

CMSC 330: Organization of Programming Languages

Recursive Descent Parser Example

Recursive Descent Parser

- For terminals, create function `match(a)`
 - If lookahead is `a` it consumes the lookahead by advancing the lookahead to the next token, and returns
- For each nonterminal `N`, create a function `parse_N`
 - Let $N \rightarrow \beta_1 \mid \dots \mid \beta_k$ be the productions of `N`
 - For each production $N \rightarrow \beta_i$ (where $\beta_i = \alpha_1 \dots \alpha_n$)
 - If the lookahead is in `First(β_i)`
 - Call `parse_ α_1 ()`; ... ; `parse_ α_n ()`

CMSC 330

2

First Sets

- Definition
 - `First(γ)`, for any terminal or nonterminal `γ` , is the set of initial terminals of all strings that `γ` may expand to
- For a terminal `a`
 - `First(a) = { a }`
- For a nonterminal `N`
 - If $N \rightarrow \epsilon$, then add `ϵ` to `First(N)`
 - If $N \rightarrow \alpha_1 \alpha_2 \dots \alpha_n$, then add
 - `First(α_1)` if $\epsilon \notin \text{First}(\alpha_1)$
 - Else add `{ First(α_1) - ϵ } \cup First($\alpha_2 \dots \alpha_n$)`

CMSC 330

3

Left Recursion Elimination Algorithm

- Given grammar
 - $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta$
- Rewrite grammar as
 - $A \rightarrow \beta L$
 - $L \rightarrow \alpha_1 L \mid \alpha_2 L \mid \dots \mid \alpha_n L \mid \epsilon$
- Repeat as necessary

CMSC 330

4

Left Factoring Algorithm

- Given grammar
 - $A \rightarrow x\alpha_1 \mid x\alpha_2 \mid \dots \mid x\alpha_n \mid \beta$
- Rewrite grammar as
 - $A \rightarrow xL \mid \beta$
 - $L \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$
- Repeat as necessary

CMSC 330

5

Example 1 – Left Factoring

Consider the grammar

```
S → T + S | T
T → U * T | U
U → (S) | V
V → 0 | 1 | ... | 9
```

Common prefix requires applying left factoring

```
S → T + S | T
T → U * T | U
```

Left factor S

```
S → T + S | T
↓
S → T L
L → + S |  $\epsilon$ 
```

Left factor T

```
T → U * T | U
↓
T → U M
M → * T |  $\epsilon$ 
```

CMSC 330

6

Example 1 – Computing First Sets

Grammar
 $S \rightarrow TL$
 $L \rightarrow +S \mid \epsilon$
 $T \rightarrow UM$
 $M \rightarrow *T \mid \epsilon$
 $U \rightarrow (S) \mid V$
 $V \rightarrow 0 \mid 1 \mid \dots \mid 9$

Compute First Sets
 $First(V) = \{0..9\}$
 $First(U) = First((S)) \cup First(V)$
 $= \{(\) \cup \{0..9\} = \{(\ , 0..9\}$
 $First(M) = First(*T) \cup First(\epsilon)$
 $= \{*\} \cup \{\epsilon\} = \{*, \epsilon\}$
 $First(T) = First(UM) = First(U)$
 $= \{(\ , 0..9\}$
 $First(L) = First(+S) \cup First(\epsilon)$
 $= \{+\} \cup \{\epsilon\} = \{+, \epsilon\}$
 $First(S) = First(TL) = First(T)$
 $= \{(\ , 0..9\}$

CMSC 330

7

Example 1 – Recursive Descent Parser

Grammar
 $S \rightarrow TL$
 $L \rightarrow +S \mid \epsilon$
 $T \rightarrow UM$
 $M \rightarrow *T \mid \epsilon$
 $U \rightarrow (S) \mid V$
 $V \rightarrow 0 \mid 1 \mid \dots \mid 9$

Recursive descent parser
`parse_S() {
 // S → TL
 parse_T(); parse_L();
}`

```

parse_L() {
  // L → + S
  if (lookahead == "+") {
    match("+"); parse_S();
  }
  // L → ε
  else ;
}

parse_T() {
  // T → UM
  parse_U(); parse_M();
}

```

CMSC 330

8

Example 1 – Recursive Descent Parser

Grammar
 $S \rightarrow TL$
 $L \rightarrow +S \mid \epsilon$
 $T \rightarrow UM$
 $M \rightarrow *T \mid \epsilon$
 $U \rightarrow (S) \mid V$
 $V \rightarrow 0 \mid 1 \mid \dots \mid 9$

Recursive descent parser
`parse_M() {
 // M → * T
 if (lookahead == "*") {
 match("*"); parse_T();
 }
 else ; // M → ε
}`

```

parse_U() {
  if (lookahead == "(") { // U → (S)
    match("("); parse_S(); match(")");
  }
  else parse_V(); // U → V
}

parse_V() {
  if (lookahead == "0") { // V → 0
    match("0");
  } else if (lookahead == "1") { // V → 1
    match("1");
  } else
    ...
  } else if (lookahead == "9") { // V → 9
    match("9");
  } else error();
}

```

CMSC 330

9

Example 1 – Parsing Input

Grammar
 $S \rightarrow TL$
 $L \rightarrow +S \mid \epsilon$
 $T \rightarrow UM$
 $M \rightarrow *T \mid \epsilon$
 $U \rightarrow (S) \mid V$
 $V \rightarrow 0 \mid 1 \mid \dots \mid 9$

Lookahead in red
 Parse "1+2"

Parse	Remaining
parse_S()	"1+2"
parse_T()	"1+2"
parse_U()	"1+2"
parse_V()	"1+2"
match("1")	"1+2"
parse_M()	"+2"
parse_L()	"+2"
match("+")	"+2"
parse_S()	"2"
parse_T()	"2"
parse_U()	"2"
parse_V()	"2"
match("2")	"2"
==	""

CMSC 330

10

Example 2 – Computing First Sets

Consider the grammar
 $S \rightarrow ABd \mid aBc$
 $A \rightarrow \epsilon$
 $B \rightarrow b \mid c$

Compute First Sets

$First(b) = \{b\}$
 $First(c) = \{c\}$
 $First(B) = First(b) \cup First(c)$
 $= \{b, c\}$

$First(\epsilon) = \{\epsilon\}$
 $First(A) = First(\epsilon) = \{\epsilon\}$
 $First(ABd) = \{First(A) - \epsilon\} \cup First(Bd)$
 $= \{\epsilon - \epsilon\} \cup First(Bd)$
 $= First(Bd) = First(B) = \{b, c\}$
 $First(aBc) = \{a\}$
 $First(S) = First(ABd) \cup First(aBc)$
 $= \{b, c\} \cup \{a\}$
 $= \{a, b, c\}$

CMSC 330

11

Example 2 – Using First Sets

Grammar
 $S \rightarrow ABd \mid aBc$
 $A \rightarrow \epsilon$
 $B \rightarrow b \mid c$

First sets for RHS

$First(b) = \{b\}$
 $First(c) = \{c\}$
 $First(ABd) = \{b, c\}$
 $First(aBc) = \{a\}$

Grammar is predictive if...

$First(ABd) \cap First(aBc) = \emptyset$
 $First(b) \cap First(c) = \emptyset$
 And not left recursive...

Verify...

$First(ABd) \cap First(aBc)$
 $= \{b, c\} \cap \{a\}$
 $= \emptyset$
 $First(b) \cap First(c)$
 $= \{b\} \cap \{c\}$
 $= \emptyset$

No overlap, grammar is predictive

CMSC 330

12

Example 2 – Recursive Descent Parser

Grammar

```
S → ABd | aBc
A → ε
B → b | c
```

```
parse_A() { return; } // A → ε
```

```
parse_S() {
  // S → ABd
  if ((lookahead == "b" ||
      lookahead == "c")) {
    parse_A(); parse_B(); match("d");
  }
  // S → aBc
  else if (lookahead == "a") {
    match("a"); parse_B(); match("c");
  }
  else error();
}
```

Recursive descent parser

```
parse_B() {
  if (lookahead == "b")
    match("b"); // B → b
  else if (lookahead == "c")
    match("c"); // B → c
  else error();
}
```

CMSC 330

13

Example 2 – Parsing Input

Grammar

```
S → ABd | aBc
A → ε
B → b | c
```

Lookahead in red

Parse "bd"

parse_S()	"bd"
parse_A()	"bd"
parse_B()	"bd"
match("b")	"bd"
match("d")	"d"

Remaining

Parse "acc"

parse_S()	"acc"
match("a")	"acc"
parse_B()	"cc"
match("c")	"cc"
match("c")	"c"

Remaining

CMSC 330

14