

# Section 4: CUP & LL

CSE 401/M501

Adapted from Spring 2021

# Administrivia

- Homework 2 is due tonight!
  - You have late days if you need them (2 max)
- Parser is due one week from today
  - Be sure to check your Scanner feedback – out later this week

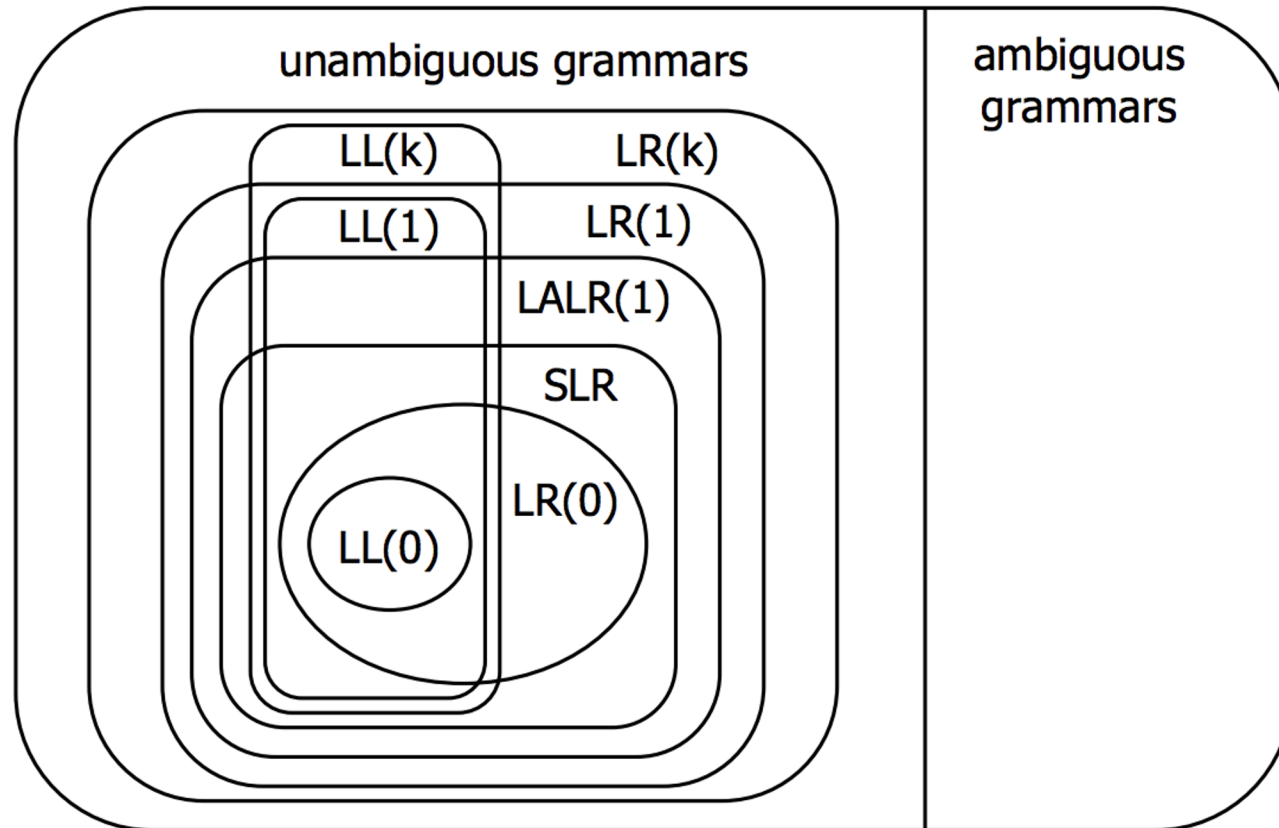


14:30-15:20 Lecture CSE2 G10 <i>ASTs &amp; visitors;</i>	17	17:00-18:00 OH (John) <i>zoom</i>	18	13:30-14:30 OH (Randy) CSE2 151 + <i>zoom</i>	19	Section <i>CUP parser generator, ASTs; LL parsing</i>	20	14:30-15:20 Lecture CSE2 G10 <i>Intro to semantics and type checking (4.1-4.2)</i>	21
16:00-17:00 OH (John) <i>zoom</i>		18:00-19:00 OH (Armand) Allen 5th floor breakout + <i>zoom</i>		14:30-15:20 Lecture CSE2 G10 <i>LL Parsing &amp; recursive descent (3.3)</i>		17:00-18:00 OH (Varun) CSE2 153 + <i>zoom</i>		16:30-17:30 OH (Armand) Allen 2nd floor breakout + <i>zoom</i>	
17:00-18:00 OH (Varun) <i>zoom</i>				16:00-17:00 OH (Apollo) CSE2 151 + <i>zoom</i>		18:00-19:00 OH (Apollo) CSE2 153 + <i>zoom</i>		17:30-18:30 OH (Randy) CSE2 151 + <i>zoom</i>	
				18:30-19:30 OH (Robert) CSE2 153 + <i>zoom</i>		19:00-20:00 OH (Robert) CSE2 153 + <i>zoom</i>			
						23:00 hw2 due (LR grammars)			

# Parser Live Demo

A video recording of this demo is posted on the website as a supplement

# Language Hierarchies



# The CUP parser generator

- Uses LALR(1)
  - A little weaker (less selective), but many fewer states than LR(1) parsers
  - Handles most realistic programming language grammars
  - More selective than SLR (or LR(0)) about when to do reductions, so works for more languages

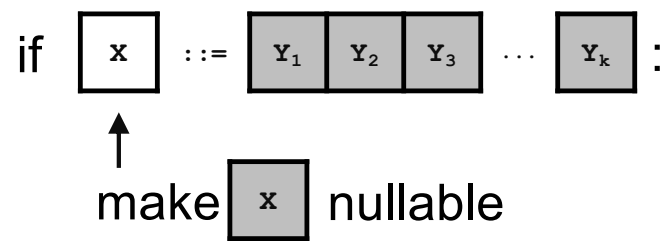
# The CUP parser generator

- LALR(1) parser generator based on YACC and Bison
- CUP can resolve some ambiguities itself
  - Precedence for reduce/reduce conflicts
  - Associativity for shift/reduce conflicts
  - Useful for our project for things like arithmetic expressions ( $\text{exp}+\text{exp}$ ,  $\text{exp}*\text{exp}$  for fewer non-terminals, then add precedence and associativity declarations).  
Read the docs

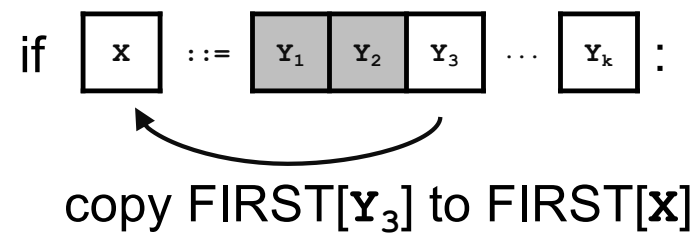
# Computing FIRST, FOLLOW, & nullable (3)

$\boxed{y}$  = nullable

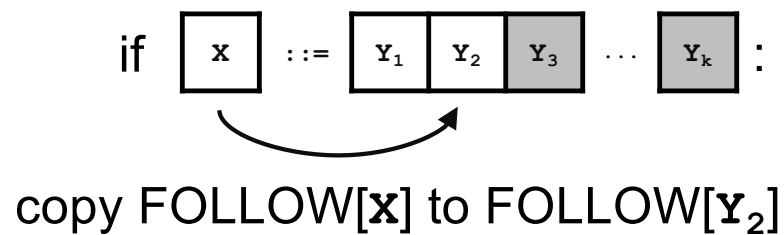
①



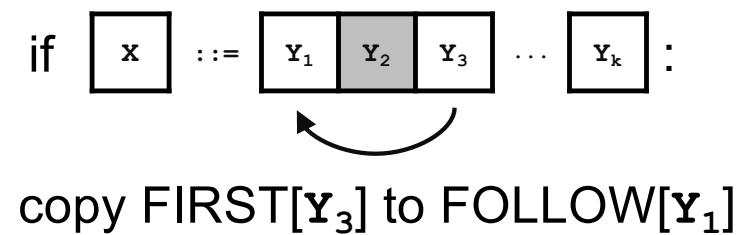
②



③



④



# Computing FIRST, FOLLOW, and nullable

```
repeat
  for each production  $X := Y_1 Y_2 \dots Y_k$ 
    if  $Y_1 \dots Y_k$  are all nullable (or if  $k = 0$ )
      set nullable[X] = true
    for each  $i$  from 1 to  $k$  and each  $j$  from  $i+1$  to  $k$ 
      if  $Y_1 \dots Y_{i-1}$  are all nullable (or if  $i = 1$ )
        add FIRST[ $Y_i$ ] to FIRST[X]
      if  $Y_{i+1} \dots Y_k$  are all nullable (or if  $i = k$ )
        add FOLLOW[X] to FOLLOW[ $Y_i$ ]
      if  $Y_{i+1} \dots Y_{j-1}$  are all nullable (or if  $i+1=j$ )
        add FIRST[ $Y_j$ ] to FOLLOW[ $Y_i$ ]
Until FIRST, FOLLOW, and nullable do not change
```

L L (k)



## Left-to-Right

Only takes one pass,  
performed from the left

## Leftmost

At each point, finds the  
derivation for the leftmost  
handle (**top-down**)

## k Terminal Lookahead

Must determine derivation  
from the next unparsed  
terminal in the string  
Typically  $k = 1$ , just like LR

# LL( $k$ ) parsing

- LL( $k$ ) scans left-to-right, builds leftmost derivation, and looks ahead  $k$  symbols
- The LL condition enable the parser to choose productions correctly with 1 symbol of look-ahead
- We can often transform a grammar to satisfy this if needed

# LL(1) parsing: An example top-down derivation of “a z x”

0.  $S ::= a B$

1.  $B ::= C x \mid y$

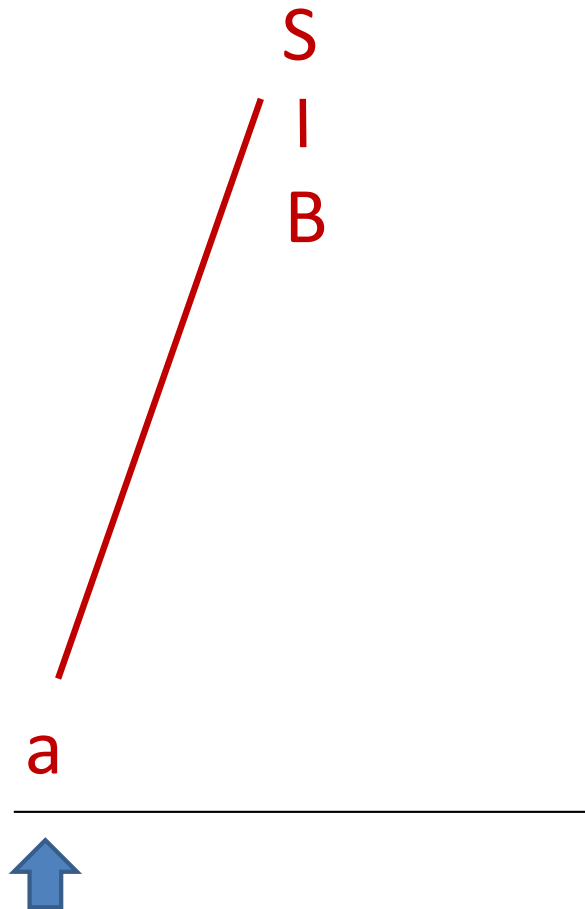
2.  $C ::= \epsilon \mid z$

Lookahead      Remaining

a

z x

# Top-Down Derivation of "a z x"



0.  $S ::= a B$

1.  $B ::= C x \mid y$

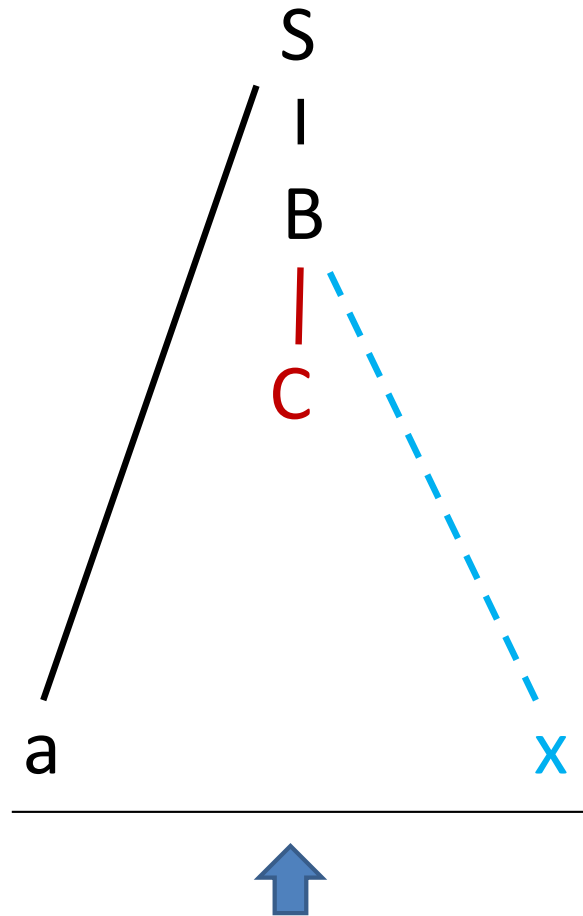
2.  $C ::= \epsilon \mid z$

Lookahead      Remaining

$a$

$z x$

# Top-Down Derivation of "a z x"



0.  $S ::= a B$

1.  $B ::= C x \mid y$

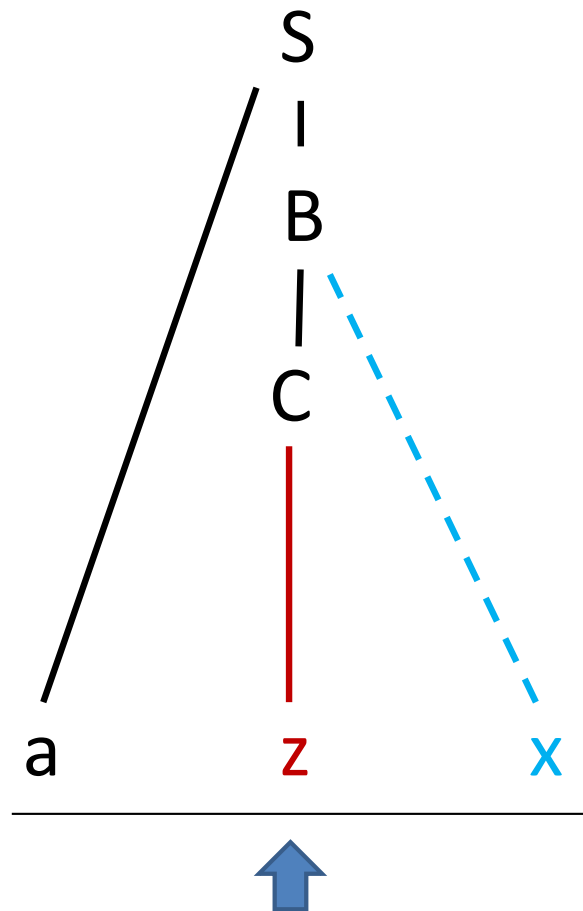
2.  $C ::= \epsilon \mid z$

Lookahead      Remaining

z

x

# Top-Down Derivation of "a z x"



0.  $S ::= a B$

1.  $B ::= C x \mid y$

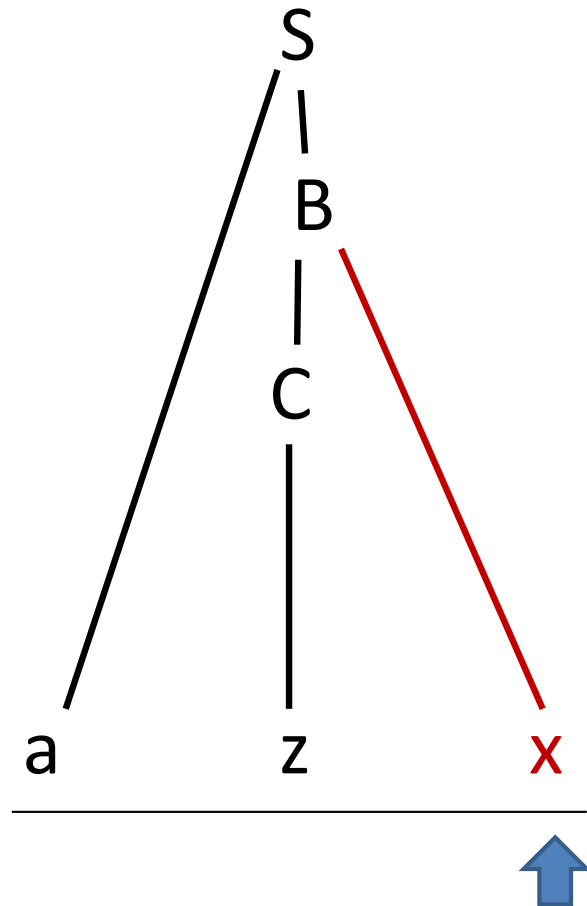
2.  $C ::= \epsilon \mid z$

Lookahead      Remaining

z

x

# Top-Down Derivation of "a z x"



0.  $S ::= a B$

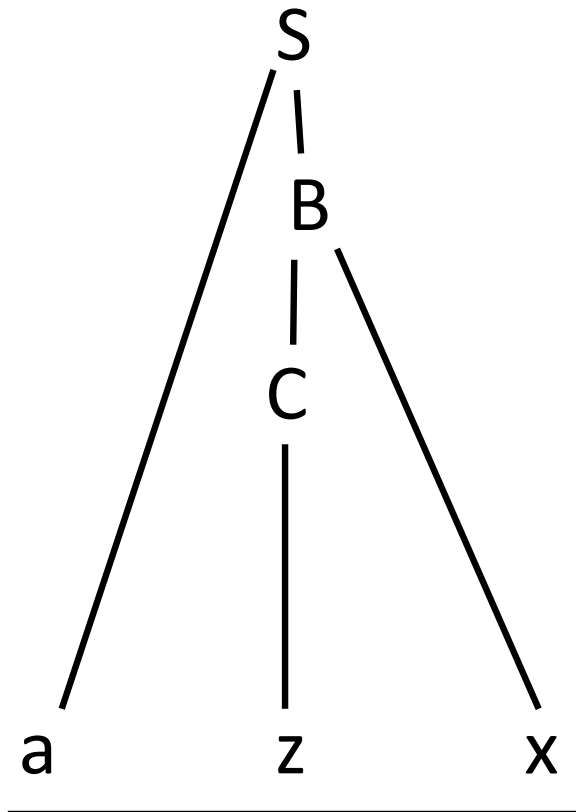
1.  $B ::= C x \mid y$

2.  $C ::= \epsilon \mid z$

Lookahead      Remaining

x

# Top-Down Derivation of "a z x"



0.  $S ::= a B$

1.  $B ::= C x \mid y$

2.  $C ::= \epsilon \mid z$

Successful parse!

# LL Condition

For each nonterminal in the grammar:

- Its *productions* must have disjoint FIRST sets

✗  $A ::= x \mid B$   
 $B ::= x$

✓  $A ::= x \mid B$   
 $B ::= y$

- If it is *nullable*, the FIRST sets of its productions must be disjoint from its FOLLOW set

✗  $S ::= A x$   
 $A ::= \varepsilon \mid x$

✓  $S ::= A y$   
 $A ::= \varepsilon \mid x$

\*\*We can often transform a grammar to satisfy this if needed

# Canonical FIRST Conflict

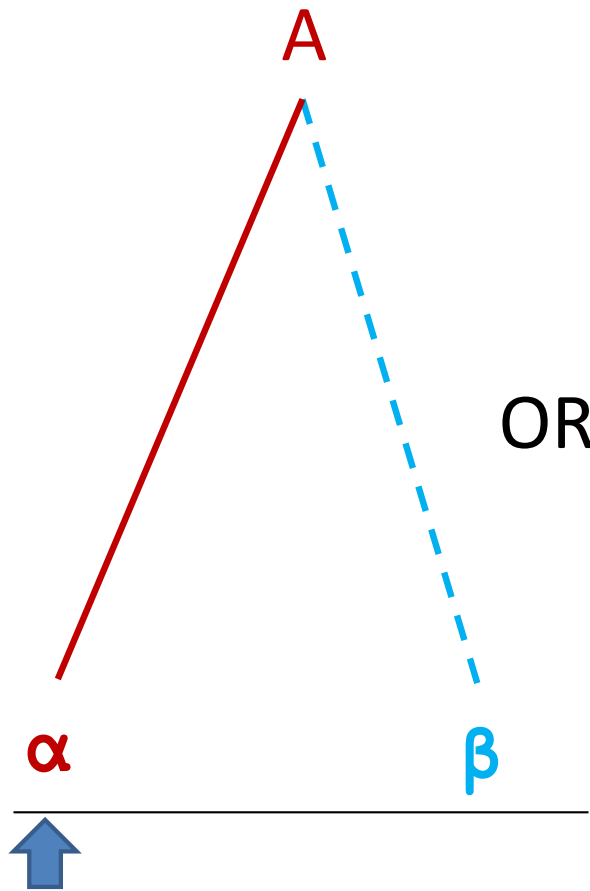
## Problem

$$0. \mathbf{A} ::= \alpha\beta \mid \alpha\gamma$$

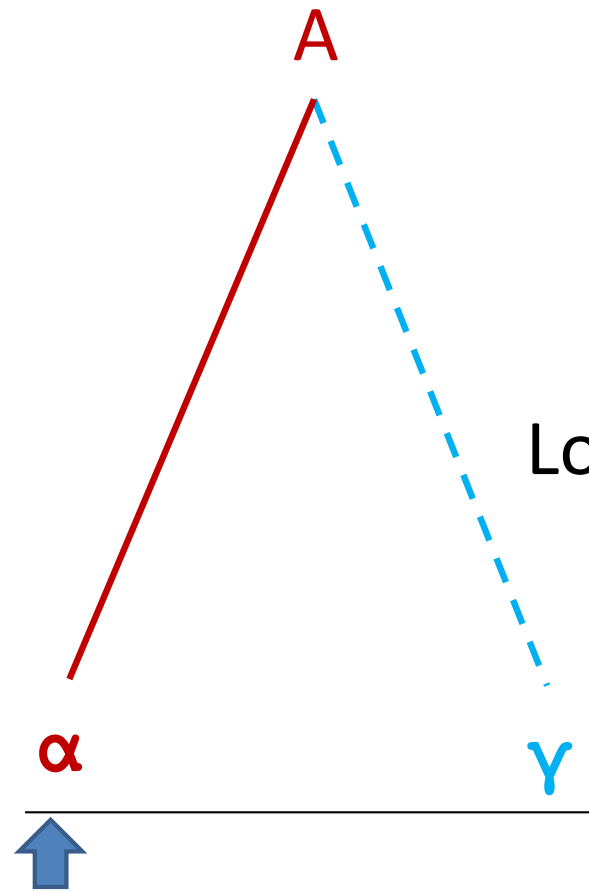
The FIRST sets of the right-hand sides for the SAME NON-TERMINAL must be disjoint!

# Let's try a top-down derivation of $\alpha\beta$

0.  $A ::= \alpha\beta \mid \alpha\gamma$



OR



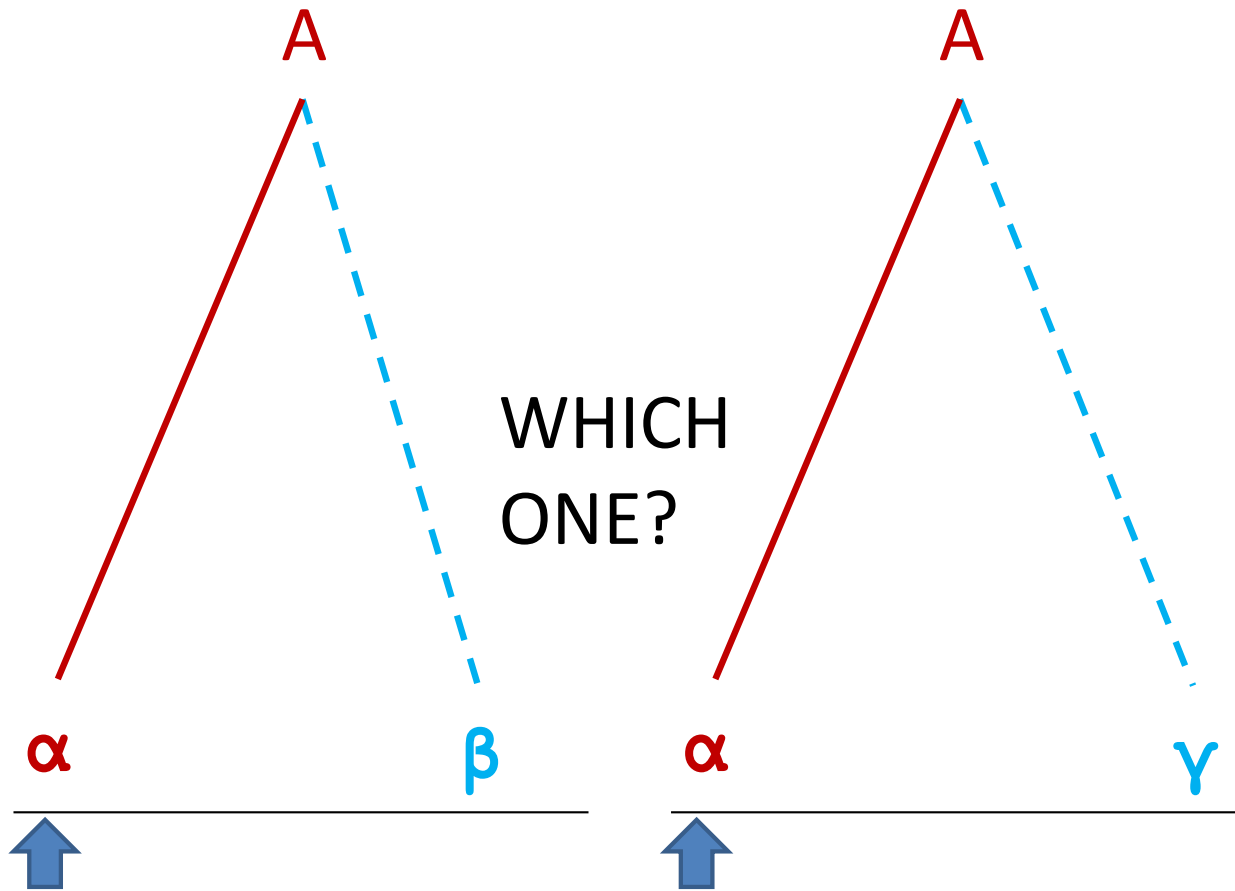
Lookahead

$\alpha$

Remaining

$\beta$

# Let's try a top-down derivation of $\alpha\beta$



0.  $A ::= \alpha\beta \mid \alpha\gamma$

We don't know!

We are using an LL(1) parser, we can't see more than  $\alpha$ !

# Canonical FIRST Conflict Solution

## Solution

0.  $A ::= \alpha\beta \mid \alpha\gamma$

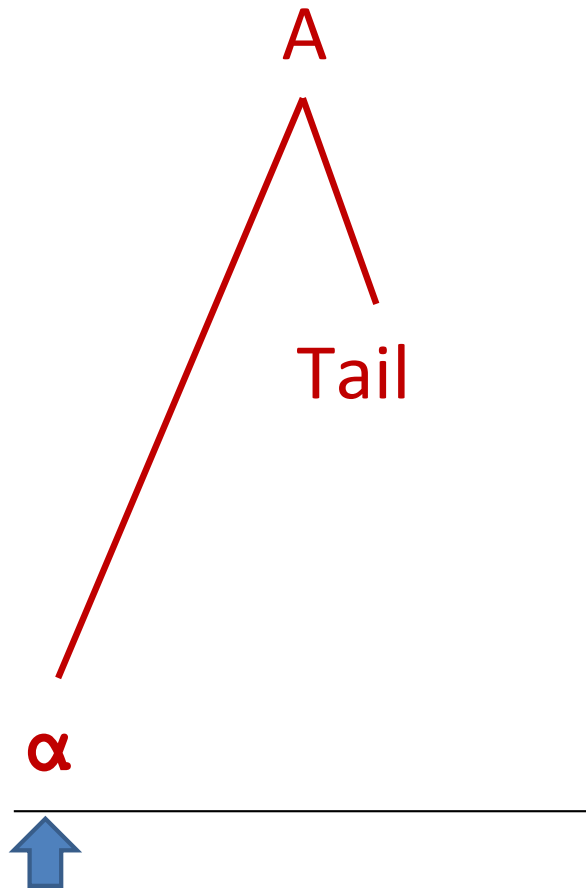
0.  $A ::= \alpha \text{Tail}$

1.  $\text{Tail} ::= \beta \mid \gamma$

Factor out the  
common prefix

When multiple productions of a nonterminal share a common prefix, turn the different suffixes into a new nonterminal.

# Top-Down Derivation of “ $\alpha\beta$ ”



0.  $A ::= \alpha \text{ Tail}$

1.  $\text{Tail} ::= \beta \mid \gamma$

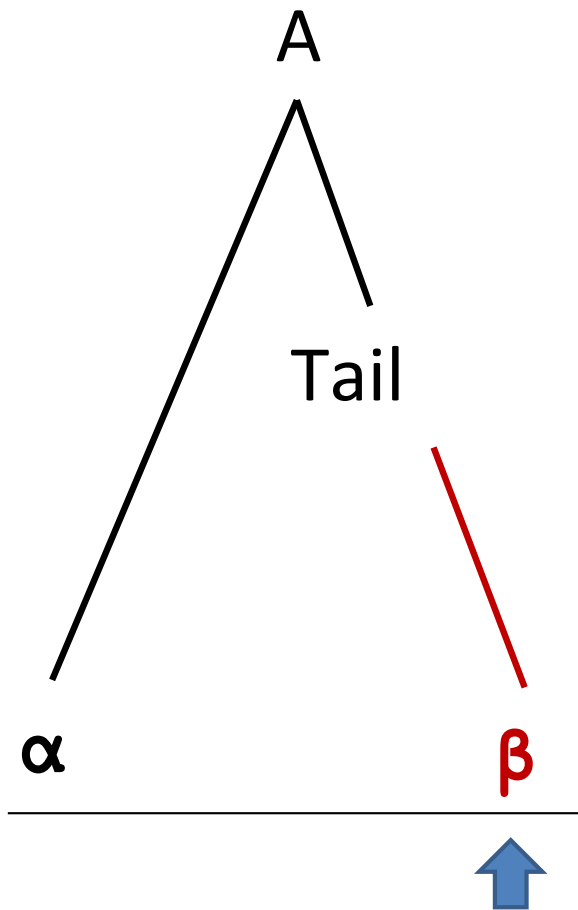
Lookahead

Remaining

$\alpha$

$\beta$

# Top-Down Derivation of “ $\alpha\beta$ ”



0.  $A ::= \alpha \text{ Tail}$

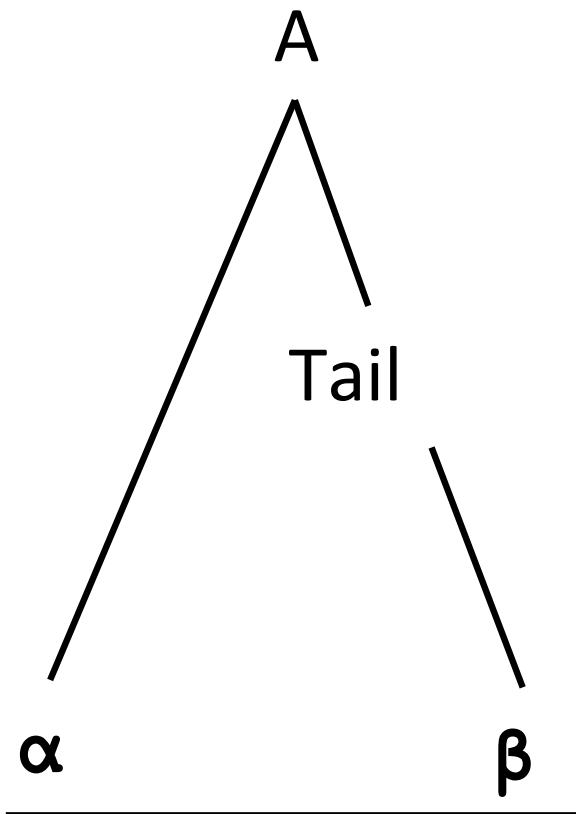
1.  $\text{Tail} ::= \beta \mid \gamma$

Lookahead

Remaining

$\beta$

# Top-Down Derivation of “ $\alpha\beta$ ”



0.  $A ::= \alpha Tail$

1.  $Tail ::= \beta \mid \gamma$

Successful parse!

# Changing original grammar a little (Grammar 1)

- 0.  $S ::= a B \mid a w$
- 1.  $B ::= C x \mid y$
- 2.  $C ::= \varepsilon \mid z$

Lookahead      Remaining

a

z x

# What's the issue?

0.  $S ::= \mathbf{a} B \mid \mathbf{a} w$   
1.  $B ::= C x \mid y$   
2.  $C ::= \varepsilon \mid z$

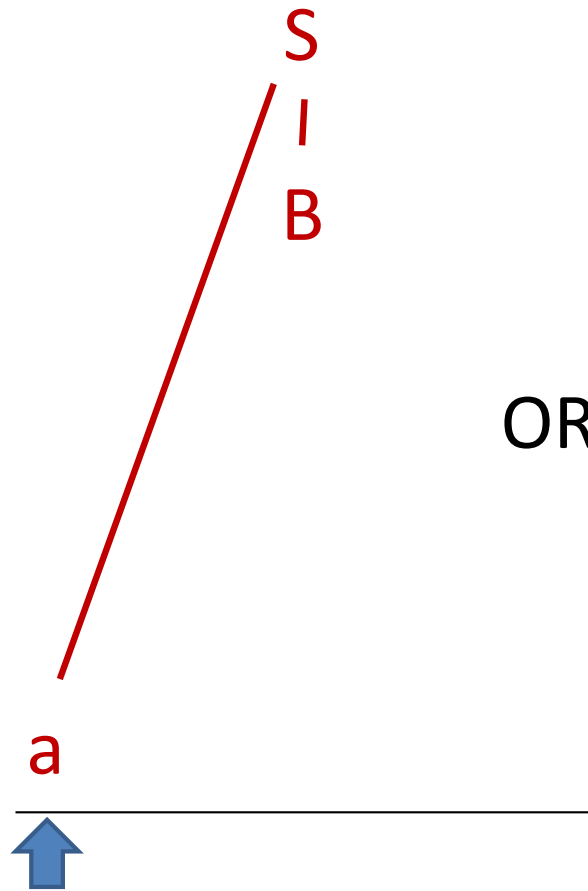
There's a FIRST Conflict!

# Top-Down Derivation of "a z x": LL(1) can't parse

0.  $S ::= a B \mid a w$

1.  $B ::= C x \mid y$

2.  $C ::= \varepsilon \mid z$

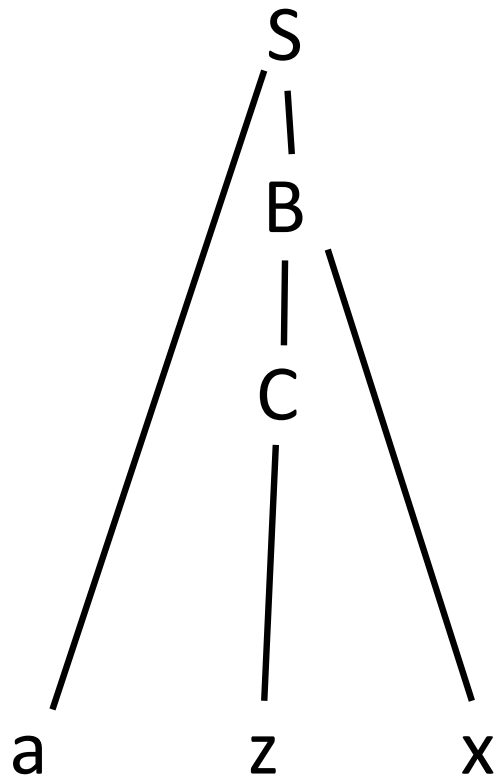


Lookahead      Remaining

a

z x

# Parse Tree without changing Grammar



0.  $S ::= a B \mid a w$

1.  $B ::= C x \mid y$

2.  $C ::= \varepsilon \mid z$

# Applying the Fix: Factor out the Common Prefix

0. **S ::= a Tail**

1. **Tail ::= B | w**

2. B ::= C x | y

3. C ::= ε | z

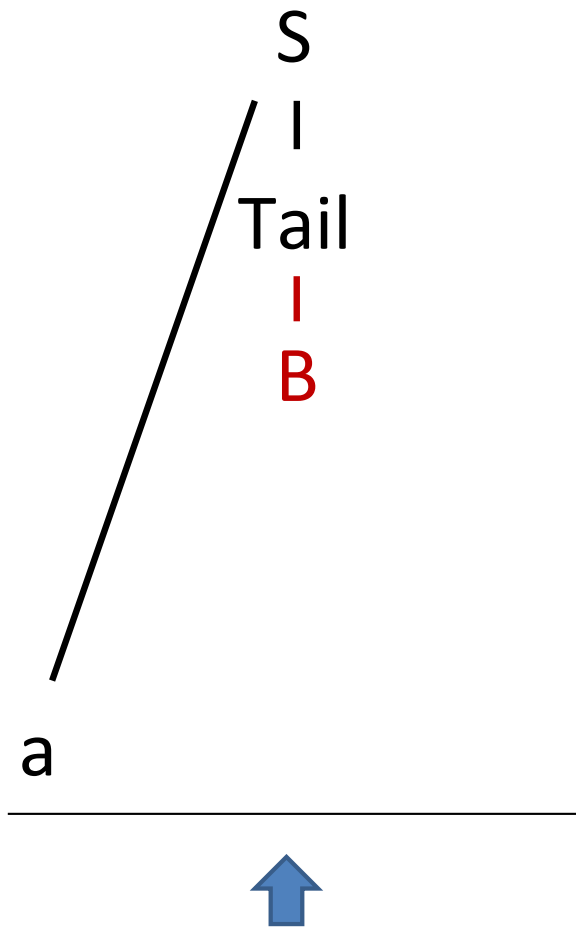
# Top-Down Derivation of "a z x"



0. **S ::= a Tail**
1. **Tail ::= B | w**
2. **B ::= C x | y**
3. **C ::= ε | z**

Lookahead	Remaining
<b>a</b>	z x

# Top-Down Derivation of "a z x"



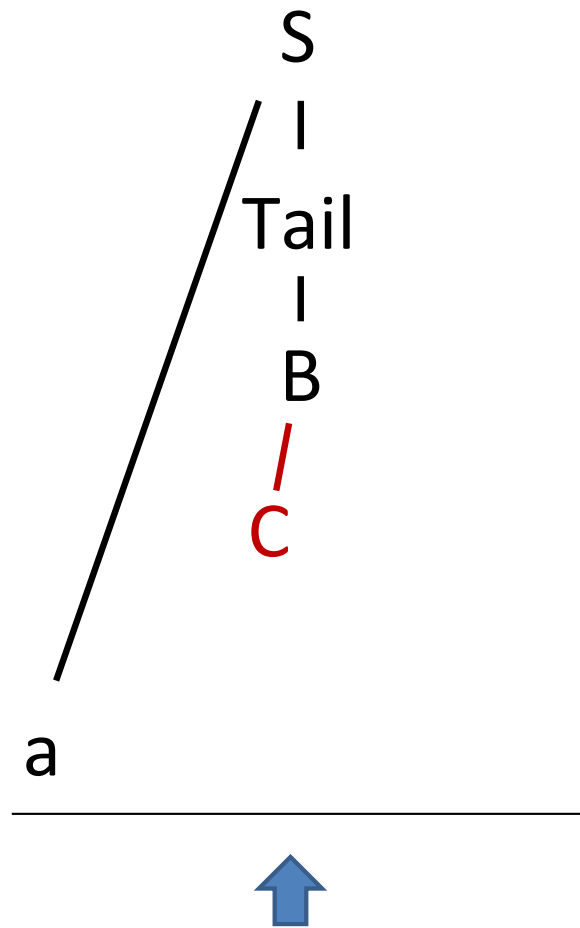
0.  $S ::= a \text{ Tail}$
1.  $\text{Tail} ::= B \mid w$
2.  $B ::= C \ x \mid y$
3.  $C ::= \varepsilon \mid z$

Lookahead      Remaining

z

x

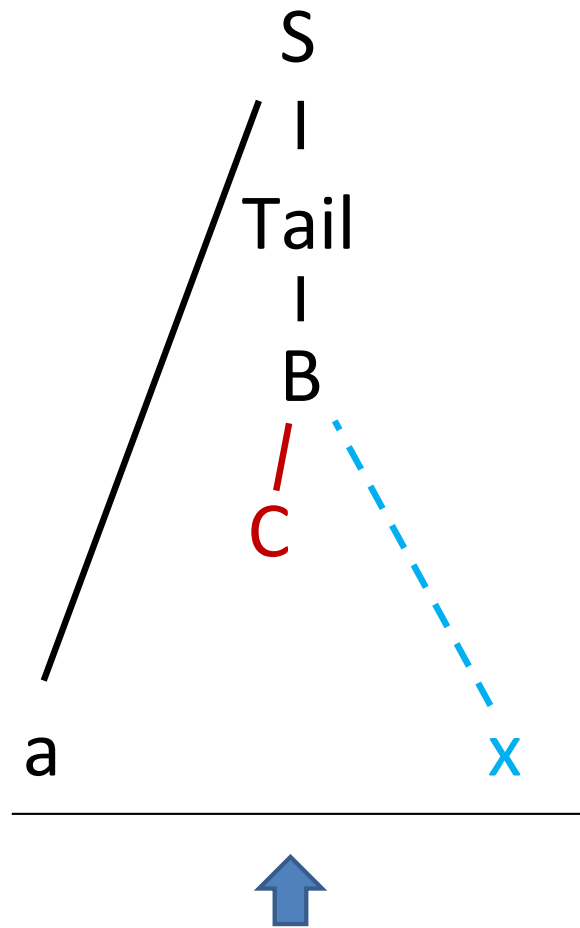
# Top-Down Derivation of "a z x"



0.  $S ::= a \text{ Tail}$
1.  $\text{Tail} ::= B \mid w$
2.  $B ::= C \ x \mid y$
3.  $C ::= \varepsilon \mid z$

Lookahead	Remaining
<span style="border: 1px solid black; padding: 2px;">z</span>	x

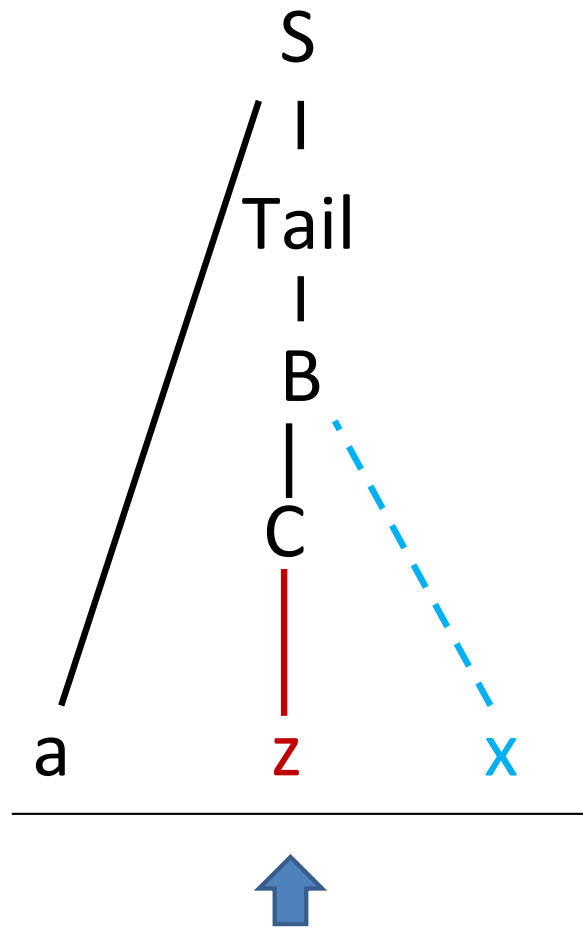
# Top-Down Derivation of "a z x"



0. **S ::= a Tail**
1. **Tail ::= B | w**
2. **B ::= C x | y**
3. **C ::= ε | z**

Lookahead	Remaining
<span style="border: 1px solid black; padding: 2px;">z</span>	x

# Top-Down Derivation of "a z x"



0. **S ::= a Tail**

1. **Tail ::= B | w**

2. **B ::= C x | y**

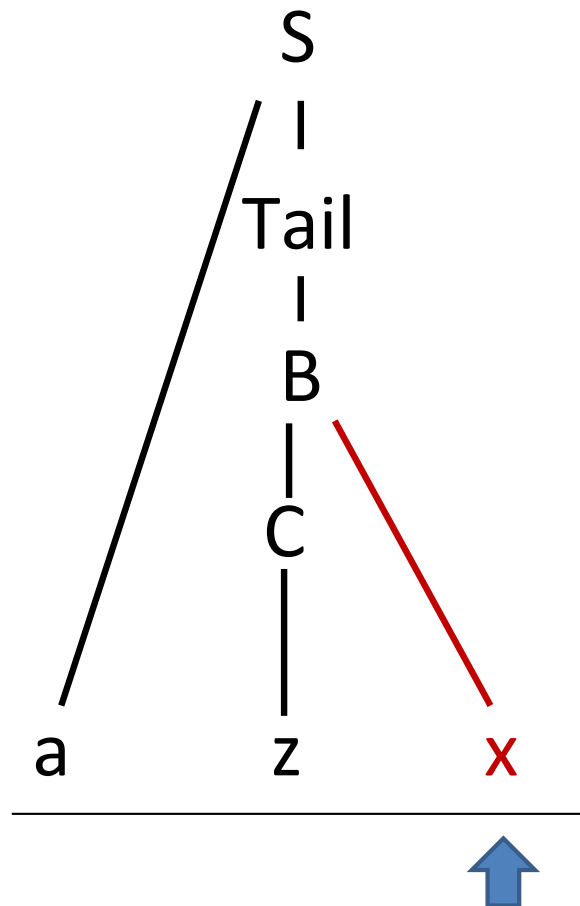
3. **C ::= ε | z**

Lookahead      Remaining

**z**

x

# Top-Down Derivation of "a z x"

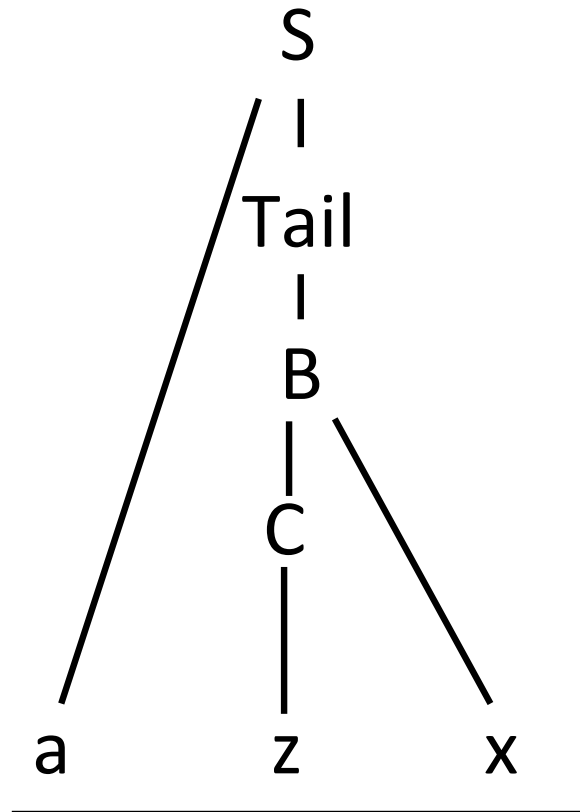


0.  $S ::= a \text{ Tail}$
1.  $\text{Tail} ::= B \mid w$
2.  $B ::= C \ x \mid y$
3.  $C ::= \varepsilon \mid z$

Lookahead      Remaining

x

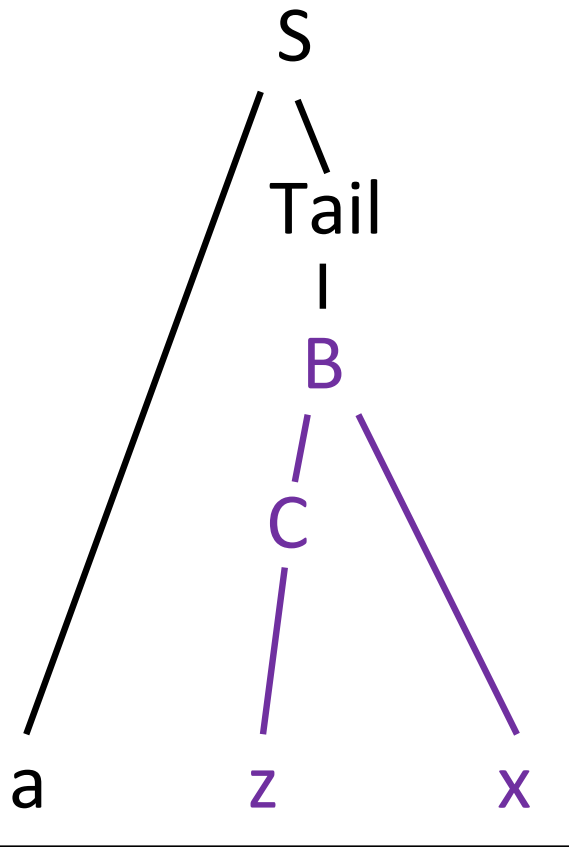
# Top-Down Derivation of "a z x"



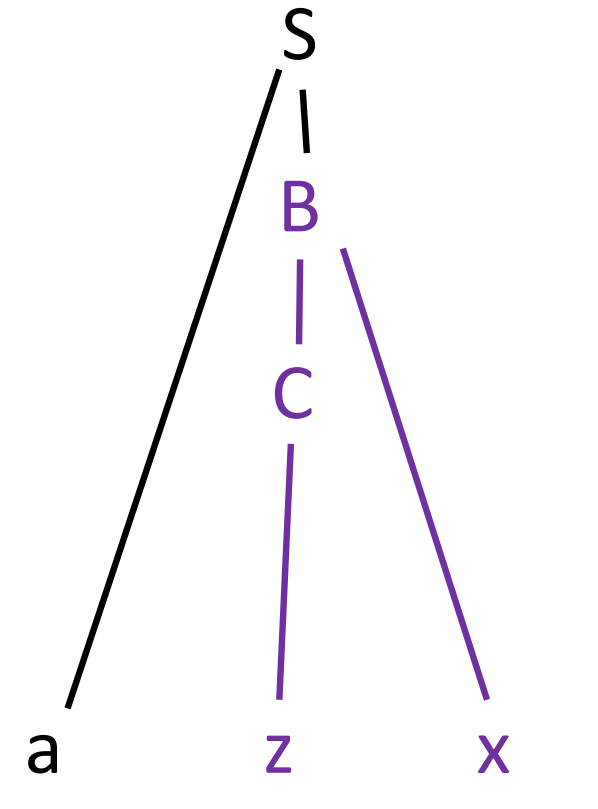
0. **S ::= a Tail**
1. **Tail ::= B | w**
2. **B ::= C x | y**
3. **C ::= ε | z**

Success!

# Comparing Parse Trees



Purple trees  
are the same!



# LL Condition

For each nonterminal in the grammar:

- Its *productions* must have disjoint FIRST sets

✗  $A ::= x \mid B$   
 $B ::= x$

✓  $A ::= x \mid B$   
 $B ::= y$

- If it is *nullable*, the FIRST sets of its productions must be disjoint from its FOLLOW set

✗  $S ::= A x$   
 $A ::= \varepsilon \mid x$

✓  $S ::= A y$   
 $A ::= \varepsilon \mid x$

\*\*We can often transform a grammar to satisfy this if needed

# Canonical FIRST FOLLOW Conflict

## Problem

0.  $A ::= B \alpha$

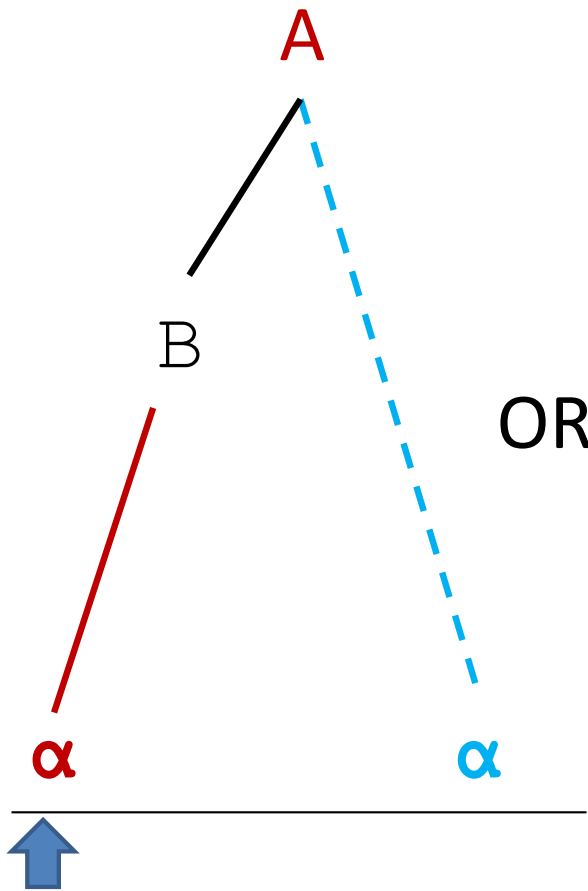
1.  $B ::= \alpha \mid \epsilon$

Because B is nullable, its FOLLOW set must be disjoint from the FIRST sets of its right-hand sides!

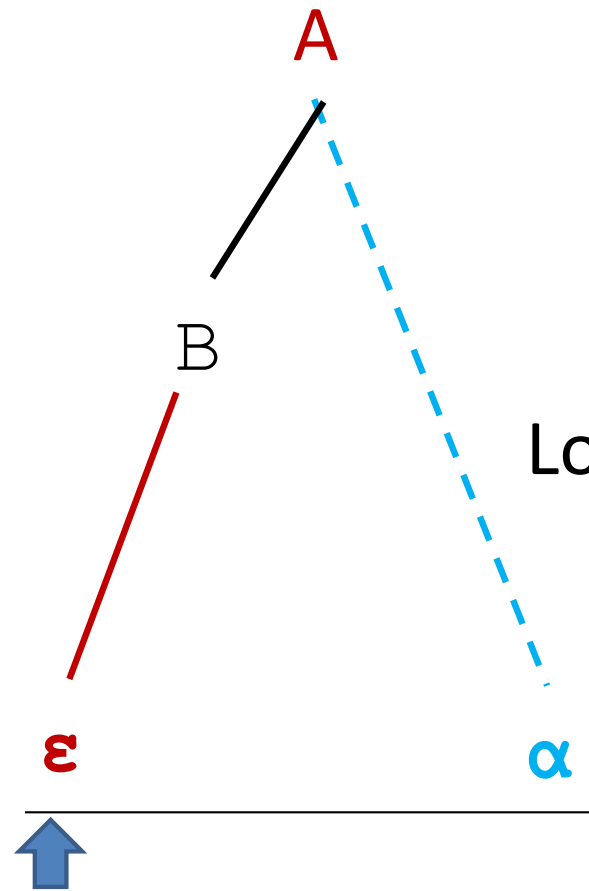
# Let's try a top-down derivation of " $\alpha$ "

0.  $A ::= B \alpha$

1.  $B ::= \alpha \mid \epsilon$



OR

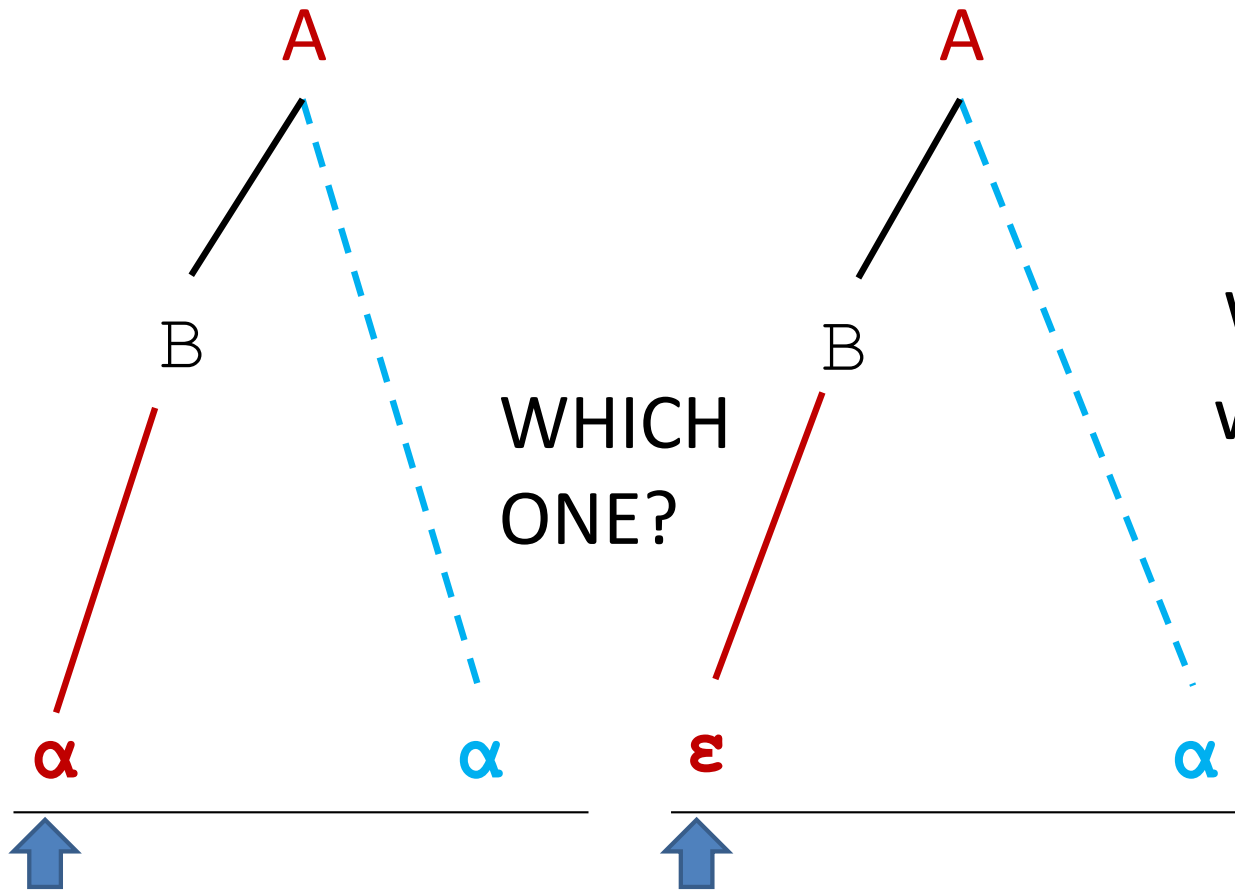


Lookahead

Remaining

$\alpha$

# Let's try a top-down derivation of " $\alpha$ "



0.  $A ::= B \alpha$

1.  $B ::= \alpha \mid \epsilon$

We don't know! Again,  
we can't see more than  
 $\alpha$ !

# Canonical FIRST FOLLOW Conflict Solution

## Solution

0.  $A ::= B \alpha$

1.  $B ::= \alpha \mid \epsilon$

Substitute the  
common prefix

0.  $A ::= \alpha\alpha \mid \alpha$

0.  $A ::= \alpha \text{ Tail}$

Factor out the  
tail

1.  $\text{Tail} ::= \alpha \mid \epsilon$

# Watch out for Nullability! (Grammar 2)

Changing the grammar again...

0.  $S ::= a B$

1.  $B ::= C x \mid y$

2.  $C ::= \epsilon \mid x$

Lookahead

Remaining

a

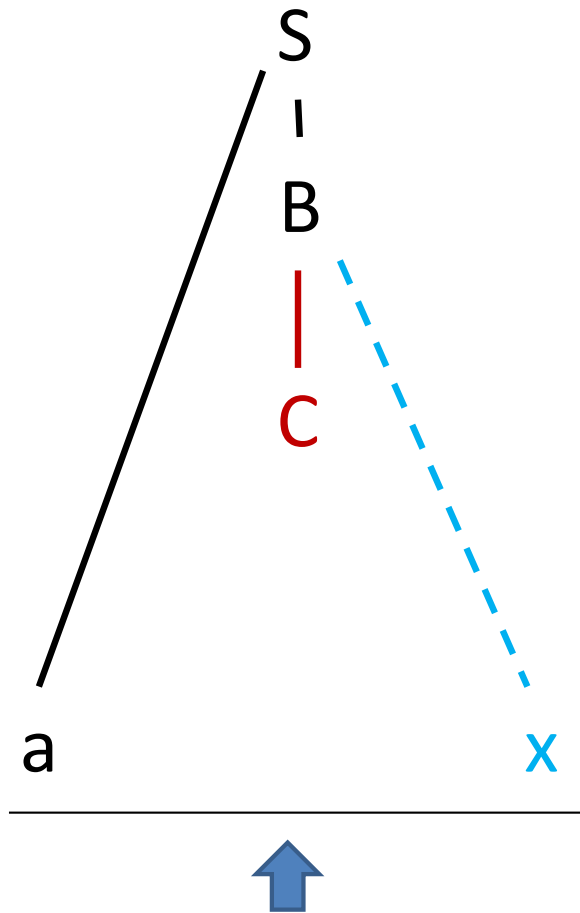
x

# What's the issue?

0.  $S ::= a B$   
1.  $B ::= C \mathbf{x} \mid y$   
2.  $C ::= \varepsilon \mid \mathbf{x}$

FIRST FOLLOW Conflict

# Top down derivation of "ax"



0.  $S ::= a B$

1.  $B ::= C x \mid y$

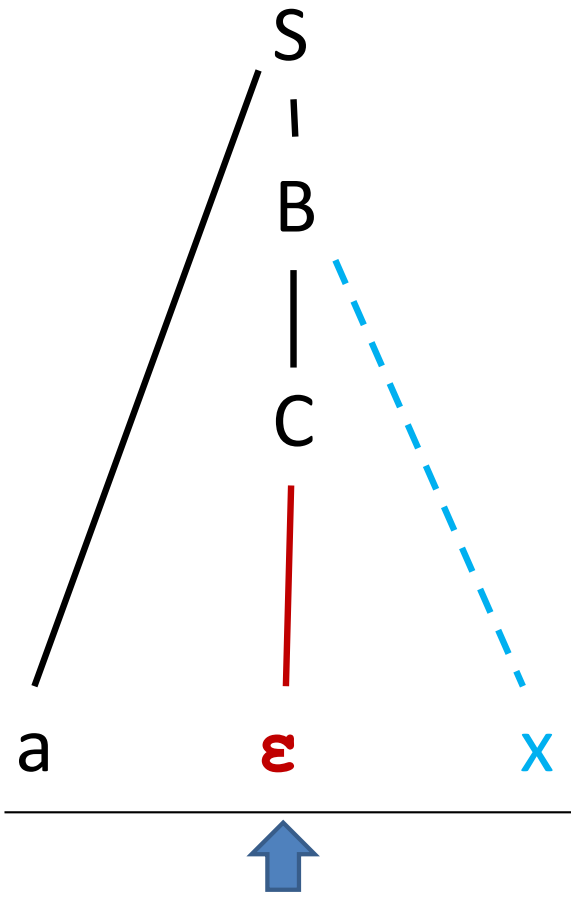
2.  $C ::= \epsilon \mid \mathbf{x}$

Lookahead Remaining

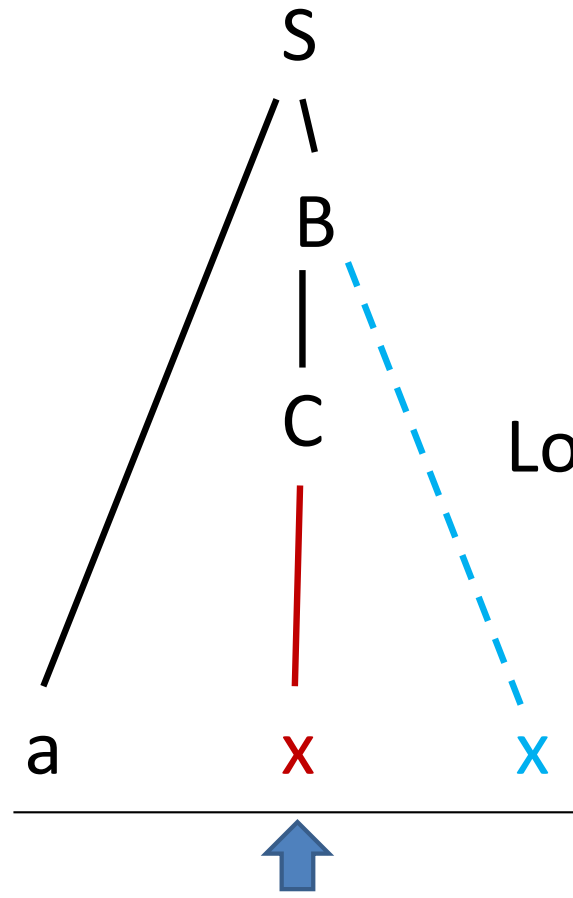
$x$

# Top down derivation of "ax"

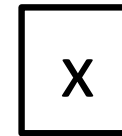
- 0.  $S ::= a B$
- 1.  $B ::= C x \mid y$
- 2.  $C ::= \epsilon \mid x$



OR



Lookahead Remaining



# Applying the Fix: Substitute the Common Prefix,

1

0.  $S ::= a B$

1.  $B ::= x \mid xx \mid y$

~~2.  $C ::= \epsilon \mid x$~~

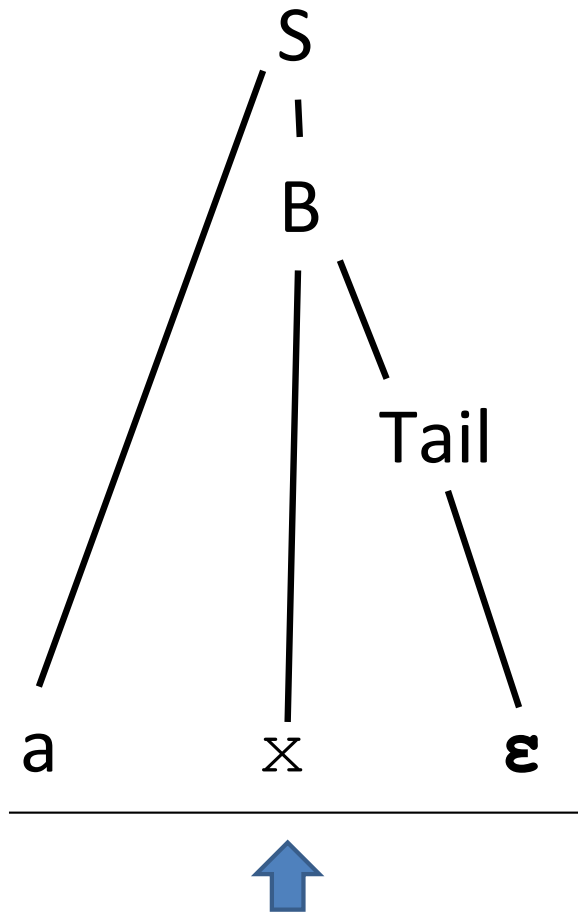
2

0.  $S ::= a B$

1.  $B ::= x Tail \mid y$

2.  $Tail ::= x \mid \epsilon$

# Top down derivation of "ax"



0.  $S ::= a B$

1.  $B ::= x Tail \mid y$

2.  $Tail ::= x \mid \epsilon$

Lookahead Remaining

$x$