5 Top-down parsers

5.1 Top-Down Parsing Overview

Principles of Top-Down Parsing

- Parsing is the second step in compiling, analyzing syntax using **pushdown automata (PDA)**.
- **Top-down parsers** build derivation trees starting from the root (start symbol) to leaves (input string).
- **Key steps**: Simulate leftmost derivations by replacing variables with grammar rules, matching terminals with input.
- **Non-determinism** arises when multiple rules apply to the same variable; resolved using look-ahead.

Comparison with Bottom-Up Parsers

- **Bottom-up parsers** start from leaves (input) and work backward to the root.
- Top-down parsers are intuitive but less powerful; modern compilers (e.g., GCC, Clang) use hand-written top-down parsers.

5.2 Pushdown Automata (PDA) for Parsing

Constructing a PDA from a CFG

- For a CFG G, build a PDA P_G with:
  - **Produce transitions**: Replace a variable A with the right-hand side of a rule $A \to \alpha$.
  - **Match transitions**: Consume a terminal from the input if it matches the top of the stack.
- **Example**: For arithmetic expressions, PDA transitions simulate leftmost derivations.

Non-Determinism in PDAs

- Non-determinism occurs when multiple rules apply to the same variable.
- **Solution**: Use **look-ahead** to predict the correct rule.

5.3 Predictive Parsers with Look-Ahead

k-Look-Ahead PDAs (k-LPDAs)

- **Definition**: A PDA extended with k-character look-ahead to resolve non-determinism.
- **Semantics**: Transitions depend on the next k input characters without consuming them.
- **Equivalence to PDAs**: Any k-LPDA can be converted to an equivalent (non-deterministic) PDA.

Deterministic Parsing with Look-Ahead

- **Example**: A trivial grammar with rules $S \to a$ and $S \to b$ becomes deterministic with 1-character look-ahead.
- **Key insight**: Look-ahead allows the parser to choose rules based on future input.

5.4 First^k and Follow^k Sets

Definitions

- **First**$^k(\alpha)$: The set of terminal prefixes (up to length k) derivable from sentential form α.
- **Follow**$^k(A)$: Terminals that can appear immediately after A in any derivation.

Computation Algorithms

- **First^k**:
  1. Initialize terminals as their own First sets.
  2. Iteratively update First sets for variables using grammar rules.
- **Follow^k**:
  1. Initialize Follow(S) = $\{\varepsilon\}$.
  2. Propagate Follow sets through rules of the form $A \to \alpha B \beta$.

Example

- For the grammar

$$\begin{aligned} A &\to aaa \\ &\to Bbb \\ &\to Cdd \\ B &\to b \\ C &\to c \\ &\to \varepsilon \end{aligned}$$

compute:

- First$^1(A) = a, b, c, d$
- Follow^1(C) = $\{d\}$ (from context in $A \to Cdd$).

5.5 LL(k) Grammars

Definition and Conditions

- **LL(k) Grammar**: For every pair of derivations $S \Rightarrow^* wA\gamma$, the next k symbols uniquely determine the rule to apply.
- **Strong LL(k)**: A stricter syntactic condition requiring $\text{First}^k(\alpha \cdot \text{Follow}^k(A))$ for distinct rules $A \to \alpha_1$ and $A \to \alpha_2$ to be disjoint.

Hierarchy and Relationships

- **Strict Hierarchy**: $\text{LL}(k) \subsetneq \text{LL}(k+1)$.
- **LL(1) vs. Strong LL(1)**: All LL(1) grammars are strong LL(1), but this fails for $k \geq 2$.
- **DCFL Relationship**: $\bigcup_{k \geq 0} \text{LL}(k) \subsetneq \text{DCFL}$.

Example

- Grammar $S \to aAa \mid bABa$ is **LL(2)** but not LL(1) due to ambiguous look-ahead.

5.6 LL(1) Parsers

Action Table Construction

- **Structure**: Rows for stack symbols, columns for terminals.
- **Entries**: Rule numbers (produce), "Match," "Accept," or "Error."
- **Algorithm**:
  1. Initialize all cells to "Error."
  2. Fill cells using First and Follow sets for each rule.

Parsing Algorithm

1. Initialize the stack with the start symbol.
2. For each step:
   - **Produce**: Replace a variable with the right-hand side of a rule.
   - **Match**: Consume a terminal from the input.
   - **Accept/Error**: Based on stack and input state.

Recursive Descent Implementation

- **Key Idea**: Map each grammar rule to a function that checks look-ahead and invokes sub-functions.
- **Example**:

```
def parse_S():
    if lookahead in First(S → a):
        consume terminals and call sub-functions
```

Key Points to Remember

- **Top-Down Parsing**: Builds derivations from the start symbol using PDA simulations.
- **Look-Ahead**: Resolves non-determinism by previewing input (k-LPDAs).
- **First^k and Follow^k**: Critical for predicting rule applications in LL(k) grammars.
- **LL(k) Hierarchy**: Strict inclusion with increasing k; no finite k covers all DCFLs.
- **LL(1) Parsers**: Use action tables and recursive descent for deterministic parsing.
- **Strong LL(k)**: A syntactic subset of LL(k) grammars, equivalent for $k = 1$.