LL(1) Parse Table Generation

S ::= Sb | a

This grammar is left recursive, hence not suitable for recursive descent or LL parsing.

Step 1. Remove Left Recursion pg 39 in notes

S ::= a | aA
A :: = b | bA

Now the grammar has common prefixes

S is a non-term, a,b are terms.

LL(1) Parse Table Generation

S ::= a | aA
A :: = b | bA

Now the grammar has common prefixes

Step 2. Remove common prefixes pg 39 notes

S ::= aB
B ::= “” | A
A ::= bC
C ::= “” | A

Now two non-terminals are the same. Simplify.

LL(1) Parse Table Generation

S ::= aB
B ::= “” | A
A ::= bC
C ::= “” | A

Step 3. Simplify

S ::= aB
B ::= “” | A
A ::= bB

Now two non-terminals are the same. Simplify.

Looks OK Next the firsts and follows.

LL(1) Parse Table Generation

The First Set of a string of symbols is the set of tokens (plus “” indicator) that may appear when the string is expanded. This is only interesting when the string begins with one or more non-Terminals.

The Follow Set of a non-Terminal is the set of tokens that can immediately follow the non-Terminal in some syntactic form.

LL(1) Parse Table Generation

S ::= aB
B ::= “” | A
A ::= bB

Looks OK Next the firsts and follows.

Step 4. Compute firsts of all Non-Terms pg 43 Notes

First(S) = { a }
First(B) = { b, “” } empty string indicates B can be empty
First(A) = { b }

Whenever we expect an S, the next token must be a
Whenever we expect a B, the next token must be b or whatever could follow a B
Whenever we expect an A, the next token must be a b.

LL(1) Parse Table Generation

S ::= aB
B ::= “” | A
A ::= bB

Step 5. Compute follows of all Non-Terms pg 44, Notes

Follow(S) = { $ }
Follow(B) = { $ }
Follow(A) = { $ }

$ = end of string

LL(1) Parse Table Generation

S ::= aB
B ::= “” | A
A ::= bB

First(S) = { a } Follow(S) = { $ }
First(B) = { b, “” } Follow(B) = { $ }
First(A) = { b } Follow(A) = { $ }

Step 6. Check LL(1) pg 44 notes

Rule 1 does not apply, Rule 2 applies to B
Require First(B) * Follow(B) = {} O.K.

* means set intersection

LL(1) Parse Table Generation

S ::= aB
B ::= “” | A
A ::= bB

Step 7. Write the grammar in standard form (number the productions).

1. S ::= aB
2. B ::= “”
3. B ::= A
4. A ::= bB

LL(1) Parse Table Generation

1. S ::= aB
2. B ::= “”
3. B ::= A
4. A ::= bB

Step 8. Compute the predict function for each production.

S: Predict(1) = first(aB) = { a }
B: Precict(2) = first(empty) + follow(B) = Follow(B) = { $ }
B: Predict(3) = first(A) = { b }
A: Predict(4) = first(bB) = { b }

LL(1) Parse Table Generation

S: Predict(1) = { a }
B: Precict(2) = { $ }
B: Predict(3) = { b }
A: Predict(4) = { b }

Step 9. Re-arrange into a table

	a	b	$
S	1		
A		4	
B		3	2

Finally, output this table and the standard form grammar to the parser.

LL(1) Parse Table Generation

	a	b	$
S	1		
A		4	
B		3	2

1. S ::= aB
2. B ::= “”
3. B ::= A
4. A ::= bB

Parser

1. Terminal on parse stack--match against input.
2. Non-Term on parse stack -- replace with RHS of predicted production using next input token.
3. Action on parse stack -- execute it.