Lec 8 Parsing I

Context-free grammar (CFG)

Def A CFG G consists of
- A finite set of terminal symbols $\Sigma = \{a, b, \dots\}$
- A finite set of non-terminal symbols $N = \{A, B, \dots\}$
- A finite set of productions $A \to \alpha$ where $\alpha \in (N \cup \Sigma)^*$
- A start symbol $S \in N$

Ex. $S \xrightarrow{\text{emp}} \varepsilon$ $\quad S \xrightarrow{\text{par}} [S]$ $\quad S \xrightarrow{\text{dup}} SS$

Def $L(G)$ for G: CFG denotes the <u>language</u> of G $\{\alpha \in \Sigma^* \mid S \to^* \alpha\}$
aka a set of all derivable sentences

Def A <u>derivation</u> $S \to^* \alpha$ is a seq of production applications

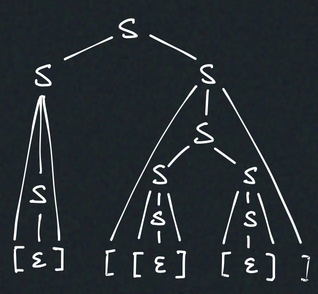
Def A production rule <u>applications</u> is a step $\beta_1 A \beta_2 \to \beta_1 \alpha \beta_2$ (not care about context)
provided that $A \to \alpha$ is a production rule

Ex. Derive `[][[][]]`

```
S              [][[S][S]]  →  [][[][S]]
↓                  ↑              ↓
SS             [][[S]S]       [][[][]]
↓                  ↑
[S][]          [][SS]
↓                  ↑
[S][S]    →    [][S]
```

Observe: order often doesn't matter

Parse tree — abstract away order



Problems
- how to parse bottom up?
- which parse tree to generate if multiple are valid?

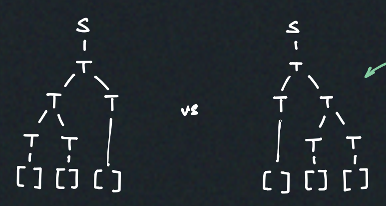
Ambiguity

Ex. in above example, $S \xrightarrow{\text{dup}} SS \xrightarrow{\text{emp}} S$

1. Maybe replace emp

$S \xrightarrow{\text{emp}} \varepsilon$ $\quad S \xrightarrow{\text{non-emp}} T$ $\quad T \xrightarrow{\text{sing}} []$ $\quad T \xrightarrow{\text{dup}} TT$ $\quad T \xrightarrow{\text{par}} [T]$

More problem:



[1] Suppose we want to only allow this one

2. [1] Only allow further duplication on one side

$S \xrightarrow{\text{emp}} \varepsilon$ $\quad S \xrightarrow{\text{non-emp}} T$ $\quad T \xrightarrow{\text{sing}} []$ $\quad T \xrightarrow{\text{dup}} UT$ $\quad U \to []$ $\quad U \to [T]$ $\quad T \to U$

3. [2] This works too

$S \xrightarrow{\text{emp}} \varepsilon$ $\quad S \xrightarrow{\text{next}} [S]S$

Shift-reduced parsing

Predictive parsing with lookahead
Keep state, one pass to parse (linear time)

Goal: given $w \in \Sigma^*$, find $S \to^* w$

Restriction: LR(k) grammar
- k: usually $k=1$, assume today
- k symbol lookahead
- R: right-most derivation: apply rule to right-most non-term
- L: left to right

{ Usually all LR(k) grammar can reduce to LR(1), so LR(1) general enough

Alg LR(1) parsing — Idea: use a transition system

Start state: $\varepsilon \parallel w_0$ (current location; ε: processed; w_0: not yet processed)

Final state: $S \parallel \varepsilon$ (S: parsed tree; ε: input consumed)

Intermediate state: $\gamma \parallel w_i$ (w_i: suffix)

Transitions
1. <u>Shift</u> $\quad \gamma \parallel aw \to \gamma a \parallel w$
2. <u>Reduce</u> $\quad \gamma\alpha \parallel w \to \gamma A \parallel w$
   if $A \to \alpha$ in grammar

Ex. Parse `[][[]]` with rules in [2]

```
ε ‖ [][[]]  →  [ ‖ ][[]]
   ↓              ↓
S ‖ [][[]]     [S ‖ ][[]]  →  [S] ‖ [[]]  →  [S][ ‖ []]
   ↓                             ↓              ↓
   :(                         [S]S ‖ [[]]    [S][S ‖ []]
                                 ↓              ↓
                                 :(             :(

[S][[ ‖ ]]
   ↓
[S][[S ‖ ]]  →  [S][[S] ‖ ]
                   ↓
                [S][[S]S ‖ ]
                   ↓
                [S][S ‖ ]  →  [S][S] ‖ ε
                                 ↓
                              [S][S]S ‖ ε
                                 ↓
                              [S]S ‖ ε
                                 ↓
                              S ‖ ε
```

observe reduce takes place on the right, so we get rightmost derivations

Parse Table

$\gamma \backslash a$	[	]	EOF
ε	shift	shift	red(emp)
⋮		⋮	

Problems if grammar not in LR(1)

- shift-reduce conflict — ex. $S \to SS$
  not know if to shift or reduce
  → try specify whether to shift or reduce
- reduce-reduce conflict — ∴ bad grammar :(