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 $N \rightarrow \beta_1 \mid ... \mid \beta_k$ be the productions of N
  - For each production $N \rightarrow \beta_i$ (where $\beta_i = \alpha_1 \ ... \ \alpha_n$)
  - If the lookahead is in First(β_i)
    - Call parse_α_1( ); ... ; parse_α_n( )

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 → ε, then add ε to First(N)
  - If N → α_1 α_2 ... α_n, then add
    - First(α_1) if ε∉ First(α_1)
    - Else add { First(α_1) – ε} ∪ First(α_2 ... α_n)

Example 1 – Computing First Sets

Consider the grammar

S → ABd | aBc

A → ε

B → b | c

Compute First Sets

First(b) = {b}

First(c) = {c}

First(B) = First(b) ∪ First(c)

= {b,c}

First(ε) = {ε}

First(A) = First(ε) = {ε}

First(ABd) = {First(A) – ε} ∪ First(Bd)

= {ε– ε} ∪ First(Bd)

= First(Bd) = First(B) = {b, c}

First(aBc) = { a }

First(S) = First(ABd) ∪ First(aBc)

= {b, c} ∪ {a}

= {a,b,c}

Example 1 – Using First Sets

Grammar
- S → ABd | aBc
- A → ε
- B → b | c

First sets for RHS
- First(b) = {b}
- First(c) = {c}
- First(ABd) = {b, c}
- First(aBc) = { a }

Grammar is predictive if…
- First(ABd) ∩ First(aBc) = ∅
- First(b) ∩ First(c) = ∅
- And not left recursive…

Verify…
- First(ABd) ∩ First(aBc)
  - = {b, c} ∩{ a }
  - = ∅
- First(b) ∩ First(c)
  - = {b} ∩{c}
  - = ∅

No overlap, grammar is predictive

Example 1 – Recursive Descent Parser

Grammar
- S → ABd | aBc
- A → ε
- B → b | c

Recursive descent parser

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

```
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( );
}
```

Example 1 – Parsing Input

Grammar

S → ABd | aBc
A → ε
B → b | c

Lookahead in red

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

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

Example 2 – Computing First Sets

Consider the grammar

S → (L) | a
L → L,S | S

Compute First Sets

First(S) = ∅ for now
First(L) = ∅ for now
First((L)) = { (}
First(a) = { a }

First(S) = First((L)) ∪ First(a)
= { (} ∪ {a} = {(,a}
First(L) = First(L,S) ∪ First(S)
= First(L) ∪ First(S)
= ∅ ∪ First(S) for now
= { (,a } for now
First(L,S) = First(L)
= { (,a } for now
First(L) = First(L,S) ∪ First(S)
= First(L) ∪ First(S)
= { (,a } ∪ { (,a } = {(,a}
First(L,S) = First(L)
= { (,a }

Example 2 – Using First Sets

Grammar

S → (L) | a

L → L,S | S

First sets for RHS

First(a) = {a}

First((L)) = { (}

First(S) = { (,a }

First(L,S) = { (,a }

Grammar is predictive if…

First((L)) ∩ First(a) = ∅

First(L,S) ∩ First(S) = ∅

And not left recursive

Verify…

First((L)) ∩ First(a)

= { (} ∩{ a }

= ∅

First(L,S) ∩ First(S)

= { (,a } ∩ { (,a }

= { (,a }

Overlap! Grammar is not predictive

Example 2 – Rewriting the Grammar

Grammar

S → (L) | a

L → L,S | S

c) Can the grammar be parsed by a backtracking parser?

No, due to left recursion,

L → L,S

d) Rewrite grammar using the rule for eliminating left recursion

S → (L) | a

L → L,S | S

↓

S → (L) | a

L → S M

M → , S M | ε

Example 2 – Recomputing First Sets

Grammar

S → (L) | a

L → S M

M → , S M | ε

e) Compute First Sets

First(S) = { (, a }

First(L) = { (, a }

First(M) = {ε, , }

First(a) = { a }

f) Grammar is predictive if…

First((L)) ∩ First(a) = ∅

First(, S M) ∩ First(ε) = ∅

And not left recursive

Verify…

First((L)) ∩ First(a)

= { (} ∩{ a }

= ∅

First(, S M) ∩ First(ε)

= { , } ∩ {ε}

= ∅

No overlap, grammar is predictive

Example 2 – Recursive Descent Parser

Grammar

S → (L) | a

L → S M

M → , S M | ε

g) Parser

```
parse_S( ) {
    if (lookahead == "(") {
        match("("); parse_L();
        match(")");   // S → (L)
    }
    else if (lookahead == "a")
        match("a");   // S → a
    else error( );
}
```

```
parse_L( ) {
    if ((lookahead == "(")  ||
        (lookahead == "a")) {
        parse_S();    // L → S M
        parse_M();
    }
}
parse_M( ) {
    if (lookahead == ",") {
        match(","); parse_S();
        parse_M();    // M → , S M
    }
    else return;      // M → ε
}
```

Example 2 – Parsing Input

Grammar

S → (L) | a
L → S M
M → , S M | ε

Lookahead in red

Parse "(a,a)"	Remaining
parse_S( )	"(a,a)"
match("(")	"(a,a)"
parse_L( )	"a,a)"
parse_S( )	"a,a)"
match("a")	"a,a)"
parse_M( )	",a)"
match(",")	",a)"
parse_S( )	"a)"
match("a")	"a)"
parse_M( )	")"
match(")")	")"
	""

Example 3 – Computing First Sets

Consider the grammar

E → E + T | T
T → a | (E)

Compute First Sets

First(E) = ∅ for now
First(T) = ∅ for now
First(a) = { a }
First((E)) = { (}

First(T) = First(a) ∪ First((E))
= { a } ∪ { (} = { (, a }
First(E) = First(E+T) ∪ First(T)
= First(E) ∪ First(T)
= ∅ ∪ First(T) for now
= { (, a } for now
First(E+T) = First(E)
= { (, a } for now
First(E) = First(E+T) ∪ First(T)
= { (, a } ∪ { (, a } = { (, a }

Example 3 – Rewriting the Grammar

Grammar

E → E + T | T

T → a | (E)

b) Is the grammar ambiguous?

No. Unique left-most derivations.

c) Can the grammar be parsed by a predictive parser?

No, since First((E+T)) ∩ First(T) = { (, a }

d) Can the grammar be parsed by a backtracking parser?

No, due to left recursion, where E → E + T

e) Rewrite the grammar using the rule for eliminating left recursion

E → E + T | T

T → a | (E)

↓

E → T L

L → + T L | ε

T → a | (E)

Example 3 – Recomputing First Sets

Consider the grammar

E → T L

L → + T L | ε

T → a | (E)

f) Compute First Sets

First(a) = { a }

First((E)) = { (}

First(T) = First(a) ∪ First((E))
= { a } ∪ { (} = { (, a }

First(E) = First(T L)
= First(T) = { (, a }

First(+ T L) = {+ }

First(ε) = {ε }

First(L) = First(+ T L) ∪ First(ε)
= = {+} ∪ {ε } = {+, ε }

Example 3 – Using First Sets

Consider the grammar

E → T L

L → + T L | ε

T → a | (E)

First(a) = { a }

First((E)) = { (}

First(+ T L) = {+ }

First(ε) = {ε }

g) Grammar is predictive if…

First(+ T L) ∩ First(ε) = ∅

First(a) ∩ First((E)) = ∅

And not left recursive

Verify…

First(+ T L) ∩ First(ε)

= { + } ∩ {ε}

= ∅

First(a) ∩ First((E))

= { a } ∩{ (}

= ∅

No overlap, grammar is predictive

Example 3 – Recursive Descent Parser

Grammar

E → T L

L → + T L | ε

T → a | (E)

Recursive descent parser

```
parse_E( ) {
    // E → TL
    if ((lookahead == "(") ||
        (lookahead == "a")) {
        parse_T( ); parse_L( );
    }
    else error( );
}
```

```
parse_L( ) {
    if (lookahead == "+") {    // L → +TL
        match("+"); parse_T( ); parse_L( );
    }
    else ;                     // L → ε
}
parse_T( ) {
    if (lookahead == "a")      // T → a
        match("a");
    else if (lookahead == "(") { // T → (E)
        match("("); parse_E( ); match(")");
    }
    else error( );
}
```

Example 3 – Parsing Input

Grammar

E → T L

L → + T L | ε

T → a | (E)

Lookahead in red

Parse "a+a+a"	Remaining
parse_E( )	"a+a+a"
parse_T( )	"a+a+a"
match("a")	"a+a+a"
parse_L( )	"+a+a"
match("+")	"+a+a"
parse_T( )	"a+a"
match("a")	"a+a"
parse_L( )	"+a"
match("+")	"+a"
parse_T( )	"a"
match("a")	"a"
parse_L( )	""
	""

Example 3 – A Slightly Different Grammar

Consider the grammar

E → T + E | T

T → a | (E)

Vs. previous grammar

E → E + T | T

T → a | (E)

a) Can the grammar be parsed by a predictive parser?

No, since First(T+E) ∩ First(T) ≠ ∅

b) Would grammar accept same language?

Yes – all sums of a's

c) What is the difference between this grammar and the previous grammar rewritten to eliminate left recursion?

This grammar is right associative

Previous grammar is left associative