Top Down Parsing

TEACHING ASSISTANT: DAVID TRABISH

Examples

```
void f(int a) {
      ((((8))));
}
```

Examples

```
void f(int a) {
      ((((8))));
}
```

Valid

Examples

```
void f(int a) {
      ((((8);
}
```

Examples

```
void f(int a) {
      ((((8);
}
```

Invalid

Examples

```
void f(int a) {
      5;;;;;
}
```

Examples

```
void f(int a) {
        5;;;;;
}
```

Valid

Examples

```
void f(int a[]) {

}
```

Examples

```
void f(int a[]) {

}
```

Valid

Examples

```
void f() {
        int a[];
}
```

Examples

```
void f() {
        int a[];
}
```

Invalid

Examples

```
void f() {
        int a[10.0];
}
```

Examples

```
void f() {
        int a[10.0];
}
```

Invalid

Examples

```
void f() {
        int i = 0;
        i = 8;
        int j = i;
}
```

Examples

```
void f() {
        int i = 0;
        i = 8;
        int j = i;
}
```

Valid

Examples

```
void f() {
        int i = 0;
        i = 8;
        int j = z;
}
```

Examples

```
void f() {
        int i = 0;
        i = 8;
        int j = z;
}
```

Valid

Examples

```
void f() {
        int i = 0;
        i = 8;
        int j = i
}
```

Examples

```
void f() {
        int i = 0;
        i = 8;
        int j = i
}
```

Invalid

Examples

```
void f() {
        g()++;
}
```

Examples

```
void f() {
        g()++;
}
```

Valid

Language of Balanced Parentheses

Contains string of the form:

- 8, (1), (((03))), …

Not allowing:

- ((1), 8()

Language of Balanced Parentheses

Contains string of the form:

- 8, (1), (((03))), …

Not allowing:

- ((1), 8()

Is there a DFA/NFA that accepts the language?
Is there a regular expression the accepts the language?

Language of Balanced Parentheses

The language is **not regular**

- There is no DFA that accepts it

Proof:

- …

Context Free Grammar

Formal Definition: …

Context Free Grammar

Example:

- $S \rightarrow c$
- $S \rightarrow aSb$

Which words belong to this grammar?

Context Free Grammar

Example:

- $S \rightarrow c$
- $S \rightarrow aSb$

Which words belong to this grammar?

- $c, acb, aacbb, aaacbbb, \ldots$

Context Free Grammar

Is the language of **balanced parentheses** has a CFG?

Context Free Grammar

Is the language of **balanced parentheses** has a CFG?

- $S \rightarrow D \mid (D)$
- $D \rightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9$

Context Free Grammar: Questions

Are there languages which are **has no CFG**? Yes

Is it possible to have **multiple** CFG describing the same? Yes

Predictive Parser: Definition

- TODO…

Predictive Parser: Example

The language of balanced parentheses:

- $S \rightarrow D \mid (D)$
- $D \rightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9$

has a predictive parser.

Predictive Parser: Example

```
void parse_S() {
  switch (token) {
  case INT:
    parse_token(INT);
    break;
  case L_PAREN:
    parse_token(L_PAREN);
    parse_S();
    parse_token(R_PAREN);
    break
  default:
    // error
  }
}
```

```
void parse_token(int expected) {
  if (token == expected) {
      token = lexer.next_token();
  } else {
    // error
  }
}
```

Predictive Parser: Example

What happens for the input (7)?

Call trace:

- parse_S
  - parse_token // match with ‘(‘
  - parse_S
    - parse_token // match with ‘7‘
  - parse_token // match with ‘)‘

Predictive Parser: Example

What happens for the input ((7)?

Call trace:

- parse_S
  - parse_token // match with ‘(‘
  - parse_S
    - parse_token // match with ‘(‘
    - parse_S
      - parse_token // match with ‘7’
    - parse_token // match with ‘)’
  - parse_token // error, expecting ‘)’

Language of Balanced Parentheses 2

Find a CFG for a language with the 3 kinds of parentheses:

- (), [], {}

Contains string of the form:

- (([][]{}))[]
- [()]

Not allowing:

- (({))

Language of Balanced Parentheses 2

CFG definition:

- $S \rightarrow (S)S$
- $S \rightarrow [S]S$
- $S \rightarrow \{S\}S$
- $S \rightarrow \epsilon$

Language of Balanced Parentheses 2

```
void parse_S() {
  switch (token) {
  case L_PAREN:
    parse_S1();
    break;
  case L_BRACKET:
    parse_S2();
    break;
  case L_BRACE:
    parse_S3();
    break;
  default:
    // error
  }
}
```

```
void parse_S1() {
  parse_token(L_PAREN);
  parse_S();
  parse_token(R_PAREN);
  parse_S();
}
void parse_S2() {
  parse_token(L_BRACKET);
  parse_S();
  parse_token(R_BRACKET);
  parse_S();
}
void parse_S3() {
  parse_token(L_BRACE);
  parse_S();
  parse_token(R_BRACE);
  parse_S();
}
```

Calculator Language

A language with binary operators (+,-,*,/) and numbers:

- 1
- 1+1
- (1+1)*(7/2)
- 2+1-7

Calculator Language

A (possible) CFG for that language:

- $S \rightarrow N$
- $S \rightarrow S + S$
- $S \rightarrow S - S$
- $S \rightarrow S * S$
- $S \rightarrow S \ / \ S$
- $S \rightarrow (S)$

Calculator Language

A (possible) CFG for that language:

- $S \rightarrow N$
- $S \rightarrow S + S$
- $S \rightarrow S - S$
- $S \rightarrow S * S$
- $S \rightarrow S \ / \ S$
- $S \rightarrow (S)$

Will predictive parsing work here?

Left Recursion

There is no predictive parser which can handle the previous CFG

Why?

- If the first token was 5, we can't predict the right rule
- It can be 5 ($S \rightarrow N$)
- But also can be 5+8 ($S \rightarrow S + S$)

Left Recursion

Why it happens?

In the rule $S \rightarrow S + S$:

- S itself appears on the **left side** of the alternative

If we still want a predictive parser

- Need to **eliminate** left recursion

Left Recursion Elimination

If we have:

- $X \rightarrow a$
- $X \rightarrow Xb$

Then the language contains:

- $a, ab, abb, abbb, \ldots$

Define an alternative CFG:

- $X \rightarrow aY$
- $Y \rightarrow bY \mid \epsilon$

Left Recursion Elimination

In general, if we have:

- $X \rightarrow a_1 \mid a_2 \mid \ldots$
- $X \rightarrow Xb_1 \mid Xb_2 \mid \ldots$

We will rewrite as follows:

- $X \rightarrow a_1 Y | a_2 Y | \ldots$
- $Y \rightarrow b_1 Y | b_2 Y | \ldots | \epsilon$

Calculator Language

Before left recursion elimination:

- $S \rightarrow N$
- $S \rightarrow (S) \mid S + S \mid S - S \mid S * S \mid S / S$

What are our a_i, b_i ?

Calculator Language

Before left recursion elimination:

- $S \rightarrow N$
- $S \rightarrow (S) \mid S + S \mid S - S \mid S * S \mid S \,/\, S$

What are our a_i, b_i ?

- $a_1 = N, a_2 = (S)$
- $b_1 = +S, b_2 = -S, b_3 = *\,S, b_4 = /S$

Calculator Language

Before left recursion elimination:

- $S \rightarrow N$
- $S \rightarrow (S) \mid S + S \mid S - S \mid S * S \mid S \ / \ S$

The resulting CFG:

- $S \rightarrow NT \mid (S)T$
- $T \rightarrow +ST \mid -ST \mid * ST \mid /ST \mid \epsilon$

Calculator Language

TODO: Code snippet?

CFG vs Language

- A language may hove more the one CFG
- We might have a language which 2 CFG's where:
  - One has a predictive parser
  - The other one doesn't…

LL(1)

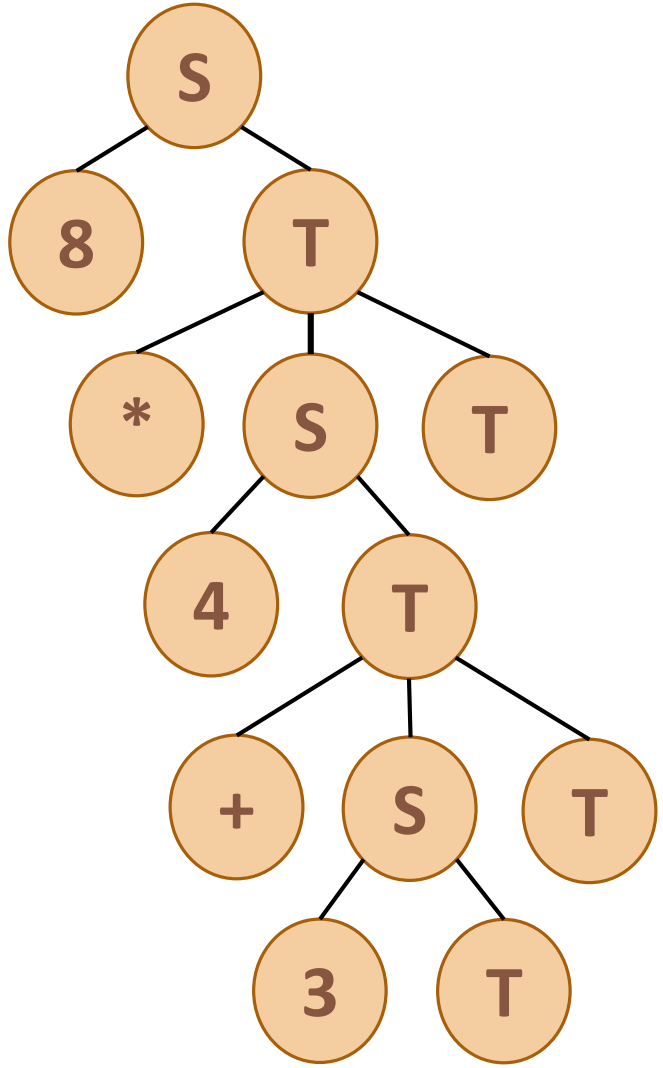
Definitions:

- A grammar that has a predictive parser is called LL(1)
- A language that has LL(1) grammar is called LL(1)

Derivation Tree

For the expression 8 * 4 + 3:

- S
- NT
- $N * ST$
- $N * NTT$
- $N * N + NTT$

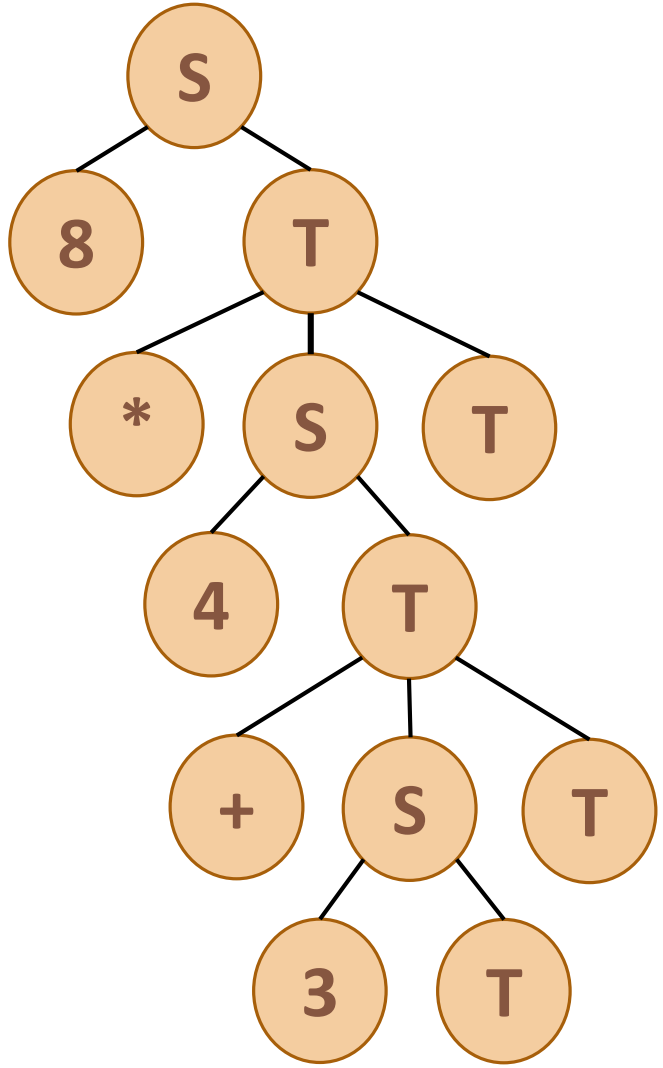


Derivation Tree

For the expression 8 * 4 + 3:

- S
- NT
- $N * ST$
- $N * NTT$
- $N * N + NTT$

Is that what we expect to have?



Operator Precedence

Our CFG does not contain information about **operator precedence!**

- The expression 8 * 4 + 3 is interpreted as 8 * (4 + 3)
- We need to find another grammar…

Operator Precedence

A CFG with **operator precedence:**

- $S \rightarrow S + T \mid S - T \mid T$
- $T \rightarrow T * F \mid T \ / \ F \mid F$
- $F \rightarrow N \mid (S)$

Operator Precedence

A CFG with **operator precedence:**

- $S \rightarrow S + T \mid S - T \mid T$
- $T \rightarrow T * F \mid T \ / \ F \mid F$
- $F \rightarrow N \mid (S)$

After eliminating **left recursion**:

- $S \rightarrow TS'$
- $S' \rightarrow +TS' \mid -TS' \mid \epsilon$
- $T \rightarrow FT'$
- $T' \rightarrow * FT' \mid /FT' \mid \epsilon$
- $F \rightarrow N \mid (S)$

Derivation Tree

With the new CFG, the derivation tree for 8 * 4 + 3: