Parsing Algorithms	
Input	1. CFG 2. word (w)
Steps	Derivation for w: ● w is in the language No derivation ● w is not in the language
Output	Parse Tree (**with leftmost derivation**)

Augmented Grammars

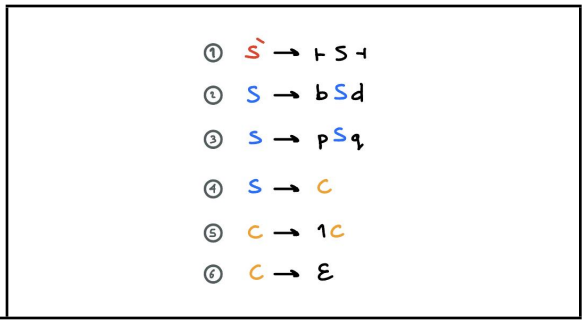
Ensures S has only 1 production rule

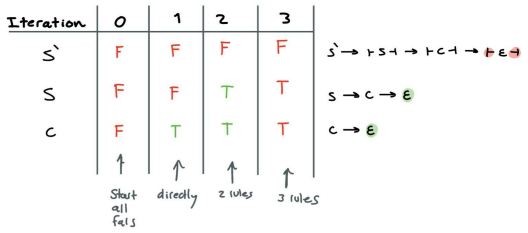
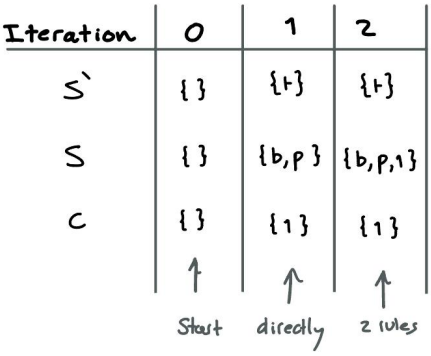
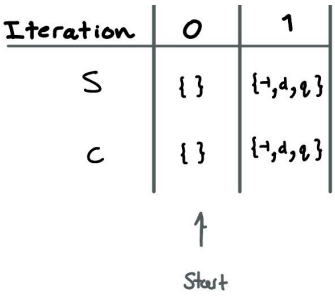
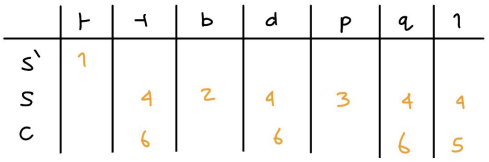
Augmented Grammar	
S'	S' = ⊢**S**⊣

Predict Table

Leftmost does not eliminate uncertainty in **choice** of rules (only **order**)

Predict[A][a] : given a non-terminal A and a lookahead terminal a, will predict which rule to choose for A



Nullable(A)	**A** ⇒* ε	
First(A)	**Look at the RHS of rule: A → BCD…** **B** is a terminal: add **B** Else: add First(**B**) If **B** is nullable: add First(**C**)	
Follow(A)	**Look at the RHS of rule:** C → …A…	
Predict Table	**For Each rule: A ⇒ B** 1) add First(**B**): add the rule 2) If Nullable(**B**) a) add Follow(A)	

LL(1) Top-Down Parsing Algorithm

A grammar is LL(1) ⇔ each cell of the Predict table contains at most 1 rule

L	Left-to-Right scan of input
L	Leftmost Derivations

1	**lookahead 1** symbol of input (for picking a rule)

```
push S'

for(each x in ⊢input⊣)
{
    while(TOS is non-terminal)
    {
      rule # = predict[TOS][x]
      TOS.pop
      TOS.push rule

      ERROR if more than 1 rule
    }

    if(TOS != next input input) ERROR
    else
            pop
            pop (from input)
}
accept //stack is empty
```

2 ways to get **ERROR**

1. TOS is a terminal, does not match the next input symbol
2. Predict[A][a] = 0 or 2+ rules

Example 1

Predict Table:

1) S` → ⊢S⊣
2) S → AyB
3) A → ab
4) A → cd
5) B → z
6) B → wx

	⊢	⊣	y	a	b	c	d	w	x	z
S`	1									
S				2		2				
A				3		4				
B					5			6		

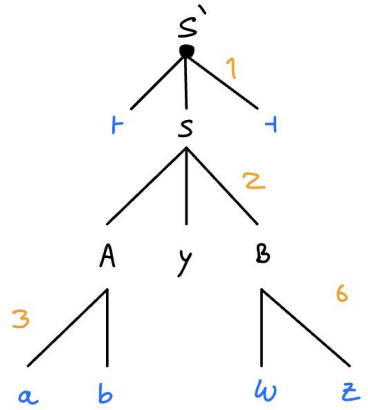
① push S` → ⊢S⊣

TOS

Read	Unread	Stack	non-terminal	terminal	Action
ε	⊢abywx⊣	⊢S⊣			
ε	⊢abywx⊣	⊢S⊣		⊢ matches ⊢	→ pop read
⊢	abywx⊣	S⊣	predict [S][a] = 2		→ pop push(B,y,A)
⊢	abywx⊣	AyB⊣	predict [A][a] = 3		→ pop push(b,a)
⊢	abywx⊣	abyB⊣		a matches a	→ pop read
⊢a	bywx⊣	byB⊣		b matches b	→ pop read
⊢ab	ywx⊣	yB⊣		y matches y	→ pop read
⊢aby	wx⊣	B⊣	predict [B][w] = 6		→ pop push(w,w)
⊢aby	wx⊣	wx⊣		w matches w	→ pop read
⊢abyw	x⊣	x⊣		x matches x	→ pop read
⊢abywx	⊣	⊣		⊣ matches ⊣	

Resulting Parse Tree:

Sequence of rules: 1, 2, 3, 6



Example 2:

Predict Table:

1) S` → ⊢S⊣
2) S → Tz
3) z → +Tz
4) z → ε
5) T → FT`
6) T` → *FT`
7) T` → ε
8) F → a|b|c

Predict:

	⊢	⊣	+	*	a	b	c
S`	1						
S					2	2	2
z		4	3				
T					5	5	5
T`		7	7	6			
F					8	8	8

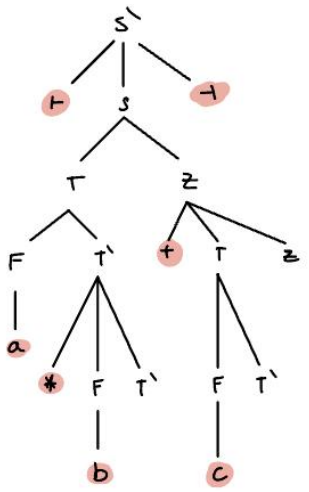
Nullable	
S`	F
S	F
z	T
T	F
T`	T
F	F

First	
S`	⊢
S	a,b,c
z	+
T	a,b,c
T`	*
F	a,b,c

Follow	
S`	∅
S	⊣
z	⊣
T	+, ⊣
T`	+, ⊣
F	*, +, ⊣

⊢a*b+c⊣

Read	Stack		Action
ε	S`		
⊢	S`	predict [⊢][S`] = 1	pop, push(⊣,S,⊢)
⊢	⊢S⊣	match(⊢)	pop read
⊢	S⊣	predict [a][S] = 2	pop, push(T, z)
⊢a	Tz⊣	predict [a][T] = 5	pop, push(F, T`)
⊢a	FT`z⊣	predict [a][F] = 8	pop, push(a)
⊢a	aT`z⊣	match(a)	pop read
⊢a*	T`z⊣	predict [*][T`] = 6	pop, push(*FT`)
⊢a*	*FT`z⊣	match(*)	pop read
⊢a*b	FT`z⊣	predict [b][F] = 8	pop, push(b)
⊢a*b	bT`z⊣	match(b)	pop read
⊢a*b+	T`z⊣	predict [+][T`] = 7	pop, push( )
⊢a*b+	z⊣	predict [+][z] = 3	pop, push(+Tz)
⊢a*b+	+Tz⊣	match(+)	pop read
⊢a*b+c	Tz⊣	predict [c][T] = 5	pop, push(FT`)
⊢a*b+c	FT`z⊣	predict [c][F] = 8	pop, push(c)
⊢a*b+c	cT`z⊣	match(c)	pop read
⊢a*b+c⊣	T`z⊣	predict [⊣][T`] = 7	pop, push( )
⊢a*b+c⊣	z⊣	predict [⊣][z] = 4	pop, push( )
⊢a*b+c⊣	⊣	match(⊣)	pop read
⊢a*b+c⊣	∅		



S` → ⊢S⊣
→ ⊢Tz⊣
→ ⊢FT`z⊣
→ ⊢aT`z⊣
→ ⊢a*FT`z⊣
→ ⊢a*bT`z⊣
→ ⊢a*bz⊣
→ ⊢a*b+Tz⊣
→ ⊢a*b+FT`z⊣
→ ⊢a*b+c⊣

Notes on LL1:

1. We push the symbols in reverse so that the leftmost symbol ends up on TOS. This ensures that the first non- terminal we encounter is the leftmost non-terminal.
2. At any point: current derived string = Read-Input + Stack

Limitations

1. Only 1 rule in the predict table:
2. If ‘A’ is nullable, it can’t has the same terminal in: first(A) and follow(A)
3. Only 1 way to derive ε from a nullable symbol

Left-Associative Grammars are **never** LL(1)
Right-Associative Grammars **can be** LL(1)

Solution

1. We must **at least** make it right recursive
2. Factor **if needed**
3. Add precedence **if needed**