CMSC 330: Organization of Programming Languages

Parser Examples 2

Example 3 – Computing First Sets

Consider the grammar
S → AS | b
A → SA | a

Compute First Sets
First(S) = ∅ for now
First(A) = ∅ for now
First(b) = {b}
First(a) = {a}
First(A) = First(SA) ∪ First(a)
= First(S) ∪ {a}
= {a} for now

First(S) = First(AS) ∪ First(b)
= First(AS) ∪ {b}
= First(A) ∪ {b}
= {a} ∪ {b} = {a,b}
First(SA) = First(S)
= {a,b}
First(A) = First(SA) ∪ First(a)
= First(S) ∪ {a}
= {a,b} ∪ {b} = {a,b}
First(S) = First(AS) ∪ First(b)
= First(AS) ∪ {b}
= First(A) ∪ {b}
= {a,b} ∪ {b} = {a,b}

Example 3 – Using First Sets

Grammar
S → AS | b
A → SA | a

First sets for RHS
First(a) = {a}
First(b) = {b}

First(A) = {a,b}
First(S) = {a,b}

Grammar is predictive if…
First(AS) ∩ First(b) = ∅
First(SA) ∩ First(a) = ∅
And not left recursive

Verify…
First(AS) ∩ First(b)
= {a, b} ∩ { b }
= { b }
First(SA) ∩ First(a)
= {a, b} ∩ { a }
= { a }

Overlap! Grammar is <u>not</u> predictive

Example 3 – Ambiguous Grammars

Grammar
S → AS | b
A → SA | a

c) Can the grammar be parsed by a backtracking parser?
No, due to (mutual) left recursion,
where S → AS derives
S → <u>SAS</u>

d) Is the grammar ambiguous?
Yes. 2 left-most derivations of "abab"

e) Are all ambiguous grammars non-parseable by predictive parsers?
Yes. RHS guaranteed to conflict

f) Are all non-ambiguous grammars parseable by predictive parsers?
No. Consider S → aab | aac

Example 4 – 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 4 – 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 <u>not</u> predictive

Example 4 – 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 4 – 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 4 – 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 4 – 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 5 – 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 5 – 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 5 – 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 5 – 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 5 – 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 5 – 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 5 – 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