

Predictive Parsing

Roadmap (Where are we?)

We set out to study parsing

- Specifying syntax
 - Context-free grammars
 - Ambiguity
- (Back Tracking) Top-down parsers
 - Algorithm & its problem with left recursion
 - Left-recursion removal
- Predictive top-down parsing ←
 - FIRST, FOLLOW, FIRST*
 - The LL(1) condition
 - Table-driven LL(1) parsers
 - Recursive descent parsers
 - Left factoring

CS430

Lecture 4

2

Predictive Parsing

Basic idea

Given $A \rightarrow \alpha \mid \beta$, the parser should be able to choose between α & β

FIRST sets

For some rhs $\alpha \in \mathcal{G}$, define $\text{FIRST}(\alpha)$ as the set of tokens that appear as the first symbol in some string that derives from α

That is, $a \in \text{FIRST}(\alpha)$ iff $\alpha \Rightarrow^* a \gamma$, for some γ

The LL(1) Property

If $A \rightarrow \alpha$ and $A \rightarrow \beta$ both appear in the grammar, we would like

$$\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$$

This would allow the parser to make a correct choice with a lookahead of exactly one symbol!

This is almost correct
Will also need FOLLOW

CS430

Lecture 4

3

The FIRST Set

$$a \in \text{FIRST}(\alpha) \text{ iff } \alpha \Rightarrow^* a \gamma, \text{ for some } \gamma$$

To build $\text{FIRST}(X)$ for all grammar symbols X :

1. if X is a terminal (token), $\text{FIRST}(X) := \{X\}$
2. if $X ::= \epsilon$, then $\epsilon \in \text{FIRST}(X)$
3. iterate until no more terminals or ϵ can be added to any $\text{FIRST}(X)$:
if $X ::= Y_1 Y_2 \dots Y_k$ then
 $a \in \text{FIRST}(X)$ if $a \in \text{FIRST}(Y_i)$ and
 $\epsilon \in \text{FIRST}(Y_j)$ for all $1 \leq j < i$
 $\epsilon \in \text{FIRST}(X)$ if $\epsilon \in \text{FIRST}(Y_i)$ for all $1 \leq i \leq k$
end iterate

Note: if $\epsilon \in \text{FIRST}(Y_i)$, then $\text{FIRST}(Y_i)$ is irrelevant, for $1 < i$

CS430

Lecture 4

4

The FIRST Set

$$a \in \text{FIRST}(\alpha) \text{ iff } \alpha \Rightarrow^* \underline{a} \gamma, \text{ for some } \gamma$$

To build $\text{FIRST}(\alpha)$ for $\alpha = X_1 X_2 \dots X_n$:

1. $x \in \text{FIRST}(\alpha)$ if $x \in \text{FIRST}(X_i)$ and
 $\epsilon \in \text{FIRST}(X_j)$ for all $1 \leq j < i$
2. $\epsilon \in \text{FIRST}(\alpha)$ if $\epsilon \in \text{FIRST}(X_i)$ for all $1 \leq i \leq n$

CS430

Lecture 4

5

The FOLLOW Set

For a non-terminal A , define $\text{FOLLOW}(A)$ as

$\text{FOLLOW}(A) :=$ the set of terminals that can appear immediately to the right of A in some sentential form.

Thus, a non-terminal's FOLLOW set specifies the tokens that can legally appear after it; a terminal has no FOLLOW set

CS430

Lecture 4

6

The FOLLOW Set

To build FOLLOW(X) for all non-terminal X:

1. Place \$ in FOLLOW(<goal>) // \$ = EOF
 iterate until no more terminals or ε can be added to any FOLLOW(X):
2. If $A \rightarrow \alpha B \beta$ then
 put $\{FIRST(\beta) - \epsilon\}$ in FOLLOW(B)
3. If $A \rightarrow \alpha B$ then
 put FOLLOW(A) in FOLLOW(B)
4. If $A \rightarrow \alpha B \beta$ and $\epsilon \in FIRST(\beta)$ then
 put FOLLOW(A) in FOLLOW(B)

CS430

Lecture 4

7

Predictive Parsing

If $A \rightarrow \alpha$ and $A \rightarrow \beta$ and $\epsilon \in FIRST(\alpha)$, then we need to ensure that $FIRST(\beta)$ is disjoint from FOLLOW(A), too

Define $FIRST^*(\delta)$ for rule $A \rightarrow \delta$ as

- $FIRST(\delta) \cup FOLLOW(A)$, if $\epsilon \in FIRST(\delta)$
- $FIRST(\delta)$, otherwise

CS430

Lecture 4

8

Predictive Parsing

The LL(1) Property

A grammar is LL(1) iff $A \rightarrow \alpha$ and $A \rightarrow \beta$ implies $FIRST^*(\alpha) \cap FIRST^*(\beta) = \emptyset$

This would allow the parser to make a correct choice with a lookahead of exactly one symbol !

Question: Can there be two rules $A \rightarrow \alpha$ and $A \rightarrow \beta$ in a LL(1) grammar such that $\epsilon \in FIRST(\alpha)$ and $\epsilon \in FIRST(\beta)$?

CS430

Lecture 4

9

Recall Top-down Parsing

A top-down parser starts with the root of the parse tree
 The root node is labeled with the goal symbol of the grammar

Top-down parsing algorithm

Construct the root node of the parse tree

Repeat until the fringe of the parse tree matches the input string

- 1 At a node labeled A, select a production with A on its lhs and, for each symbol on its rhs, construct the appropriate child
- 2 When a terminal symbol is added to the fringe and it doesn't match the fringe, backtrack
- 3 Find the next node to be expanded (label ∈ NT)

- For LL(1) grammars

- we can use $FIRST^*$ to pick the right production in step 1
- Using 1 token of lookahead

CS430

Lecture 4

10

Predictive Parsing

Given a grammar that has the LL(1) property

- Problem: NT A needs to be replaced in next derivation step
- Assume $A \rightarrow \beta_1 \mid \beta_2 \mid \beta_3$, with $FIRST^*(\beta_1) \cap FIRST^*(\beta_2) \cap FIRST^*(\beta_3) = \emptyset$

```
/* find rule for A */
if (current token ∈ FIRST+(β1))
    select A → β1
else if (current token ∈ FIRST+(β2))
    select A → β2
else if (current token ∈ FIRST+(β3))
    select A → β3
else
    report an error and return false
```

Grammars with the LL(1) property are called *predictive grammars* because the parser can "predict" the correct expansion at each point in the parse.

Parsers that capitalize on the LL(1) property are called *predictive parsers*.

One kind of predictive parser is the *recursive descent parser*. The other is a *table-driven parser*.

CS430

Lecture 4

11

LL(1) Parser Example

Is the following grammar LL(1)?

$S ::= a S b \mid \epsilon$

$FIRST(aSb) = \{ a \}$
 $FIRST(\epsilon) = \{ \epsilon \}$

$FIRST^*(aSb) = \{ a \}$
 $FIRST^*(\epsilon) = (FIRST(\epsilon) - \{ \epsilon \}) \cup FOLLOW(S) = \{ \$, b \}$

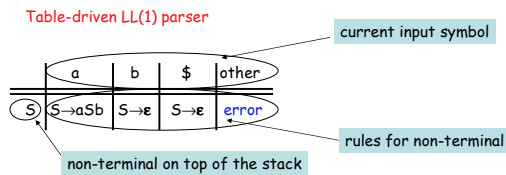
LL(1)? YES, since $\{ a \} \cap \{ \$, b \} = \emptyset$

CS430

Lecture 4

12

LL(1) Parser Example



CS430

Lecture 4

13

Building Table-driven Top Down Parsers

Building the complete table

- Need a row for every NT & a column for every T
- Need an algorithm to build the table

Filling in $TABLE[X,y]$, $X \in NT, y \in T$

- entry is the rule $X ::= \beta$, if $y \in FIRST^+(\beta)$
- entry is error otherwise (can treat empty entry as implicit error)

If any entry is defined multiple times, G is not LL(1)

This is the LL(1) table construction algorithm

CS430

Lecture 4

14

LL(1) Skeleton Parser

```

token ← next_token()
push $ onto Stack // $ used to mark EOF
push the start symbol, S, onto Stack
TOS ← top of Stack
loop forever
  if TOS = $ and token = $ then
    break & report success (accept)
  else if TOS is a terminal then
    if TOS matches token then
      pop Stack // recognized TOS
      token ← next_token()
    else report error looking for TOS
  else // TOS is a non-terminal
    if TABLE[TOS,token] is A → B1B2...Bn then
      pop Stack // get rid of A
      push Bn, Bn-1, ..., B1 // in that order
    else report error expanding TOS
TOS ← top of Stack
    
```

exit on success

CS430

Lecture 4

15

Table-driven LL(1) Parser Example

	a	b	\$	other
S	S → aSb	S → ε	S → ε	error

How to parse input a a b b b ?

Describe action as sequence of states

(PDA stack content, remaining input, next action)

PDA stack content: [X, ... Z], where Z is the TOS

next actions: rule or next input+pop or error or accept

CS430

Lecture 4

16

Table-driven LL(1) Parser Example

	a	b	\$	other
S	S → aSb	S → ε	S → ε	error

For S → aSb
 1. Pop S
 2. Push b, push S, push a
 (i.e., in reverse order)

Stack

```

[$, S],
[$, b, S, a],
[$, b, S],
[$, b, b, S, a],
[$, b, b, S],
[$, b, b, b, S, a],
[$, b, b, b, S],
[$, b, b, b, b],
[$, b, b, b],
[$, b, b],
[$, b],
[$],
    
```

Remaining Input

```

aaabbb$,
aaabbb$,
aabbb$,
aabbb$,
abbb$,
abbb$,
bbb$,
bbb$,
bb$,
bb$,
b$,
$,
    
```

Action

```

S → aSb
next input+pop
S → aSb
next input+pop
S → aSB
next input+pop
S → ε
next input+pop
next input+pop
next input+pop
accept
    
```

CS430

Lecture 4

17

Recursive Descent Parsers

- Description

→ Top-down parser built from a set of mutually-recursive procedures
 → Each procedure usually implements a nonterminal from the grammar

- Implementation

→ Backtracking

- Choose different production if current choice fails

→ LL(1) Predictive

- Compare lookahead token to $FIRST^+$ sets to select production

→ Utility function

```

match(t) {
  if (lookahead == t)
    lookahead ← next_token();
  else error();
}
    
```

CS430

Lecture 4

18

Recursive Descent LL(1) Parser Implementation

- Terminals
 - Terminals in the input stream appear as token lookahead
 - Can advance lookahead token using: `lookahead ← next_token()`
- Nonterminals
 - Every NT is associated with a parsing procedure
 - The parsing procedure for $A \in NT$, `proc A`
 - Responsible for parsing and consuming any string that can be derived from A
 - Choose to replace A with β for production $A \rightarrow \beta$
 - If `lookahead` $\in \text{FIRST}^*(\beta)$
 - Error if lookahead token does not match any production
- Parser is invoked by calling `proc S` for start symbol S

CS430

Lecture 4

19

Recursive Descent Parser Example 1

- Given grammar $S \rightarrow xyz \mid abc$
 - $\text{FIRST}^*(xyz) = \{x\}$, $\text{FIRST}^*(abc) = \{a\}$
- Parser


```

parse_S(){
  if (lookahead == "x"){
    match("x"); match("y"); match("z"); // S → xyz
  }
  else if (lookahead == "a"){
    match("a"); match("b"); match("c"); // S → abc
  }
  else error();
}
            
```

CS430

Lecture 4

20

Recursive Descent Parser Example 2

- Given grammar $S \rightarrow A \mid B$ $A \rightarrow x \mid y$ $B \rightarrow z$
 - $\text{FIRST}^*(A) = \{x, y\}$, $\text{FIRST}^*(B) = \{z\}$
- Parser


```

parse_S(){
  if ((lookahead == "x") ||
      (lookahead == "y"))
    parse_A(); // S → A
  else if (lookahead == "z")
    parse_B(); // S → B
  else error();
}

parse_A(){
  if (lookahead == "x")
    match("x"); // A → x
  else if (lookahead == "y")
    match("y"); // A → y
  else error();
}

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

CS430

Lecture 4

21

Recursive Descent Parser Example 3

- Given grammar $S \rightarrow aSb \mid \epsilon$
 - $\text{FIRST}^*(aSb) = \{a\}$, $\text{FIRST}^*(\epsilon) = \text{Follow}(S) = \{b, \$\}$
- Parser


```

parse_S(){
  if (lookahead == "a"){
    match("a"); parse_S(); match("b"); // S → aSb
  } else if ((lookahead == "b") || (lookahead == "$"))
    ; // S → ε
  else error();
}
            
```

CS430

Lecture 4

22

Left Factoring

What if my grammar does not have the LL(1) property?
 ⇒ Sometimes, we can transform the grammar

The Algorithm

$\forall A \in NT$,
 find the longest prefix α that occurs in two or more right-hand sides of A
 if $\alpha \neq \epsilon$ then replace all of the A productions,
 $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma$,
 with
 $A \rightarrow \alpha Z \mid \gamma$
 $Z \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$
 where Z is a new element of NT
 Repeat until no common prefixes remain

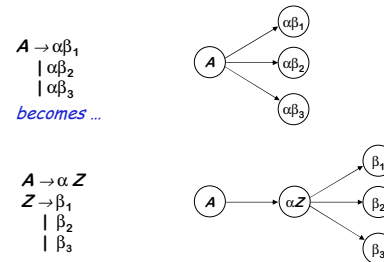
CS430

Lecture 4

23

Left Factoring

A graphical explanation for the same idea



CS430

Lecture 4

24

Left Factoring (An example)

Consider the following fragment of the expression grammar

Factor \rightarrow Identifier
 | Identifier [ExprList]
 | Identifier (ExprList)

FIRST(rhs_1) = { Identifier }
 FIRST(rhs_2) = { Identifier }
 FIRST(rhs_3) = { Identifier }

After left factoring, it becomes

Factor \rightarrow Identifier Arguments
 Arguments \rightarrow [ExprList]
 | (ExprList)
 | ϵ

FIRST(rhs_1) = { Identifier }
 FIRST(rhs_2) = { [}
 FIRST(rhs_3) = { (}
 FIRST(rhs_4) = FOLLOW(Factor)
 \Rightarrow It has the LL(1) property

This form has the same syntax, with the LL(1) property

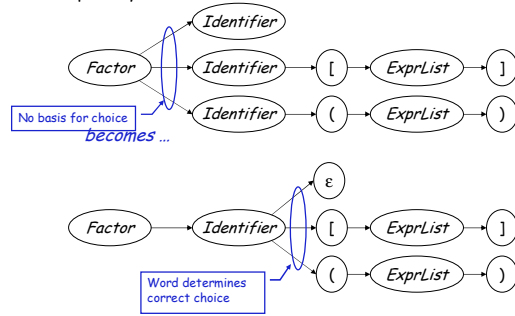
CS430

Lecture 4

25

Left Factoring

Graphically



CS430

Lecture 4

26

LL(1) Example - Grammar & Left Factoring

Original Grammar

Goal \rightarrow Expr
 Expr \rightarrow Term + Expr
 | Term - Expr
 | Term
 Term \rightarrow Factor * Term
 | Factor / Term
 | Factor
 Factor \rightarrow num
 | id

Left Factored Grammar

Goal \rightarrow Expr
 Expr \rightarrow Term Expr'
 Expr' \rightarrow + Expr'
 | - Expr'
 | ϵ
 Term \rightarrow Factor Term'
 Term' \rightarrow * Term
 | / Term
 | ϵ
 Factor \rightarrow num
 | id

CS430

Lecture 4

27

LL(1) Example - First Sets

Grammar

Goal \rightarrow Expr { num, id }
 Expr \rightarrow Term Expr' { num, id }
 Expr' \rightarrow + Expr' { + }
 | - Expr' { - }
 | ϵ { ϵ }
 Term \rightarrow Factor Term' { num, id }
 Term' \rightarrow * Term { * }
 | / Term { / }
 | ϵ { ϵ }
 Factor \rightarrow num { num }
 | id { id }

FIRST Sets

Production Nonterminals

Goal	{ num, id }
Expr	{ num, id }
Expr'	{ +, -, ϵ }
Term	{ num, id }
Term'	{ *, /, ϵ }
Factor	{ num, id }

1. if X is a token, FIRST(X) := { X }
 2. if $X ::= \epsilon$, then $\epsilon \in \text{FIRST}(X)$
 3. if $X ::= Y_1 Y_2 \dots Y_k$ then
 $a \in \text{FIRST}(X)$ if $a \in \text{FIRST}(Y_i)$...

LL(1) Example - Follow Sets

Grammar

Goal \rightarrow Expr
 Expr \rightarrow Term Expr'
 Expr' \rightarrow + Expr'
 | - Expr'
 | ϵ
 Term \rightarrow Factor Term'
 Term' \rightarrow * Term
 | / Term
 | ϵ
 Factor \rightarrow num
 | id

FIRST Sets FOLLOW Sets

Goal	{ num, id }	{ \$ }
Expr	{ num, id }	{ \$ }
Expr'	{ +, -, ϵ }	{ \$ }
Term	{ num, id }	{ +, -, \$ }
Term'	{ *, /, ϵ }	{ +, -, \$ }
Factor	{ num, id }	{ *, /, +, -, \$ }

1. Place \$ in FOLLOW(*goal*)
 2. If $A \rightarrow \alpha\beta$ then
 put $\text{FIRST}(\beta) - \epsilon$ in FOLLOW(A)
 3. If $A \rightarrow \alpha\beta$ then
 put FOLLOW(A) in FOLLOW(B)
 4. If $A \rightarrow \alpha\beta$ and $\epsilon \in \text{FIRST}(\beta)$ then
 put FOLLOW(A) in FOLLOW(B)

CS430

Lecture 4

29

LL(1) Example - LL(1) Table

Grammar

Goal \rightarrow Expr { num, id }
 Expr \rightarrow Term Expr' { num, id }
 Expr' \rightarrow + Expr' { + }
 | - Expr' { - }
 | ϵ { ϵ }
 Term \rightarrow Factor Term' { num, id }
 Term' \rightarrow * Term { * }
 | / Term { / }
 | ϵ { ϵ }
 Factor \rightarrow num { num }
 | id { id }

FIRST Sets FOLLOW Sets

Goal	{ num, id }	{ \$ }
Expr	{ num, id }	{ \$ }
Expr'	{ +, -, ϵ }	{ \$ }
Term	{ num, id }	{ +, -, \$ }
Term'	{ *, /, ϵ }	{ +, -, \$ }
Factor	{ num, id }	{ *, /, +, -, \$ }

For TABLE[X,y], X \in NT, y \in T
 entry is the rule $X \rightarrow \beta$, if $y \in \text{FIRST}^+(\beta)$

	num	id	+	-	*	/	\$
Goal	Goal \rightarrow Expr	Goal \rightarrow Expr					
Expr	Expr \rightarrow Term Expr'	Expr \rightarrow Term Expr'					
Expr'	Expr' \rightarrow + Expr'	Expr' \rightarrow - Expr'	Expr' \rightarrow + Expr'	Expr' \rightarrow - Expr'			Expr' \rightarrow ϵ
Term	Term \rightarrow Factor Term'	Term \rightarrow Factor Term'					
Term'	Term' \rightarrow * Term	Term' \rightarrow / Term	Term' \rightarrow ϵ	Term' \rightarrow ϵ	Term' \rightarrow * Term	Term' \rightarrow / Term	Term' \rightarrow ϵ
Factor	Factor \rightarrow num	Factor \rightarrow id					

CS430

Lecture 4

30

Left Factoring (Generality)

Question

By eliminating left recursion and left factoring, can we transform an arbitrary CFG to a form where it meets the $LL(1)$ condition? (and can be parsed predictively with a single token lookahead?)

Answer

Given a CFG that doesn't meet the $LL(1)$ condition, it is undecidable whether or not an equivalent $LL(1)$ grammar exists.

Example

$\{a^n 0 b^n \mid n \geq 1\} \cup \{a^n 1 b^{2n} \mid n \geq 1\}$ has no $LL(1)$ grammar

CS430

Lecture 4

31

Language that Cannot Be $LL(1)$

Example

$\{a^n 0 b^n \mid n \geq 1\} \cup \{a^n 1 b^{2n} \mid n \geq 1\}$ has no $LL(1)$ grammar

$G \rightarrow \underline{a}Ab$
| $\underline{a}\beta b b$
 $A \rightarrow \underline{a}Ab$
| $\underline{0}$
 $B \rightarrow \underline{a}\beta b b$
| $\underline{1}$

Problem: need an unbounded number of \underline{a} characters before you can determine whether you are in the A group or the B group.

CS430

Lecture 4

32