(S) = { }
First(X) = {a, b, d, λ}	Follow(X) = {d, \$}
First(Y) = {d, λ}	Follow(Y) = {d, q, \$}

how to fix?

- Sometimes can look ahead more (make an LL(k) grammar):

$S \rightarrow a\ Y$

$S \rightarrow a\ Z$

$Y \rightarrow b$

$Z \rightarrow c$

- Sometimes, more lookahead does not help:

$S \rightarrow E$

$E \rightarrow (E + E)$

$E \rightarrow (E - E)$

$E \rightarrow x$

other fixes

- Can rewrite:

$$\begin{array}{l} S \rightarrow E \\ E \rightarrow (E + E) \\ E \rightarrow (E - E) \\ E \rightarrow x \end{array} \longrightarrow \begin{array}{l} S \rightarrow E \\ E \rightarrow (E\ O\ E) \\ E \rightarrow x \\ O \rightarrow + \mid - \end{array}$$

- Left recursion needs rewriting:

$$\begin{array}{l} S \rightarrow E \\ E \rightarrow E + x \\ E \rightarrow x \end{array} \longrightarrow \begin{array}{l} S \rightarrow E \\ E \rightarrow x\ E' \\ E' \rightarrow +\ x\ E' \\ E' \rightarrow \lambda \end{array}$$

- Or could use more powerful parser:
  - Backtracking parser, *bottom-up* parser

next: taking action

But first: A detour