Compiler Design

Today's Class Topics

Bottom Up Parsing
-Handle and Viable Prefixes
-Operator precedence parsing

Compiler Design

Handle Pruning: Reducing the string to start symbol using the handles

Handle: A Handle is a substring of a string that matched with any of the right side of the productions then That handle will be reduced with left side of the production.

Bottom-up parsing is a process of finding the handles And using them in the reductions to get the start symbol, This entire process of reducing the string to the start symbol is called as handle pruning

Viable prefix: Viable prefixes are the set of prefix of the right sentential form that can appear on the stack of a Shift Reduce Parser. i.e

Viable prefix are the prefixes of Right sentential form that do not extended beyond the end of its Handle.

Let $\alpha\beta$ is right sential form, where β is handle then viable prefixes are any prefix of $\alpha\beta$ except $\in$

NOTE:- The set of all viable prefixes of a CFL is a Regular language. i.e viable prefix can be recognise by Finite Automata

Compiler Design

Question:- Consider the following grammar and Parse the input string aab using Shift Reduce Parser

$G = \{ S \to AB, A \to aA \mid a, B \to bB \mid b \}$

LHS RHS

$S \to AB$ $B \to b$, $A \to aA$ $A \to a$

$S \Rightarrow AB \Rightarrow Ab \Rightarrow aAb \Rightarrow aab$

$\phi\beta$ $\phi\beta$ $\phi\beta$

$\phi\beta$	Viable prefix
aa	a, aa
aA	a, aA
Ab	A, Ab
AB	A, AB

Viable prefix = {a, aa, aA, A, Ab, AB}

Compiler Design

Question:- Consider the following grammar and Parse the input string aab using Shift Reduce Parser

$G = \{ S \to AB, A \to aA \mid a, B \to bB \mid b \}$

Bottom up Parsing

STACK	Input Buffer	Actions
$	aab $	Shift
$a	ab $	Shift
$aa	b $	Reduce by A → a
$aA	b $	Reduce by A → aA
$A	b $	Shift
$Ab	$	Reduce B → b
$AB	$	Reduce S → AB
$S	$	Accepted.

Viable prefix: $, $a, $aa, $aA, $A, $Ab, $AB, $S

Compiler Design

Question: Consider the following grammar, which of the following are viable Prefixes?

$G = \{ S \to AB,\ A \to aA \mid a,\ B \to bB \mid b \}$

1) aa 2) Aa 3) bb 4) bB 5) aB 6) AbB 7) ABb 8) AbbB

1) aa — viable prefix

Stack	I/p Buffer	Action
$	——— $	
$ a	——— $	A → a
$ aA	——— $	A → aA
$ A	b $	Shift
$ Ab	——— $	Reduce B → b
$ AB	——— $	Reduce S → AB
$ S		

2) Aa

Stack	I/p Buffer	Action
$	——— $	
$ Aa	——— $	Reduce A → a
$ AA	——— $	

→ Cannot reduce to S

∴ It is not viable prefix

Compiler Design

$G = \{ S \to AB,\ A \to aA \mid a,\ B \to bB \mid b \}$

3) Checking for bb

Stack	I/p Buffer	Action
$	—— $	
$b	—— $	B → b
$bB	—— $	B → bB
$B	—— $	

Cannot Reduce to S

∴ bb is not viable prefix

4) ab

Stack	I/p Buffer	Action
$	—— $	
$a	—— $	A → a
$aB		
$AB		

not possible

ab not viable prefix

Compiler Design

$G = \{ S \to AB,\ A \to aA \mid a,\ B \to bB \mid b \}$

5) AbB ✓

6) ABbX

7) AbbB

Stack	Ip Buffer	Action
$	$	
$AbB	$	B → bB
$AB	$	S → AB
$S	$	

→ Yes AbB is a viable prefix

Compiler Design

$\{S \to AB,\ A \to aA \mid a,\ B \to bB \mid b\}$

8) aaaaA

Stack	I/p Buffer	Action
$	$	
$aaaaA	$	A→aA
$aaaA	$	A→aA
$aaA	$	A→aA
$aA	$	A→aA
$A	b$	Shift
$Ab	$	B→b
$AB	$	S→AB
$S		

∴ aaaaA is a viable prefix

9) $aaaaab

$aaaaaB

↳ Cannot Reduce to S

∴ abb not viable prefix

Stack	I/p Buffer	Action
$aaaaab	$	B→b
$aaaaaB	$	

Compiler Design

Compiler Design

Compiler Design

Compiler Design

Compiler Design

Compiler Design

aCccccC

$aSb | — $ | C→b

$aSC

Question:- G = { S→CC, C→aC/b } viable prefixes ?

a) Cb b) aC c) aaC d) Ca e) abb

aC

However get StartSymbol S

∴ it is not viable prefix

Stack	I/p Buffer	Action
$	— $	
$ aC	— $	C→aC
$ C	— b $	Shift
$ Cb	→ $	C→b
$ CC	— S	S→CC
$ S		

→ aC ✓

Stack	I/p Buffer	Action
$ Ca	— $	
$ Ca	— b $	Shift
$ Cab	→ $	Reduce C→b
$ CaC	— $	Reduce C→aC
$ CC	— $	Reduce S→CC
$ S		

→ Ca is also viable prefix.

Compiler Design

Compiler Design

Compiler Design

Compiler Design

Compiler Design

$a \gtrdot b$

$a \lessdot b$

$a \doteq b$

Operator Precedence Parsing:

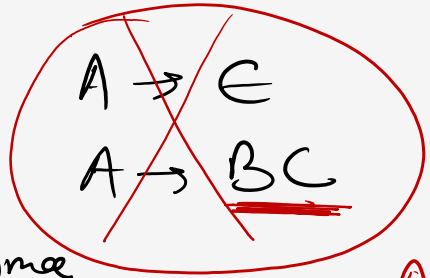
→ Applicable only for operator grammar

→ It parse Ambiguous Grammar also by using Precedence & Associativity Rules

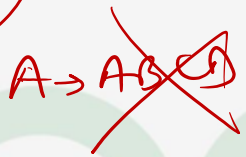
Operator Grammar:

i) No ϵ-productions ✗

ii) No Consecutive non Terminals ✗



Ex: $E \to EAE \mid id$, $A \to + \mid *$ } not operator Grammar



⇓ Convert to operator Grammar

$E \to E+E \mid E*E \mid id$

$A \to + \mid *$

operator Grammar

$a \gtrdot b$

$a \lessdot b$

$a \doteq b$

Compiler Design

Operator Precedence Passing Table :-

G: { E → E+E / E*E / id }

Apply Correct precedence and Associativity Rule

	id	+	*	$
id	—	>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	—

end marker

end marker

lol1 lol2

a b

lol lol

lol + lol

+ < lol

Compiler Design

Ambiguous Grammar

$G: \{ E \to E+E / E*E / id \}$

How to parse string id+id*id:

Let the topmost Symbol in the Stack is x and the present symbol in the input Buffer is y then Proceed as follows

1) if $x <\cdot y$ then Shift y into the STACK

2) if $x \cdot> y$ then there will be a Handle in the STACK then perform a reduce operation with an appropriate production

NOTE: Non Terminal have least precedence but greater than $

STACK	Input Buffer	Action
$ <·	id+id*id$	Shift
$ id ·>	+id*id$	Reduce E→id
$E <·	+id*id$	Shift
$E+ <·	id*id$	Shift
$E+id ·>	*id$	Reduce E→id
$E+E <·	*id$	Shift
$E+E* <·	id$	Shift
$E+E*id ·>	$	Reduce E→id
$E+E*E ·>	$	Reduce E→E*E
$E+E ·>	$	Reduce E→E+E
$E	$	Accept

Operator Precedence Table

	id	+	*	$
id	–	·>	·>	·>
+	<·	·>	<·	·>
*	<·	·>	·>	·>
$	<·	<·	<·	–

Size = $O(n^2)$ where n is the No of operators

100×100

$10^8 \times 10^8$

Compiler Design

Size = $O(n^2)$

Operator function Table:-

f \ g	id	+	*	$
id	–	>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	–

f(id) > g(+)

f(+) < g(id)

←

if f(id) > g(id)

f(id) → g(id)

f(id) → g(*)

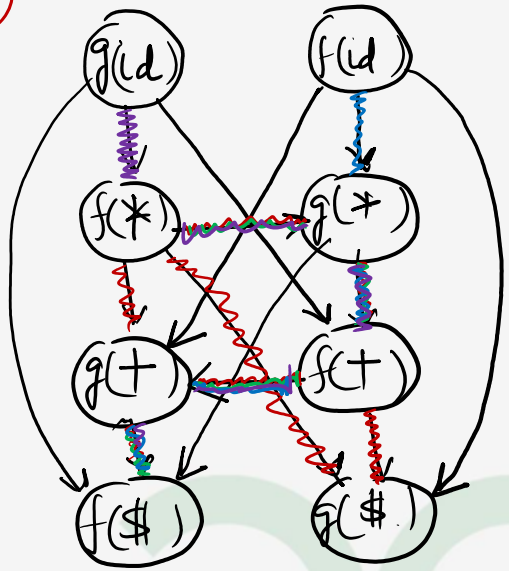
f(+) ← g(id)

operator function Table

	+	*	id	$
f	2	4	4	0
g	1	3	5	0

Size of Table = $O(2 \times n)$ = $O(n)$

No of operators



Compiler Design

Compiler Design

Thank You !