


Section 5: More on LL Grammars + Interpreters

Anand, Gavin, Yukai

Adapted from Autumn 2020

Announcements

- Parser + AST due **tonight!**
- Homework 3 (LL grammars) due next Thursday
 - Please start early – the programming portion may take a while!

14:30-15:20 Lecture zoom link Semantics; Attribute grammars (4.3) 26	27	14:30-15:20 Lecture zoom link Attribute grammars (examples) 28	<div style="text-align: center;">  </div> Section Interpreters; more about LL parsing 29 23:00 Project: parser+AST due	14:30-15:20 Lecture zoom link Symbol tables and representation of types 30
--	----	--	--	--

May				
Monday	Tuesday	Wednesday	Thursday	Friday
14:30-15:20 Lecture zoom link x86-64 (everything you forgot from 351) 03	04	14:30-15:20 Lecture zoom link x86-64 function calls & wrapup; 05	Section ASTs & semantics 06 23:00 hw3 due (LL grammars & parsing)	14:30-15:20 Lecture zoom link Code shape I - basics 07

L L (1)



Left-to-Right
Only takes one pass,
performed from the left

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

1 Terminal Lookahead
Must determine derivation
from the next unparsed
terminal in the string


Agenda


- LL Grammar Issues
 - **Review FIRST and FIRST FOLLOW Conflicts**
 - Recursion
 - Indirect Left Recursion
- Interpreters vs Compilers

LL Condition Review


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

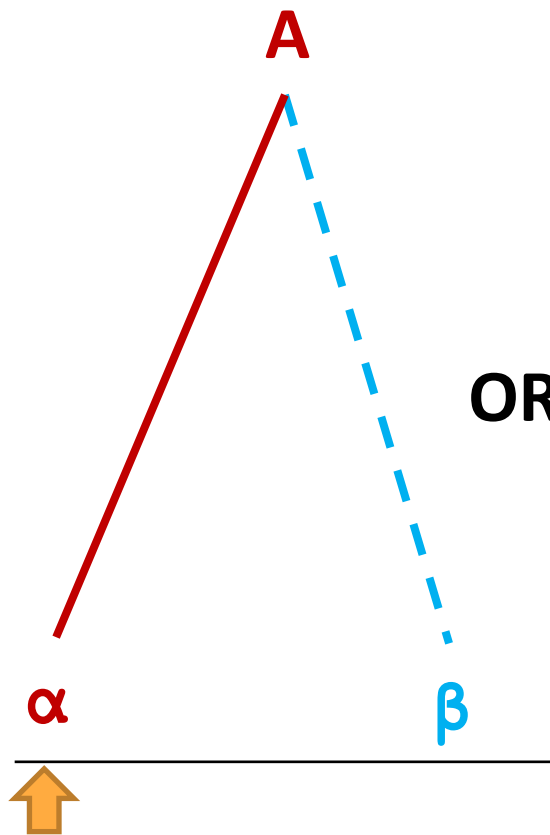
Problem

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

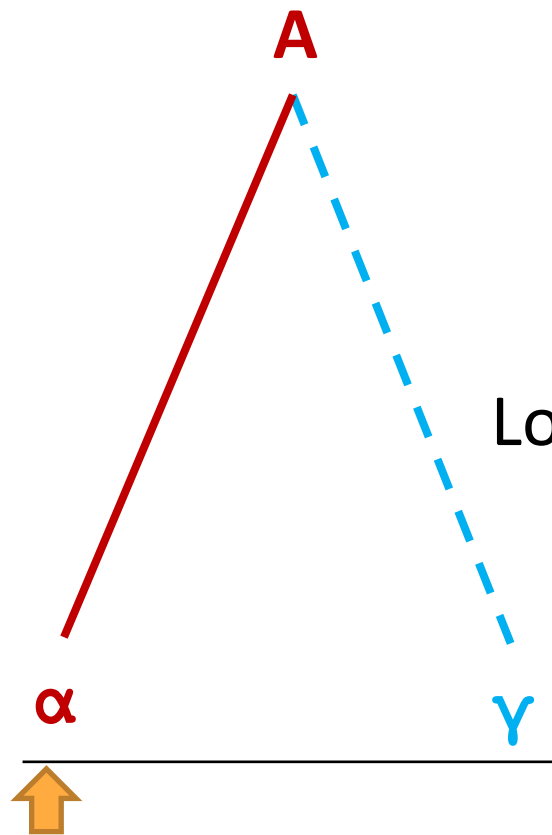
The FIRST sets of the right-hand sides for the **SAME NON-TERMINAL** must be disjoint!

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

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



OR



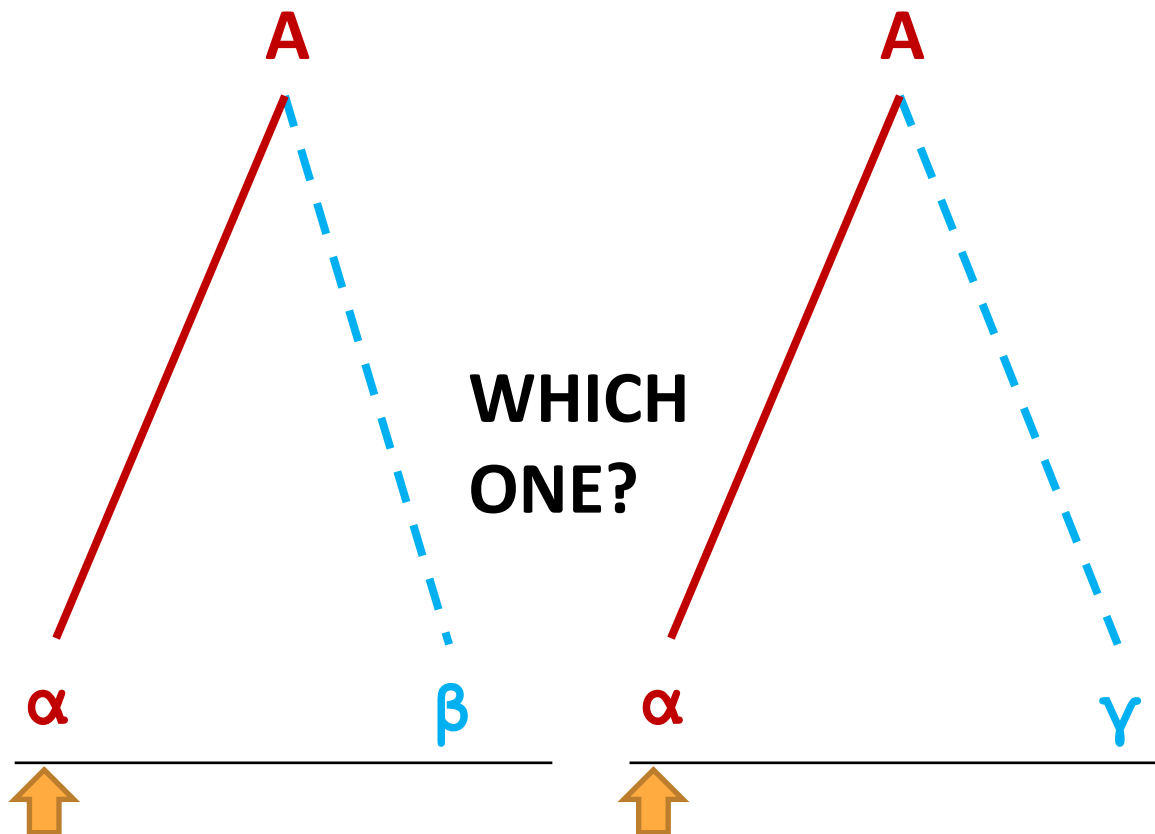
Lookahead

α

Remaining

β

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 α !

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.

Canonical FIRST FOLLOW Conflict

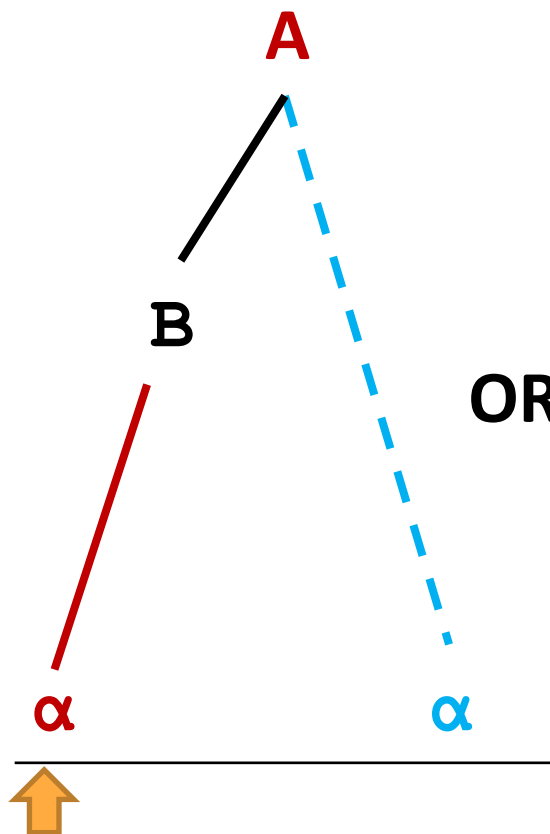
Problem

0. $A ::= B \alpha$

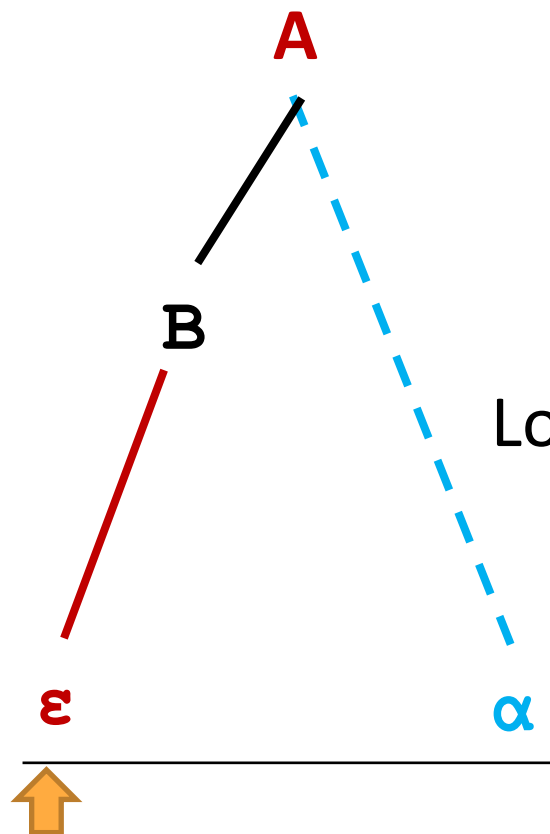
1. $B ::= \alpha \mid \epsilon$

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 " α "



OR



0. $A ::= B \alpha$

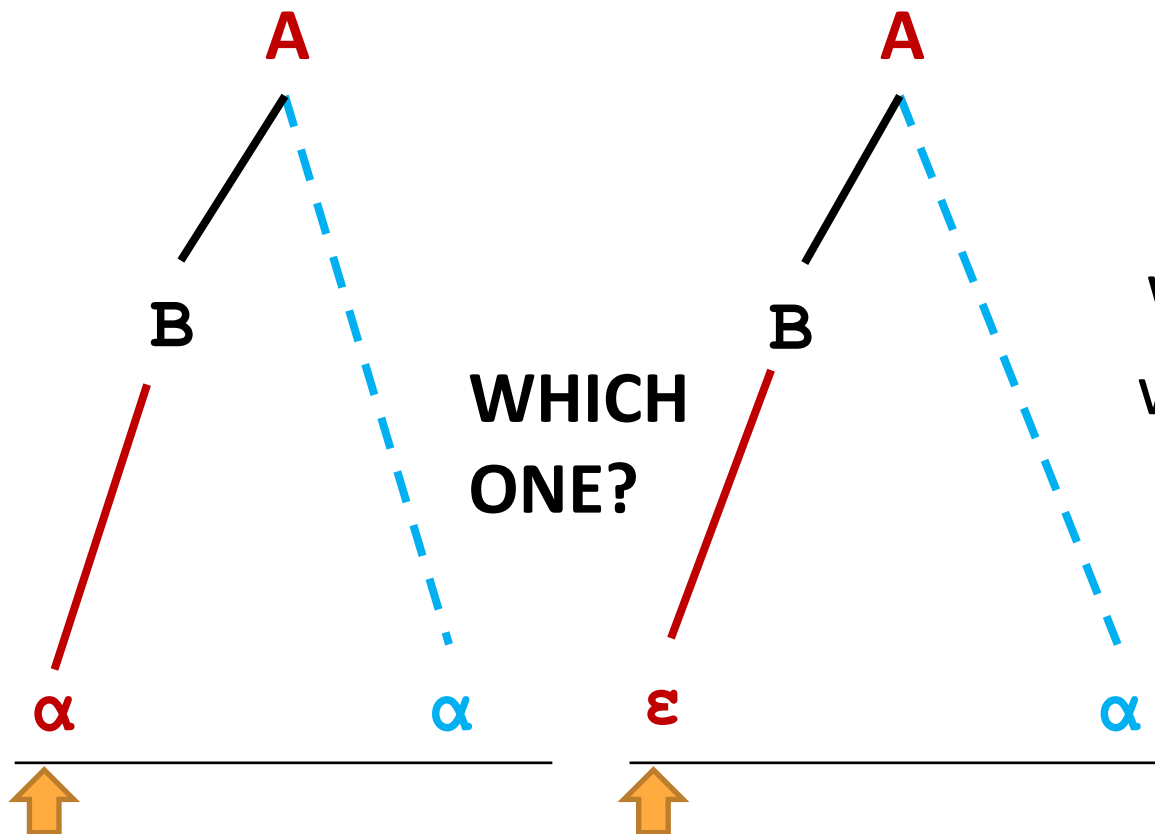
1. $B ::= \alpha \mid \epsilon$

Lookahead

Remaining

α

Let's try a top-down derivation of " α "



0. $A ::= B \alpha$

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

We don't know! Again,
we can't see more than
 α !

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

Agenda

- LL Grammar Issues
 - Review FIRST and FIRST FOLLOW Conflicts
 - **Left Recursion**
 - Indirect Left Recursion
- Interpreters vs Compilers

Classic Left Recursion example problem

$Sum ::= Sum \text{ "+" } Sum \mid Sum \text{ "-" } Sum \mid Constant$

$Constant ::= \text{"1"} \mid \text{"2"} \mid \text{"3"} \mid \dots$

$Sum ::= Constant SumTail$

$SumTail ::= \text{"+" } Constant SumTail \mid \text{"-" } Constant SumTail \mid \epsilon$

$Constant ::= \text{"1"} \mid \text{"2"} \mid \text{"3"} \mid \dots$

Canonical Left Recursion Problem

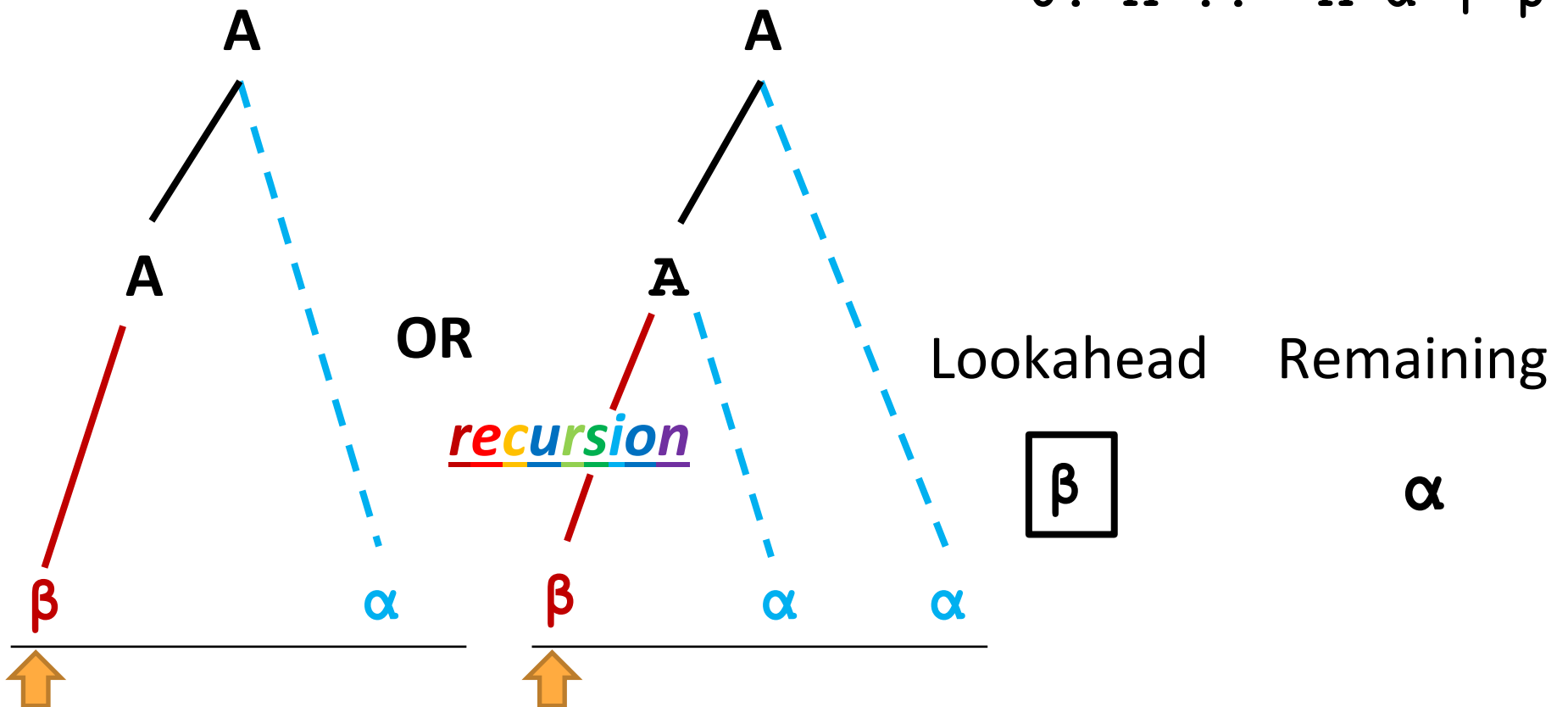
Problem

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

Left recursion can't be parsed by LL(1) parsers!

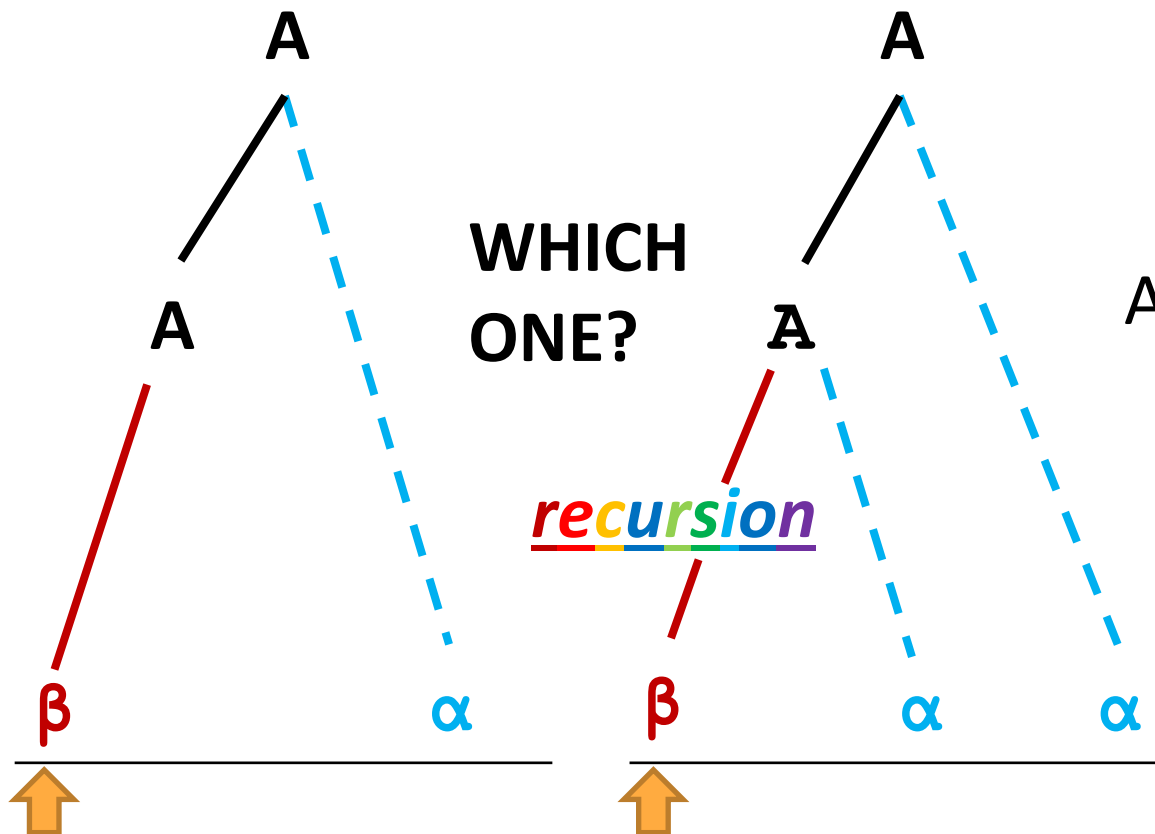
Let's try a top-down derivation of " $\beta\alpha$ "

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



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

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



All we know right now is that it starts with β !

Canonical Left Recursion Solution

Solution

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

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

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

****When a nonterminal has left-recursive productions, turn the different suffixes into a new nonterminal, appended to the remaining productions:**

1. Turn suffix of all A's recursive rhs into a tail non-terminal.
2. Append the tail to every non-recursive rhs.
3. Add the empty string (ϵ) as a rhs for the tail production.

Let's change the grammar again! (Grammar 3)

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

1. $B ::= Cx \mid y$

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

Lookahead

Remaining

a

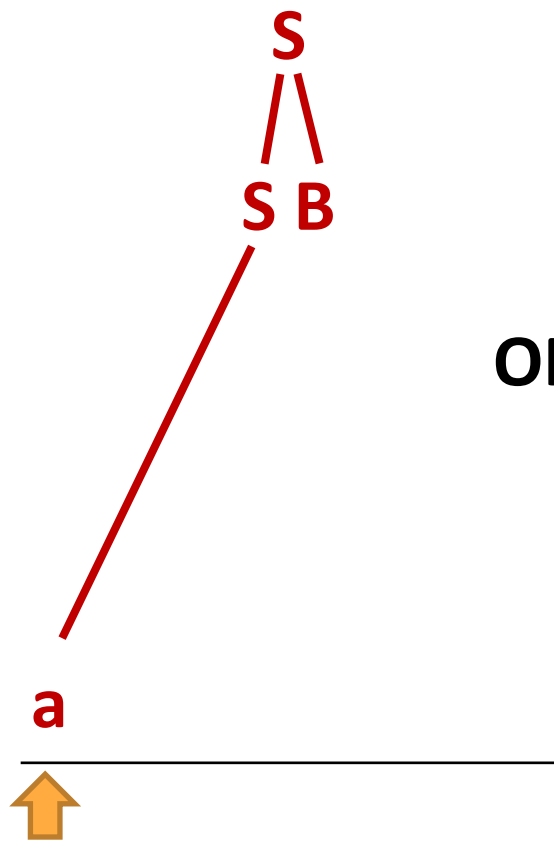
zx

What's the issue?

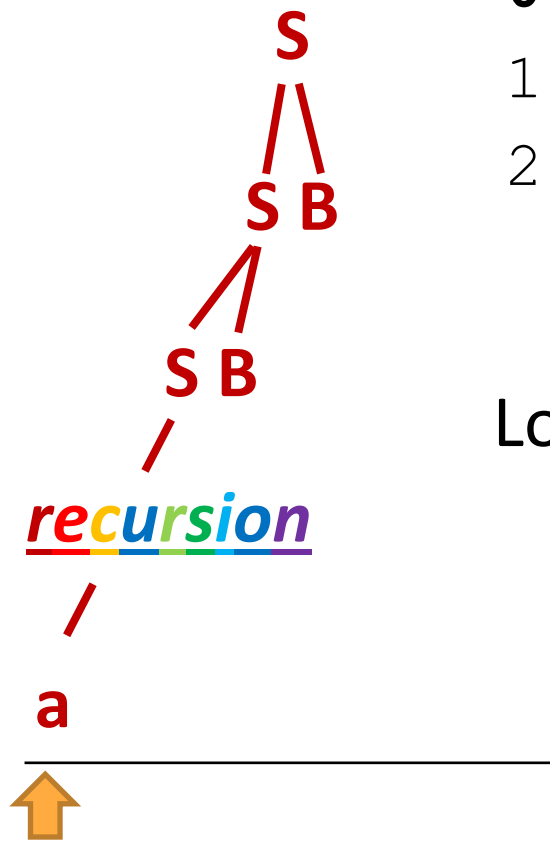
0. $S ::= \mathbf{S} B \mid a \mid w$
1. $B ::= C x \mid y$
2. $C ::= \varepsilon \mid z$

Left recursion can't be parsed by LL(1) parsers!

Top-Down Derivation of "a z x"



OR



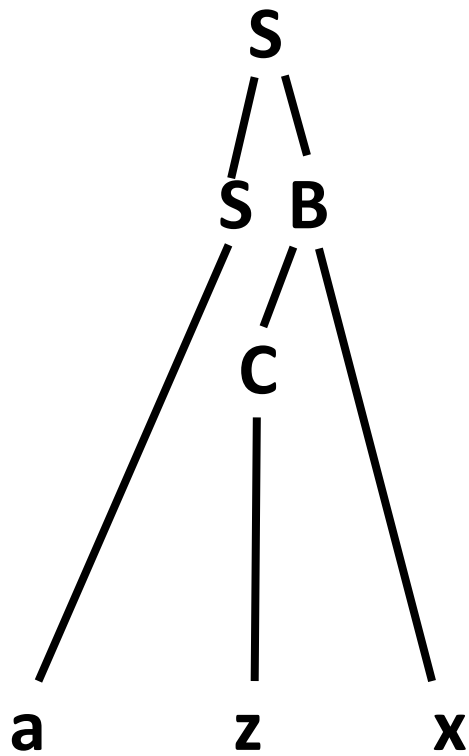
- 0. **S** ::= **S B** | **a** | **w**
- 1. **B** ::= **C x** | **y**
- 2. **C** ::= ϵ | **z**

Lookahead Remaining

a

z x

Parse Tree without changing Grammar



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

Applying the fix:

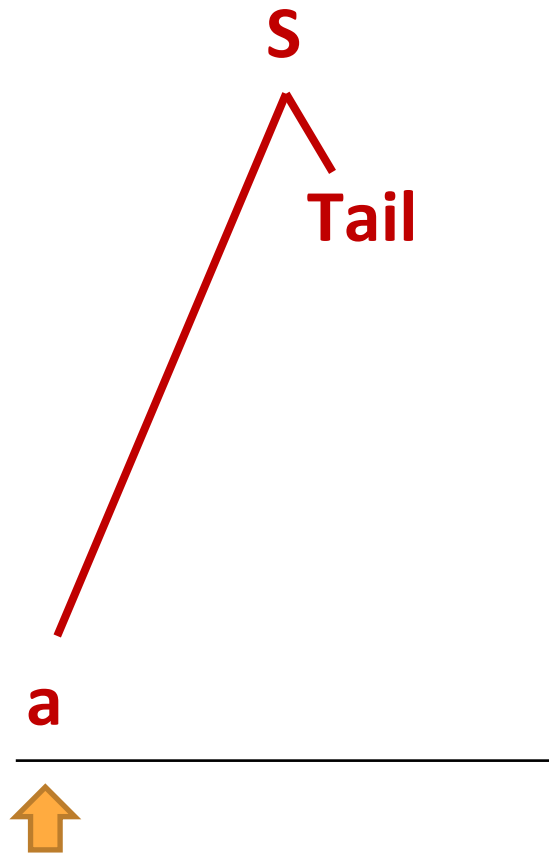
0. $S ::= a \text{ Tail} \mid w \text{ Tail}$

1. $\text{Tail} ::= B \text{ Tail} \mid \varepsilon$

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

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

Top-Down Derivation of "a z x"



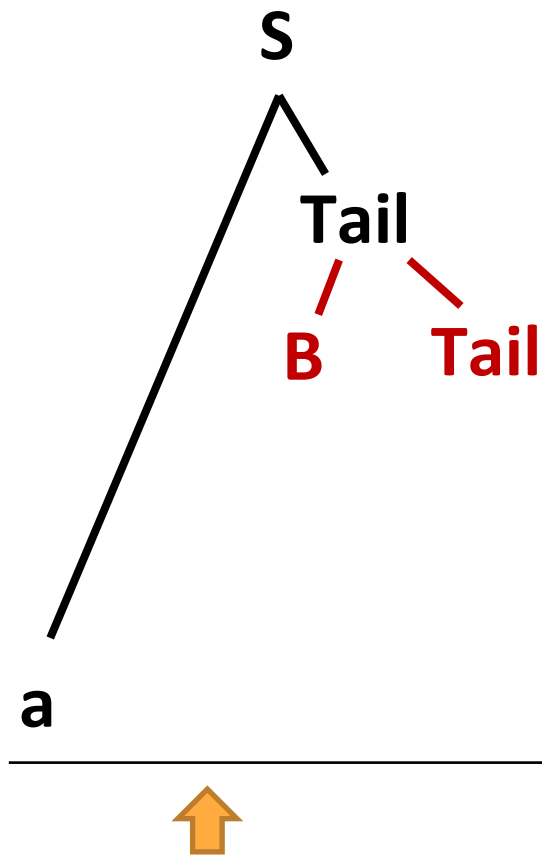
0. $S ::= a \text{ Tail} \mid w \text{ Tail}$
1. $\text{Tail} ::= B \text{ Tail} \mid \varepsilon$
2. $B ::= C x \mid y$
3. $C ::= \varepsilon \mid z$

Lookahead Remaining

a

z x

Top-Down Derivation of "a z x"



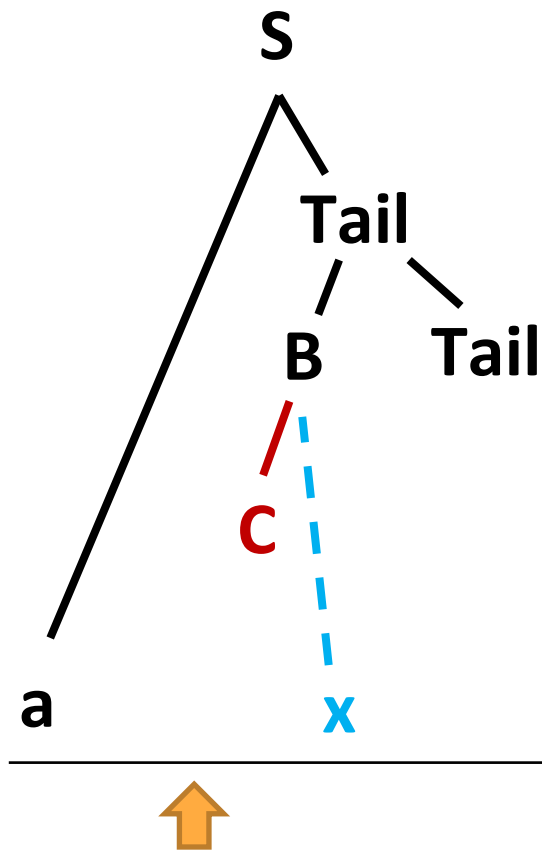
0. $S ::= a \text{ Tail} \mid w \text{ Tail}$
1. $\text{Tail} ::= B \text{ Tail} \mid \varepsilon$
2. $B ::= C x \mid y$
3. $C ::= \varepsilon \mid z$

Lookahead Remaining

z

x

Top-Down Derivation of "a z x"



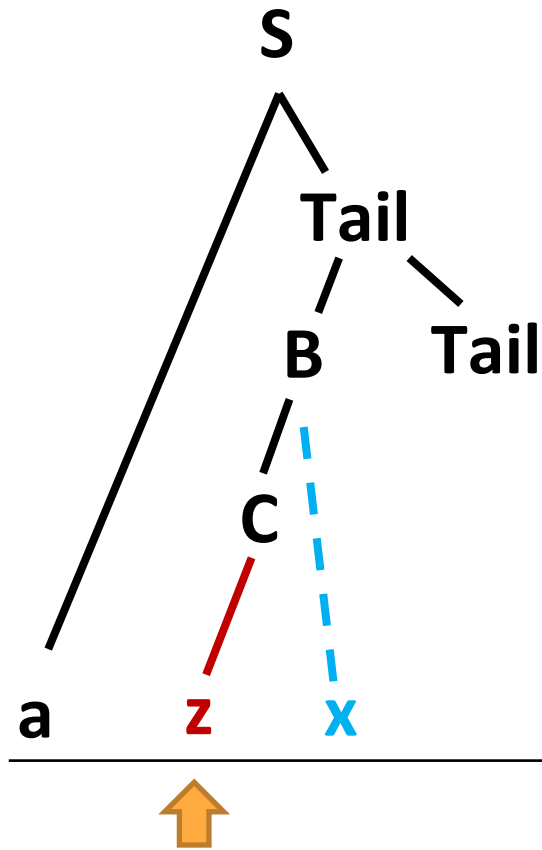
0. $S ::= a \text{ Tail} \mid w \text{ Tail}$
1. $\text{Tail} ::= B \text{ Tail} \mid \varepsilon$
2. $B ::= C x \mid y$
3. $C ::= \varepsilon \mid z$

Lookahead Remaining

z

x

Top-Down Derivation of "a z x"



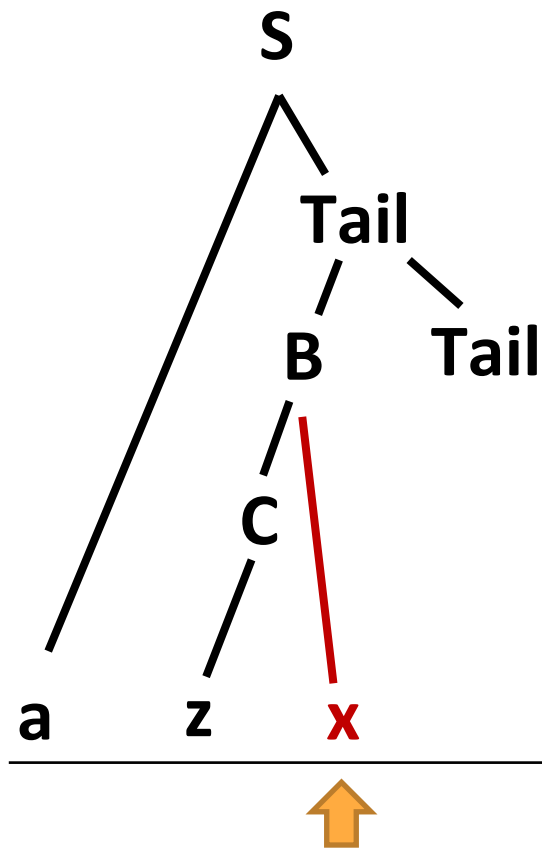
0. $S ::= a \text{ Tail} \mid w \text{ Tail}$
1. $\text{Tail} ::= B \text{ Tail} \mid \varepsilon$
2. $B ::= C x \mid y$
3. $C ::= \varepsilon \mid z$

Lookahead Remaining

z

x

Top-Down Derivation of "a z x"

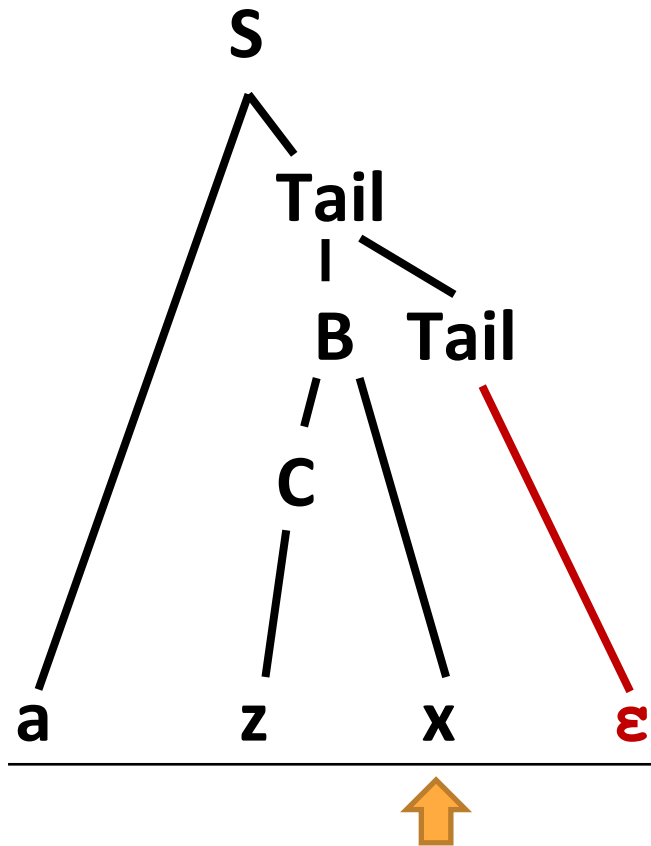


0. $S ::= a \text{ Tail} \mid w \text{ Tail}$
1. $\text{Tail} ::= B \text{ Tail} \mid \varepsilon$
2. $B ::= C x \mid y$
3. $C ::= \varepsilon \mid z$

Lookahead Remaining

x

Top-Down Derivation of "a z x"

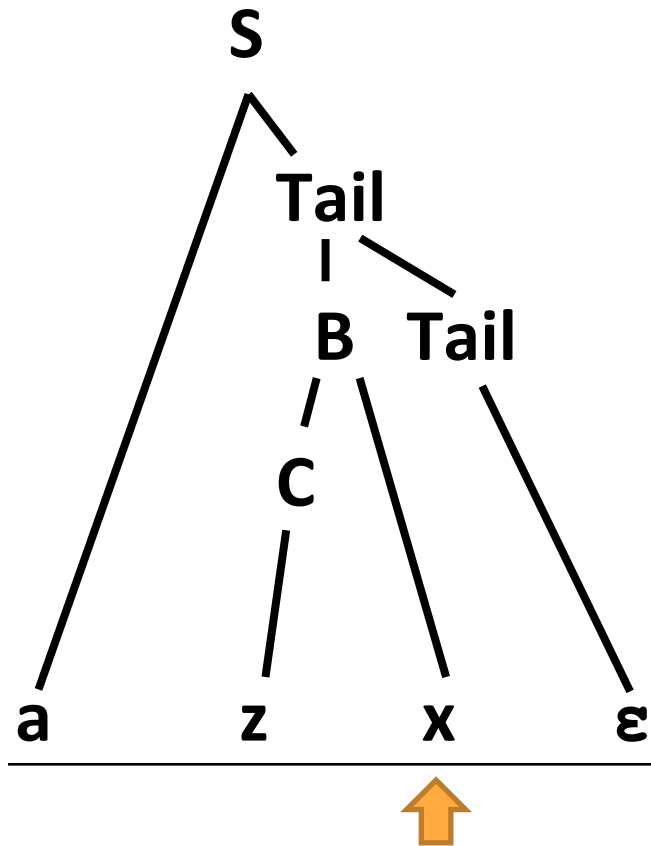


0. $S ::= a \text{ Tail} \mid w \text{ Tail}$
1. $\text{Tail} ::= B \text{ Tail} \mid \epsilon$
2. $B ::= C x \mid y$
3. $C ::= \epsilon \mid z$

Lookahead Remaining



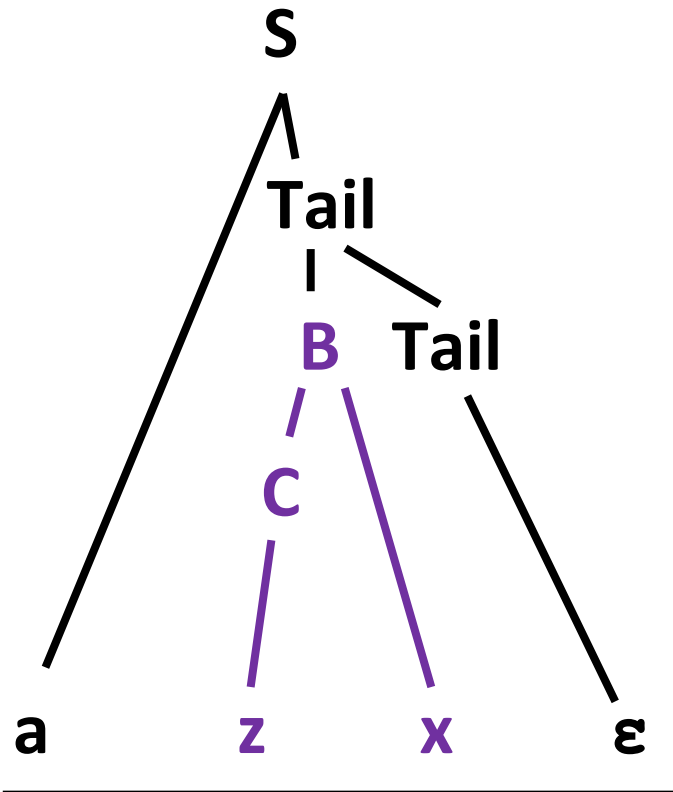
Top-Down Derivation of "a z x"



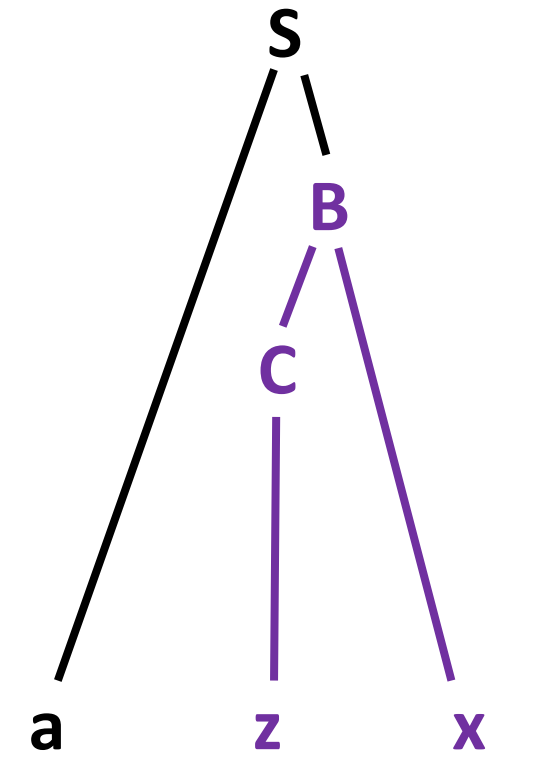
0. $S ::= a \text{ Tail} \mid w \text{ Tail}$
1. $\text{Tail} ::= B \text{ Tail} \mid \epsilon$
2. $B ::= C x \mid y$
3. $C ::= \epsilon \mid z$

Success!

Top-Down Derivation of "a z x"



Purple trees are the same again!



Agenda

- LL Grammar Issues
 - Review FIRST and FIRST FOLLOW Conflicts
 - Recursion
 - **Indirect Left Recursion**
- Interpreters vs Compilers

Classic Indirect Left Recursion Example:

$Expr ::= Ternary \mid Addition$

$Ternary ::= Expr \text{ “?” } Expr \text{ “:” } Expr$

$Addition ::= Expr \text{ “+” } Expr$

$Expr ::= Expr \text{ “?” } Expr \text{ “:” } Expr \mid Expr \text{ “+” } Expr$

Canonical Indirect Left Recursion Problem

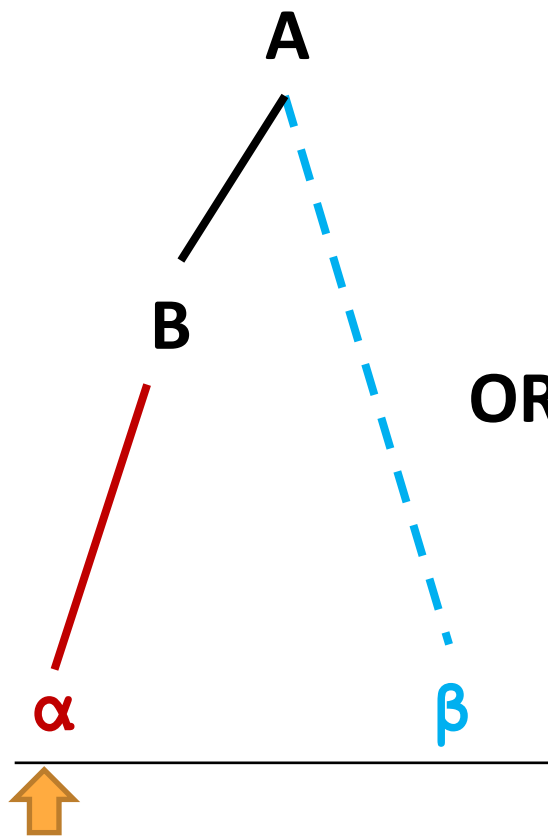
Problem

0. $A ::= B \beta$

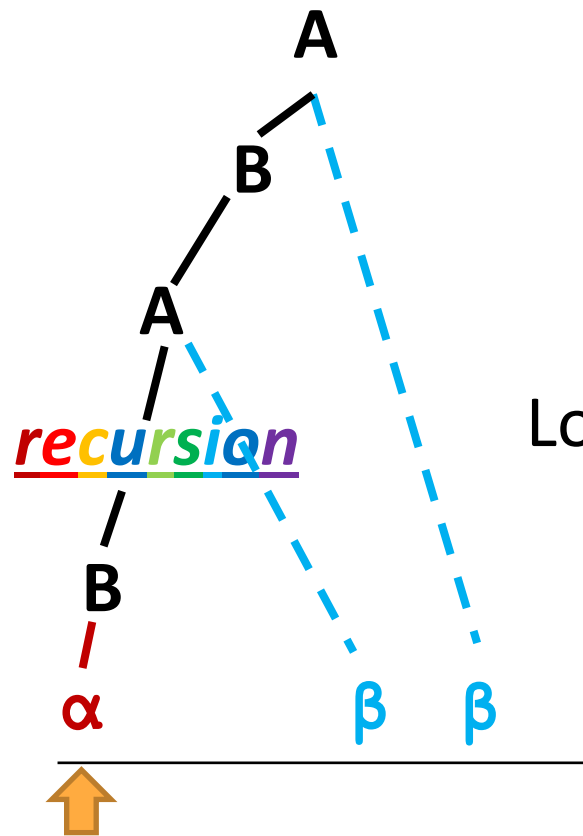
1. $B ::= A \mid \alpha$

Indirect Left recursion can't be parsed by LL(1)
parsers either

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



OR



0. $A ::= B \beta$

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

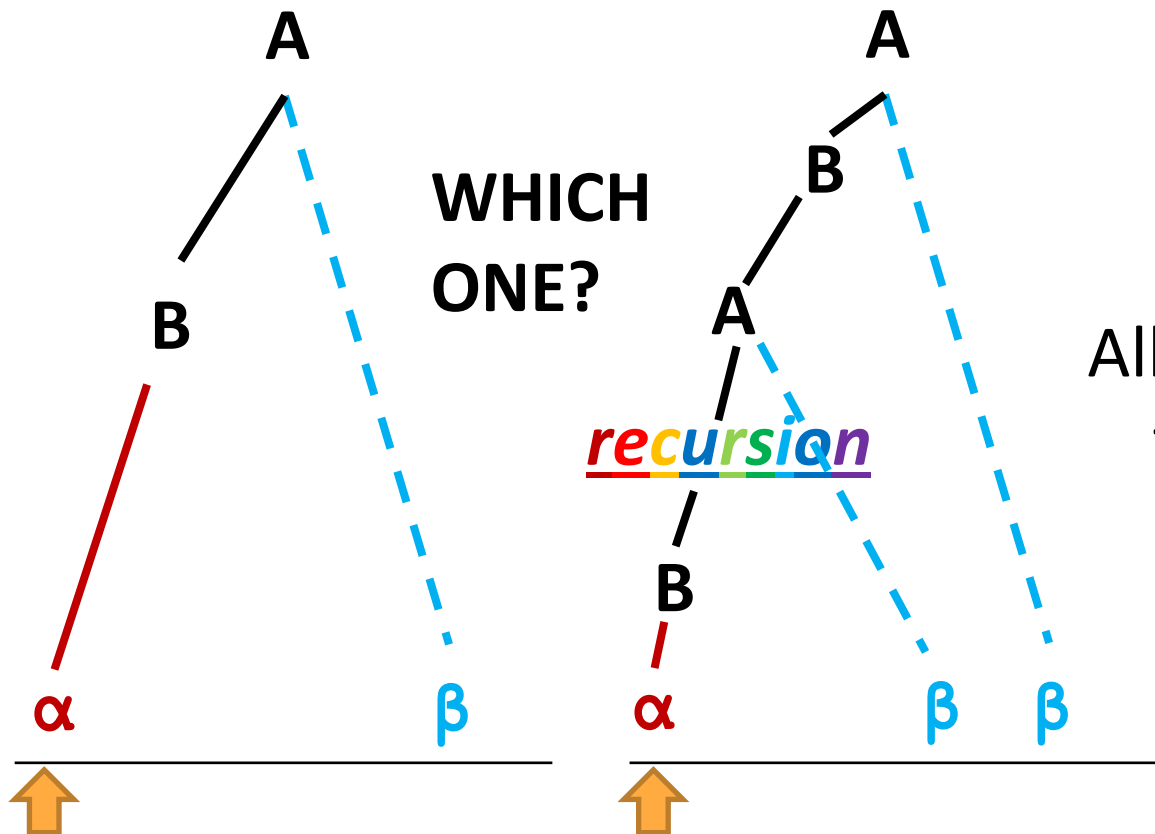
Lookahead

Remaining

α

β

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



0. $A ::= B \beta$
1. $B ::= A \mid \alpha$

All we know right now is that it starts with α !

Canonical Indirect Left Recursion Solution

Solution

0. $A ::= B \beta$

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

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

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

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

1. Substitute B into A:

- When a nonterminal has another nonterminal (B) on the left of a production, rewrite that production to use all possible expansions of B.
- Repeat until the left side of every production is a terminal or direct left recursion.

2. Fix via typical Left Recursion Solution

Note: **NEVER** remove the starting non-terminal!
The start symbol of the new grammar should be the same

Worksheet & Interpreter Demo

Indirect Left Recursion

Changing the grammar for the last time...

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

1. $B ::= S \mid z x$

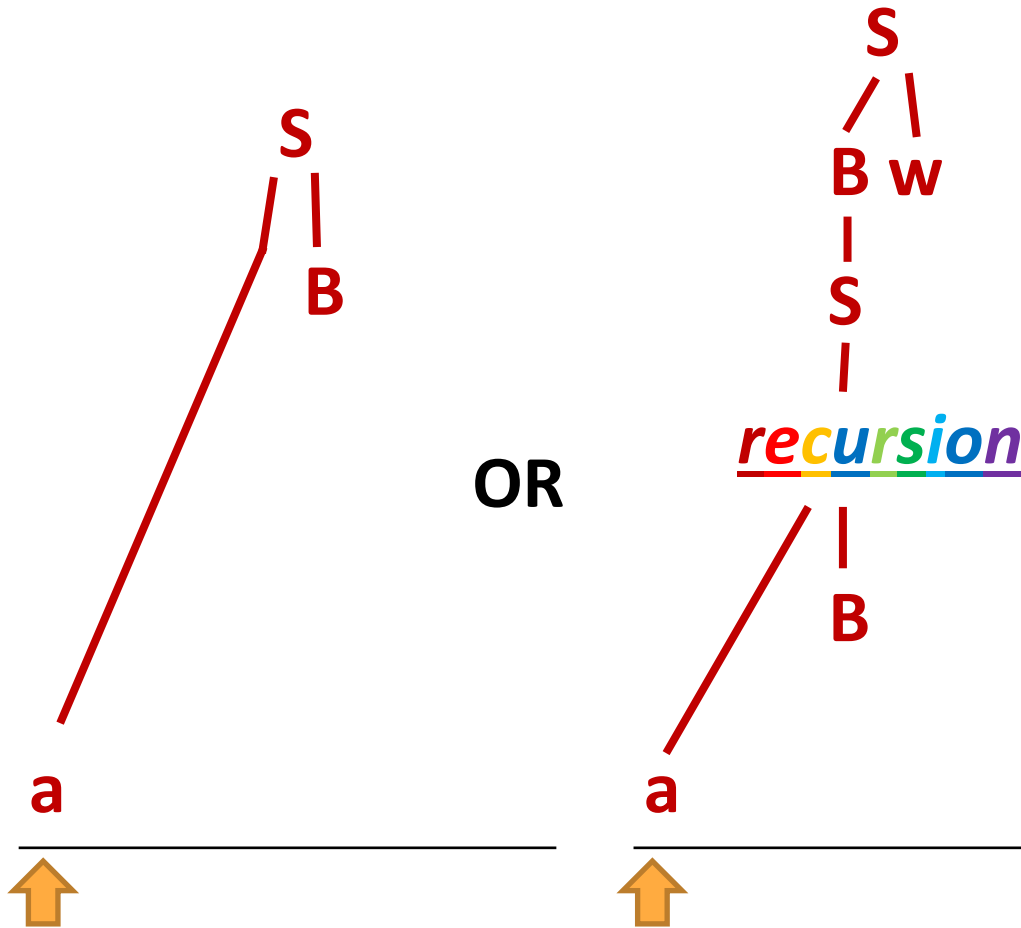
Lookahead

a

Remaining

z x

Indirect Left Recursion



OR

- 0. $S ::= B w \mid a B$
- 1. $B ::= S \mid z x$

Lookahead

Remaining

a

z x

Applying the fix: Substitute, then eliminate left recursion!

1 0. $S ::= S w \mid z x w \mid a S \mid a z x$
 ~~1. $B ::= S \mid z x$~~

2 0. $S ::= z x w T \mid a S T \mid a z x T$
 1. $T ::= w T \mid \epsilon$

3 0. $S ::= z x w T \mid a aT$
 1. $aT ::= S T \mid z x T$
 2. $T ::= w T \mid \epsilon$

Something else is wrong!
What is it?

Eliminating FIRST CONFLICT

4: sub S into
the S in aT

$$0. S ::= z x w T \mid a aT$$

$$1. aT ::= z x w T T \mid a aT T \mid z x T$$

$$2. T ::= w T \mid \varepsilon$$

5: Add a tail
to zx as it
appears twice
in (4)

$$0. S ::= z x w T \mid a aT$$

$$1. aT ::= z x zT \mid a aT T$$

$$2. zT ::= w T T \mid T$$

$$3. T ::= w T \mid \varepsilon$$

Eliminating FIRST CONFLICT

6: Sub rule
3 into the T
in rule 2.

0. $S ::= z x w T \mid a aT$
1. $aT ::= z x zT \mid a aT T$
2. $zT ::= w T T \mid w T \mid \epsilon$
3. $T ::= w T \mid \epsilon$

7: Notice in
rule 2, $w T T$ is
the same as $w T$,
since it's
recursively
appending w 's on
the right. Get rid
of $w T T$.

0. $S ::= z x w T \mid a aT$
1. $aT ::= z x zT \mid a aT T$
2. $zT ::= w T \mid \epsilon$
3. $T ::= w T \mid \epsilon$

Eliminating FIRST CONFLICT

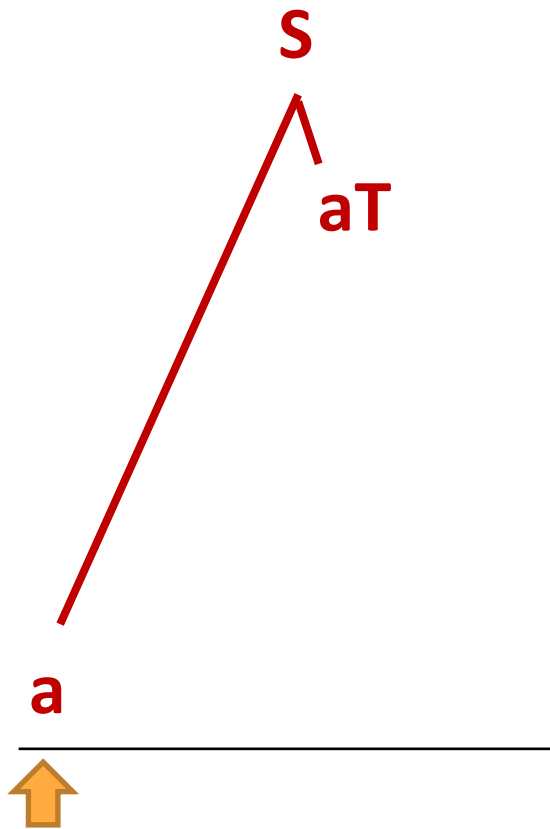
8: zT has been deleted because it was the same production as T . zT was replaced by T .

0. $S ::= z x w T \mid a aT$

1. $aT ::= z x T \mid a aT T$

2. $T ::= w T \mid \epsilon$

Top-down derivation of "azx"



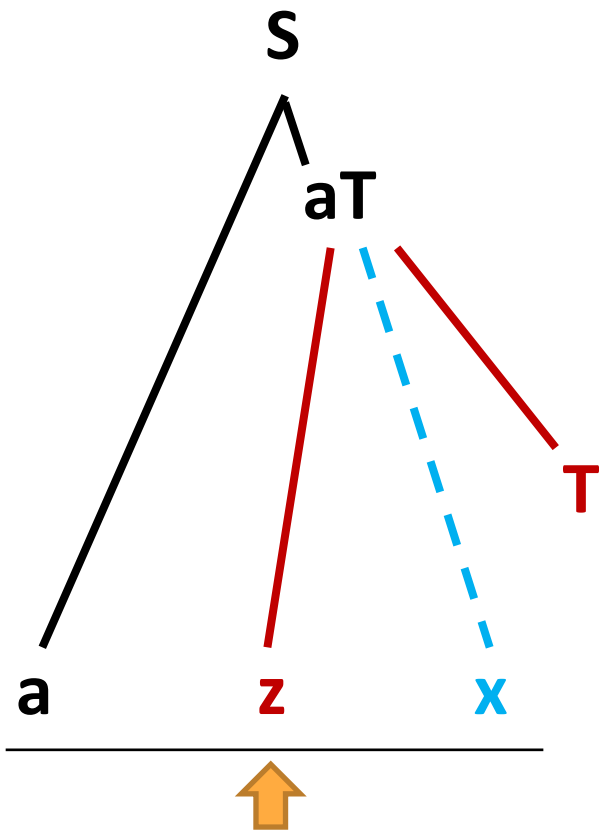
0. $S ::= z x w T \mid a aT$
1. $aT ::= z x T \mid a aT T$
2. $T ::= w T \mid \epsilon$

Lookahead Remaining

a

z x

Top-down derivation of "azx"



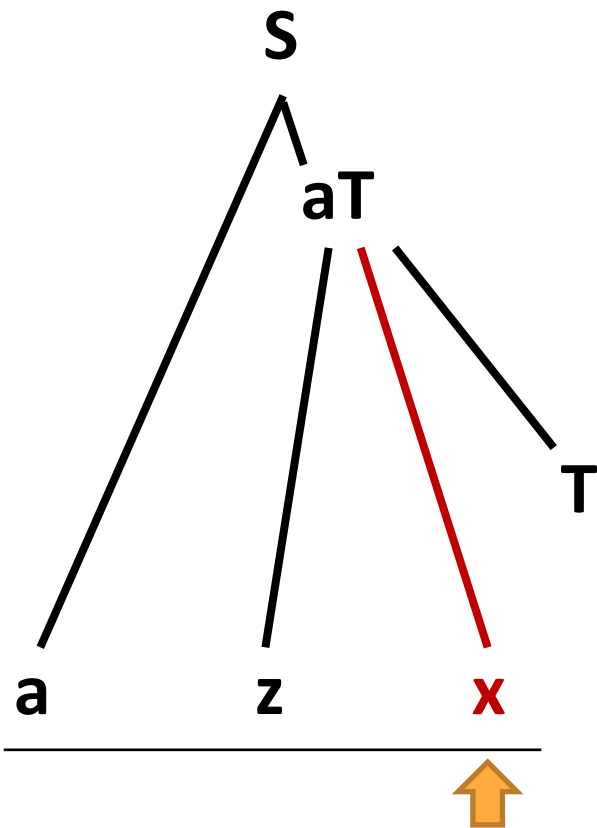
0. $S ::= z x w T \mid a aT$
1. $aT ::= z x T \mid a aT T$
2. $T ::= w T \mid \epsilon$

Lookahead Remaining

z

x

Top-down derivation of "azx"

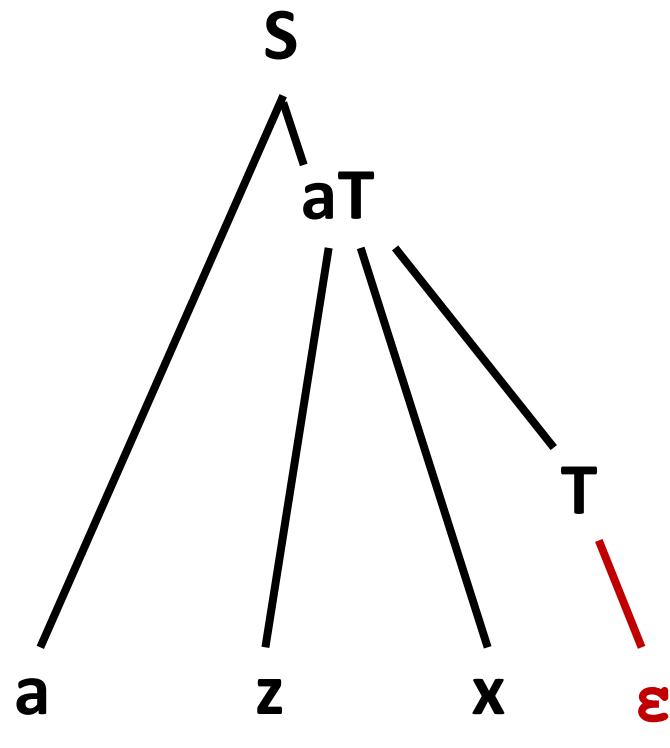


0. $S ::= z x w T \mid a aT$
1. $aT ::= z x T \mid a aT T$
2. $T ::= w T \mid \epsilon$

Lookahead Remaining

x

Top-down derivation of "azx"



0. $S ::= z x w T \mid a aT$
1. $aT ::= z x T \mid a aT T$
2. $T ::= w T \mid \epsilon$

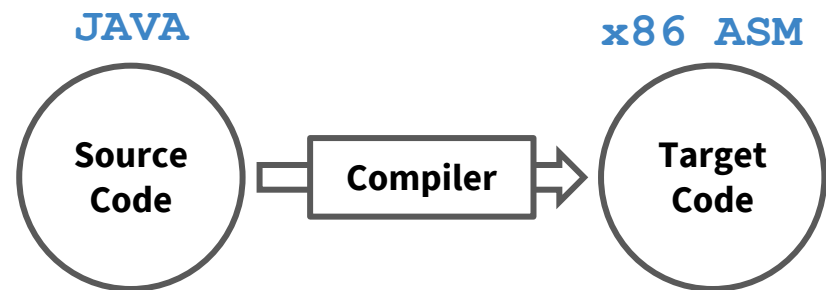
Success!

Agenda

- LL Grammar Issues
 - Review FIRST and FIRST FOLLOW Conflicts
 - Recursion
 - Indirect Left Recursion
- **Interpreters vs Compilers**

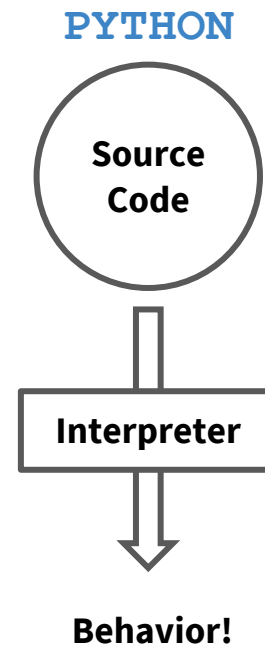
Interpreters vs. Compilers

- Compilers
 - Translate between different languages
 - e.g. MiniJava \Rightarrow x86 ASM
 - e.g. Java \Rightarrow Java Byte Code



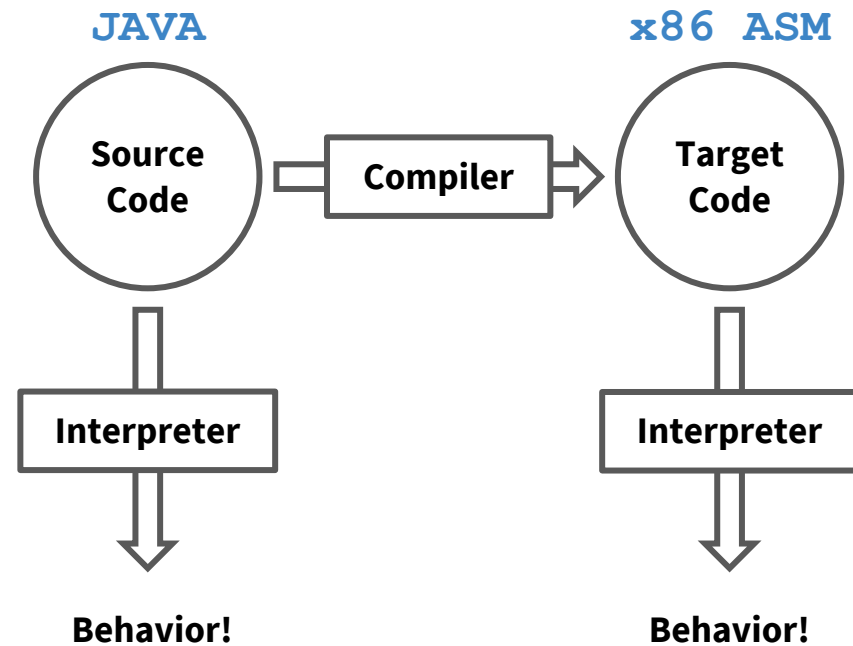
Interpreters vs. Compilers

- Compilers
 - Translate between different languages
 - e.g. MiniJava \Rightarrow x86 ASM
 - e.g. Java \Rightarrow Java Byte Code
- Interpreters
 - Take action upon a piece of code as it is read



Interpreters vs. Compilers

- Compilers
 - Translate between different languages
 - e.g. MiniJava \Rightarrow x86 ASM
 - e.g. Java \Rightarrow Java Byte Code
- Interpreters
 - Take action upon a piece of code as it is read



Implementation

- Frontend is the same!
- Interpreters
 - Execute the AST
- Compilers
 - Translate the AST

