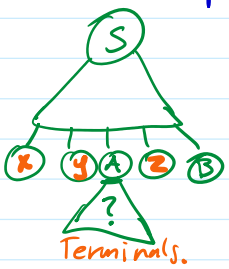
10 Recursive Descent Parsing

Monday, October 2, 2023 11:06 AM

- Types of Parsers.
  - Bottom-up parsers eg. Bison
  - Top-down parsers.
    - Produces parse tree from root to leaves.
    - eg. Recursive Descent Parser"
- Top-Down Parses
  - read input from Left-to-Right
  - produce a Leftmost derivation
  - also called LL parses.



- Most common types:
  - "Recursive descent" directy encodes grammar into functions.
  - "predictive parse table" - table driven algorithm.
- Limitations.-
  - Not all C.F.G. can we parsed this way, by an LL-parser.
  - LL-parses apply to a subclass of grammas called LL-grammars.

- RECURSIVE DESCENT PARSING:

global variable token the string from the input.
function gettoken() - read a new token from input.

Recipe: (to encode a gramma)
- One function per non-terminal symbol.

$A \to \alpha \beta \Gamma$
$A \to x\,y\,z$

parse_A()

- if a non-terminal has more than one rule the token should determine which rule to apply
- for each symbol α in the body
  - if α is a terminal symbol:
    compare with token,
    if same, consume token and read next token.
  - if α is a non-terminal symbol:
    call function parse_α()

E.G. #1

$S \rightarrow aA \mid bB$

```
FUNCTION parse_S()
  IF  token = 'a'  THEN
    getToken()
    parse_A()

  ELSIF  token = 'b'  THEN
    getToken()
    parse_B()

  ELSE
    error("Error when parsing S: 'a' or 'b' expected")
    quit("epic fail!");
  END
END.
```

- Starting the paser.

```
FUNCTION main()
  getToken()
  parse_S()   // S is the start symbol of the grammar
  IF  token # '$' THEN
    error("Expected end of input")
  END
END.
```

E.G #2.

$S \rightarrow dAc \mid b$
$A \rightarrow baB \mid \lambda$
$B \rightarrow aS$

```
FUNCTION parse_S()
  IF token = 'd' THEN
    getToken()
    parse_A()
    IF token = 'c' THEN
      getToken()
```

```
FUNCTION parse_A()
  IF token = 'b' THEN
    getToken()
    IF token = 'a' THEN
      getToken()
      parse_B()
    ELSE
      error("expecting a")
```

```
FUNCTION parse_B()
  IF token = 'a' THEN
    getToken()
    parse_S()
  ELSE
    error("expecting a")
```

TRACE:

```
parse_A()
IF token = 'c' THEN
  getToken()
ELSE
  error("expecting c")

ELSIF token = 'b' THEN
  getToken()

ELSE
  error()
```

TRACE:

cba$ reject.

dc$ token accept.

bbb$ reject.

dbaac$ Parse_S() Larson
parse_A() Reese
parse_B() Austin
parse_S() Lincoln
reject.

dbaadcc$

accept

parse_S() Hunter
parse_A() Matt
parse_B() Tim
parse_S() Ryan
parse_A() Aman

- EXTENDED BNF

We extend our format for grammar rules with 2 shorthands.

[ ] option

{ } repetition

e.g.

A → a[b]c

B → a{b}c

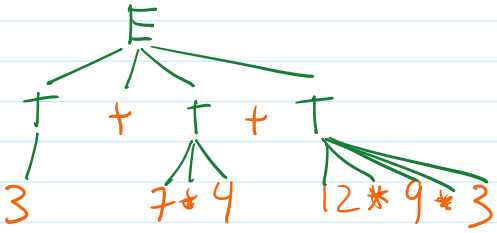
zero or more repetitions of b

A → ac
A → abc

B → ac
B → abc
B → abbc
B → abbbc
⋮

E.G.

E → T{+T}
T → int{*int}



3 + 7 * 4 + 12 * 9 * 3

Encoding.

```
FUNCTION parse_E()
  parse_T()
  WHILE token = "+" DO
    getToken()
    parse_T()
  END
END.
```

```
FUNCTION parse_T()
  parse_INT()
  WHILE token = "*" DO
    getToken()
    parse_INT()
  END
END.
```

E.G

S → if C then B [else B] fi

```
FUNCTION parse_S()
  IF token = "if" THEN
    getToken()
    parse_C()
    IF token = "then" THEN
      getToken()
      parse_B()

      IF token = "else" THEN
        getToken()
        parse_B()
      END

      IF token = "fi" THEN
        getToken()
      ELSE
         error("fi expected")

    ELSE
      error("then expected")
  ELSE
    error("if expected")
```

— EOF —