Context-free Grammars Parse Trees and Derivations

CSE 340 Spring 2021

Rida A. Bazzi

Context-free grammar

A context-free grammar consists of

Context-free grammar

A context-free grammar consists of

- A finite set NT of symbols called non-terminals

Context-free grammar

A context-free grammar consists of

- A finite set NT of symbols called non-terminals
- A finite set T of symbols called terminals (the tokens are the terminals)
- A start symbol which is a symbol from the set NT

Context-free grammar

A context-free grammar consists of

- A finite set NT of symbols called non-terminals
- A finite set T of symbols called terminals (the tokens are the terminals)
- A start symbol which is a symbol from the set NT
- A finite set of rules. Every rule has a left hand side (LHS) and a right hand side (RHS)

Context-free grammar

A context-free grammar consists of

- A finite set NT of symbols called non-terminals
- A finite set T of symbols called terminals (the tokens are the terminals)
- A start symbol which is a symbol from the set NT
- A finite set of rules. Every rule has a left hand side (LHS) and a right hand side (RHS)
  - LHS: is a non-terminal (an element of NT)
  - RHS: a sequence of symbols from T and NT : an element of (T U NT)*

Simplifying the notation

Instead of writing the set of terminals, the set of non-terminals, the start symbol and the rules in details, we simply list the rules with left hand side and right hand side separated by an arrow

Simplifying the notation

Instead of writing the set of terminals, the set of non-terminals, the start symbol and the rules in details, we simply list the rules with left hand side and right hand side separated by an arrow

Unless otherwise specified, the start symbol is the LHS of the first rule

Simplifying the notation

Instead of writing the set of terminals, the set of non-terminals, the start symbol and the rules in details, we simply list the rules with left hand side and right hand side separated by an arrow

Unless otherwise specified, the start symbol is the LHS of the first rule

The non-terminals are the LHS of rules

Simplifying the notation

Instead of writing the set of terminals, the set of non-terminals, the start symbol and the rules in details, we simply list the rules with left hand side and right hand side separated by an arrow

Unless otherwise specified, the start symbol is the LHS of the first rule

The non-terminals are the LHS of rules

The terminals are the remaining symbols (other than the symbol for epsilon ε)

Context-free grammar example

S → A
S → B
A → a A b
A → c
B → b B d
B → e

Context-free grammar notations

S → A
S → B
A → a A b
A → c
B → b B d
B → e

rule. B is the left hand side of the rule
b B d is the right hand side of the rule

This grammar has 6 rules

Context-free grammar notations

S → A
S → B
A → a A b
A → c
B → b B d
B → e

unless otherwise specified, the left side of the first rule in the list is the start symbol

rule. B is the left hand side of the rule
b B d is the right hand side of the rule

This grammar has 6 rules

Context-free grammar notation

S → A
S → B
A → a A b
A → c
B → b B d
B → e

The symbols that appear on the left side of a rule must be non terminals

The right side of a rule is a sequence of of terminals and non-terminals

T is the set of terminals
NT is the set of non-terminals

Context-free grammar notation

S →	A	The symbols that appear on the left side
S →	B	of a rule must be non terminals
A →	a A b	
A →	c	The right side of a rule is a sequence of
B →	b B d	of terminals and non-terminals
B →	e	

T is the set of terminals
NT is the set of non-terminals

In this example

T = { a , b , c , d , e }
NT = { S, A, B }

Context-free grammar notations

We can rewrite the grammar on the previous slide as follows

$S \rightarrow A \mid B$
$A \rightarrow a\,A\,b \mid c$
$B \rightarrow b\,B\,d \mid e$

Context-free grammar example

$S \rightarrow A \mid B$
$A \rightarrow a\,A\,b \mid c$
$B \rightarrow b\,B\,d \mid e$

$S \rightarrow A \mid B$

is equivalent to

$S \rightarrow A$
$S \rightarrow B$

Context-free grammar example

S → A | B
A → a A b | c
B → b B d | e

S has two rules: (1) S → A and (2) S → B

start symbol

a, b, c, d, and e are terminals

S , A , and , B are non-terminals

Context-free grammar example

NT = { S , A , B }

NT: Non Terminals

T = { a , b , c , d , e }

T: Terminals

Start Symbol = S

	LHS	RHS
rule 1	S	A
rule 2	S	B
rule 3	A	a A b
rule 4	A	c
rule 5	B	b B d
rule 6	B	e

Context-free grammar example

NT = { S , A , B }

NT: Non Terminals

T = { a , b , c , d , e }

T: Terminals

Start Symbol = S

	LHS	RHS
rule 1	S	A
rule 2	S	B
rule 3	A	a A b
rule 4	A	c
rule 5	B	b B d
rule 6	B	e

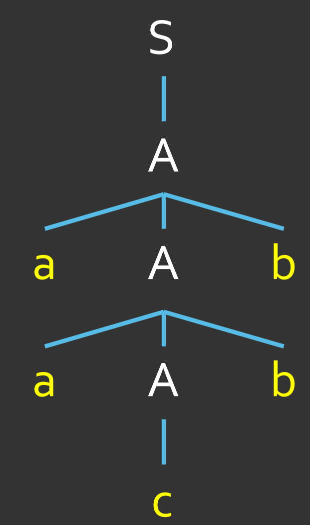
This representation is not convenient for handwritten examples, but it is useful when writing programs that manipulate grammars.

For example, a parser generator reads a grammar and represents it internally to generate a parser automatically

Parse Trees

Given a grammar and an input (sequence of tokens), a parse tree for the input is a **labeled** tree such that

$$
\begin{aligned}
\mathbf{S} &\rightarrow A \mid B \\
A &\rightarrow a\ A\ b \mid c \\
B &\rightarrow b\ B\ d \mid e
\end{aligned}
$$

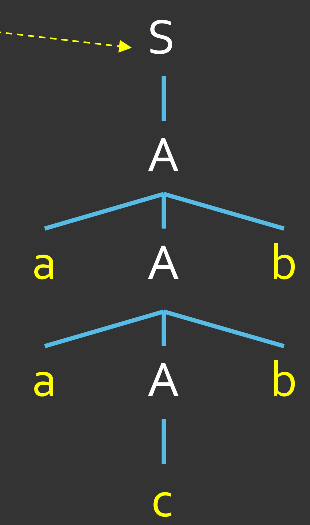


Parse Trees

Given a grammar and an input (sequence of tokens), a parse tree for the input is a **labeled** tree such that

1. The root is labeled with the start symbol

S → A | B
A → a A b | c
B → b B d | e



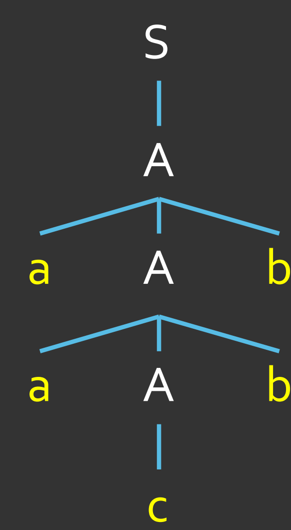
Parse Trees

Given a grammar and an input (sequence of tokens), a parse tree for the input is a **labeled** tree such that

1. The root is labeled with the start symbol

2. If a node is labeled X and its children from left to right are labeled X_1, X_2, ..., and X_k , then $X \rightarrow X_1 X_2 \ldots X_k$ must be a grammar rule

$$\mathbf{S} \rightarrow A \mid B$$
$$A \rightarrow a\ A\ b \mid c$$
$$B \rightarrow b\ B\ d \mid e$$



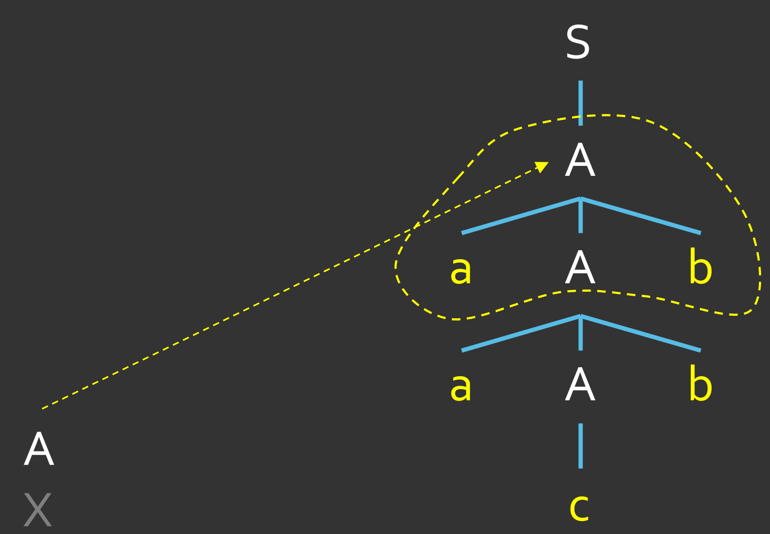
Parse Trees

Given a grammar and an input (sequence of tokens), a parse tree for the input is a **labeled** tree such that

S → A | B
A → a A b | c
B → b B d | e

1. The root is labeled with the start symbol

2. If a node is labeled X,



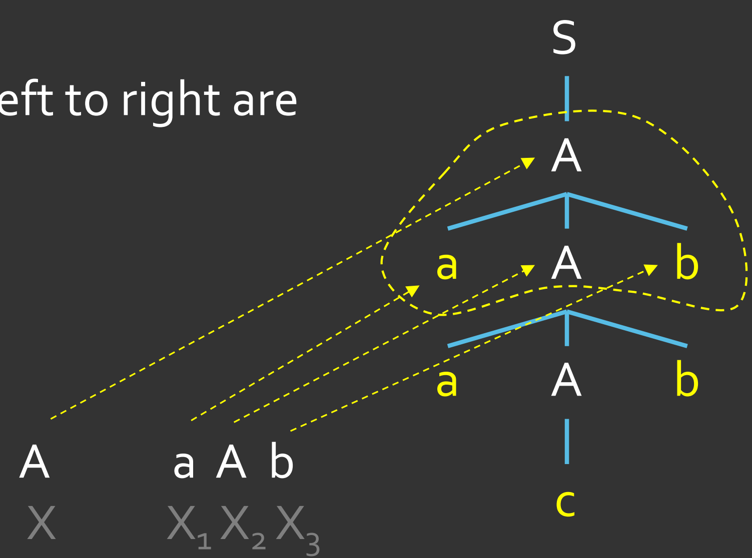
Parse Trees

Given a grammar and an input (sequence of tokens), a parse tree for the input is a **labeled** tree such that

$S \rightarrow A \mid B$
$A \rightarrow a\ A\ b \mid c$
$B \rightarrow b\ B\ d \mid e$

1. The root is labeled with the start symbol

2. If a node is labeled X and its children from left to right are labeled X_1, X_2, ..., and X_k ,



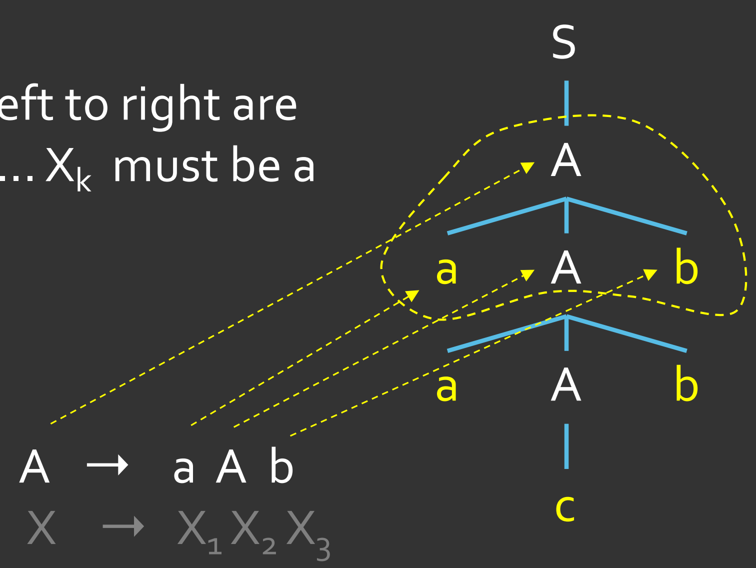
Parse Trees

Given a grammar and an input (sequence of tokens), a parse tree for the input is a **labeled** tree such that

$S \rightarrow A \mid B$
$A \rightarrow a\,A\,b \mid c$
$B \rightarrow b\,B\,d \mid e$

1. The root is labeled with the start symbol

2. If a node is labeled X and its children from left to right are labeled $X_1, X_2, \ldots,$ and X_k , then $X \rightarrow X_1\, X_2 \ldots X_k$ must be a grammar rule



Parse Trees

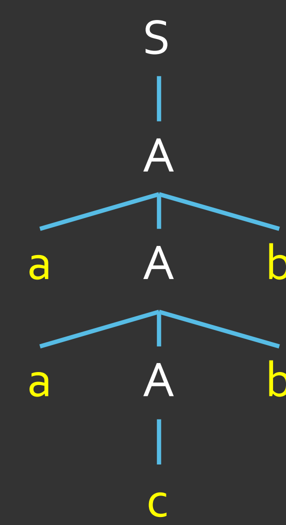
Given a grammar and an input (sequence of tokens), a parse tree for the input is a **labeled** tree such that

1. The root is labeled with the start symbol

2. If a node is labeled X and its children from left to right are labeled X_1, X_2, …, and X_k , then $X \rightarrow X_1\ X_2 \ldots X_k$ must be a grammar rule

3. Leaf nodes are labeled with terminals (tokens) or ε (epsilon)

$$\begin{aligned} \mathbf{S} &\rightarrow A \mid B \\ A &\rightarrow a\ A\ b \mid c \\ B &\rightarrow b\ B\ d \mid e \end{aligned}$$



Parse Trees

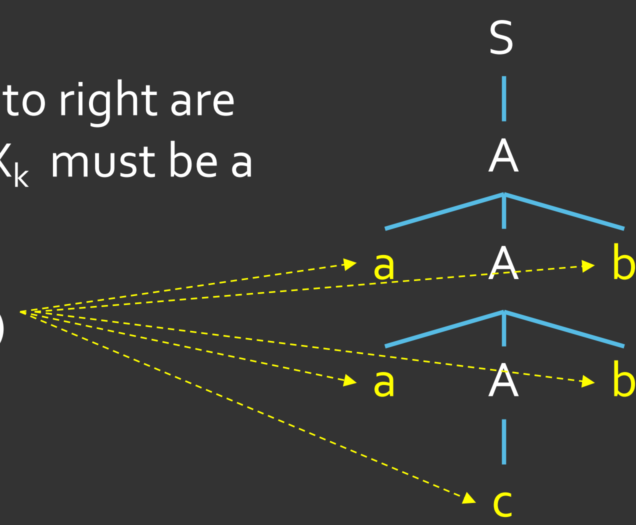
Given a grammar and an input (sequence of tokens), a parse tree for the input is a **labeled** tree such that

$S \rightarrow A \mid B$
$A \rightarrow a\ A\ b \mid c$
$B \rightarrow b\ B\ d \mid e$

1. The root is labeled with the start symbol

2. If a node is labeled X and its children from left to right are labeled $X_1, X_2, \ldots,$ and X_k , then $X \rightarrow X_1\, X_2 \ldots X_k$ must be a grammar rule

3. Leaf nodes are labeled with terminals (tokens) or ε (epsilon)



Parse Trees

Given a grammar and an input (sequence of tokens), a parse tree for the input is a **labeled** tree such that

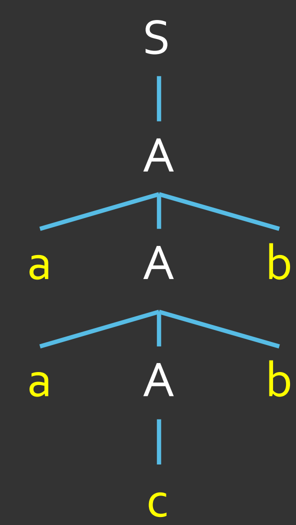
$S \rightarrow A \mid B$
$A \rightarrow a\ A\ b \mid c$
$B \rightarrow b\ B\ d \mid e$

1. The root is labeled with the start symbol

2. If a node is labeled X and its children from left to right are labeled X_1, X_2, …, and X_k , then $X \rightarrow X_1\ X_2 \ldots X_k$ must be a grammar rule

3. Leaf nodes are labeled with terminals (tokens) or ε (epsilon)

4. The input is equal to the sequence of labels of the leaves from left to right



Parse Trees

Given a grammar and an input (sequence of tokens), a parse tree for the input is a **labeled** tree such that

$S \rightarrow A \mid B$
$A \rightarrow a\,A\,b \mid c$
$B \rightarrow b\,B\,d \mid e$

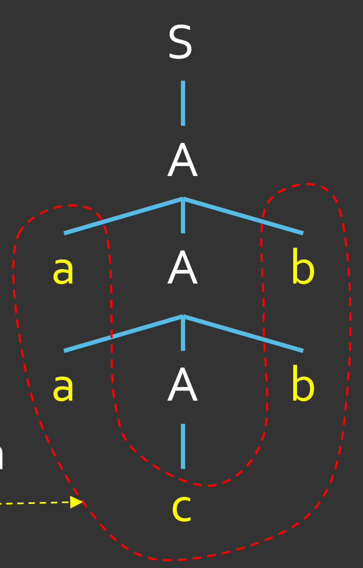
1. The root is labeled with the start symbol

2. If a node is labeled A and its children from left to right are labeled A1, A2, ..., and A_k , then $A \rightarrow A1\ A2 \ldots A_k$ must be a grammar rule

3. Leaf nodes are labeled with terminals (tokens) or ε (epsilon)

4. The input is equal to the sequence of labels of the leaves from left to right

Input aacbb



Parse Trees

Given a grammar and an input (sequence of tokens), a parse tree for the input is a **labeled** tree such that

$S \rightarrow A \mid B$
$A \rightarrow a\,A\,b \mid c$
$B \rightarrow b\,B\,d \mid e$

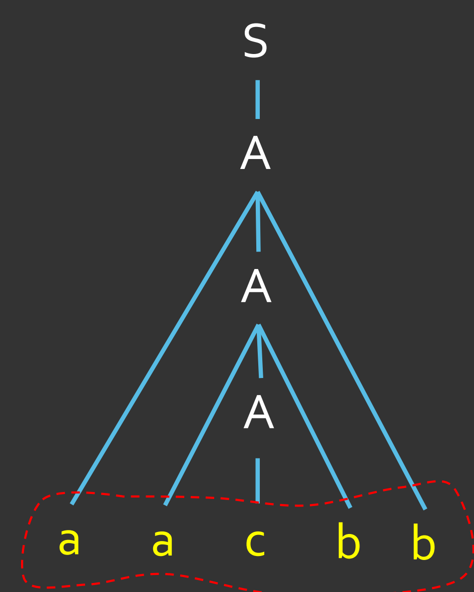
1. The root is labeled with the start symbol

2. If a node is labeled A and its children from left to right are labeled A1, A2, …, and A_k , then $A \rightarrow A1\ A2 \ldots A_k$ must be a grammar rule

3. Leaf nodes are labeled with terminals (tokens) or ε (epsilon)

4. The input is equal to the sequence of labels of the leaves from left to right

Input aacbb



Example Parse tree

S → A | B
A → a A b | c
B → b B d | e

S

Parse tree for a a c b b

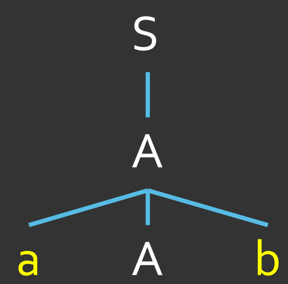
Example Parse tree

S → A | B
A → a A b | c
B → b B d | e

Parse tree for a a c b b

Example Parse tree

S → A | B
A → a A b | c
B → b B d | e

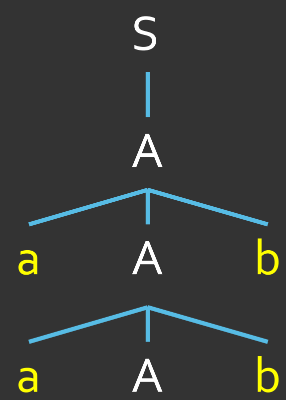


Parse tree for a a c b b

Example Parse tree

S → A | B
A → a A b | c
B → b B d | e

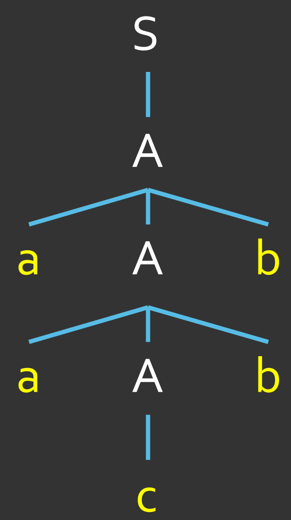
Parse tree for a a c b b



Example Parse tree

S → A | B
A → a A b | c
B → b B d | e

Parse tree for a a c b b

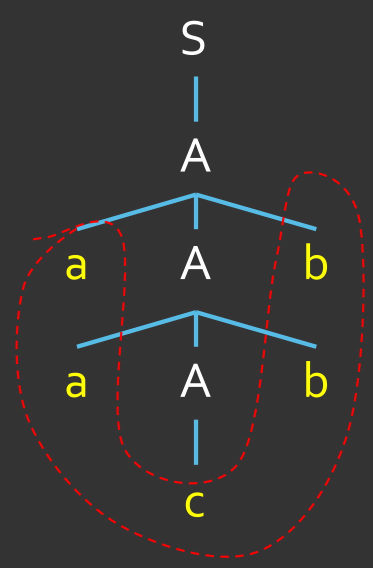


Example Parse tree

S → A | B
A → a A b | c
B → b B d | e

Parse tree for a a c b b

The leaves from left to right match the input

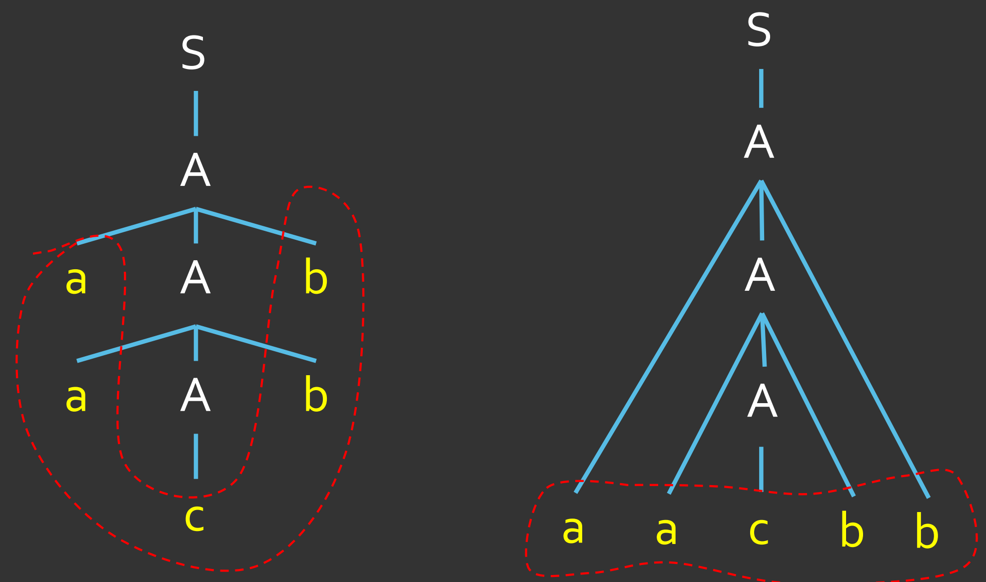


Example Parse tree

S → A | B
A → a A b | c
B → b B d | e

Parse tree for a a c b b

The leaves from left to right match the input



Parse Trees

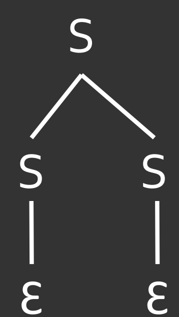
$$S \rightarrow (S) \mid S\,S \mid \varepsilon$$

$$\begin{array}{c} S \\ | \\ \varepsilon \end{array}$$

Parse Trees

$S \rightarrow (S) \mid SS \mid \varepsilon$

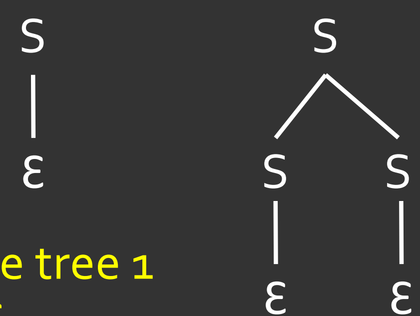
parse tree 1
for ε



parse tree 2
for ε

Parse Trees

$S \rightarrow (S) \mid SS \mid \varepsilon$



parse tree 1 for ε

parse tree 2 for ε

the grammar is ambiguous because there are two parse trees for ε

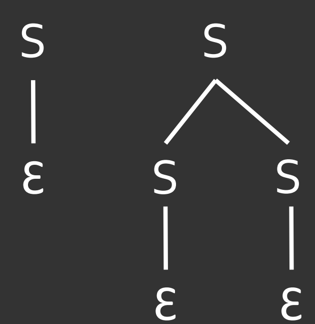
Ambiguous Grammars

- Definition. A grammar is ambiguous if and only if some string (sequence of tokens) has two different parse tree

Example

$S \rightarrow (S) \mid SS \mid \varepsilon$

the grammar in this example is ambiguous because there are two parse trees for ε

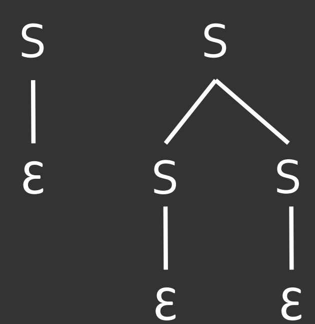


Ambiguous Grammars

- Definition. A grammar is ambiguous if and only if some string (sequence of tokens) has two different parse tree

Example

$$S \rightarrow (S) \mid SS \mid \varepsilon$$



the grammar in this example is ambiguous because there are two parse trees for ε

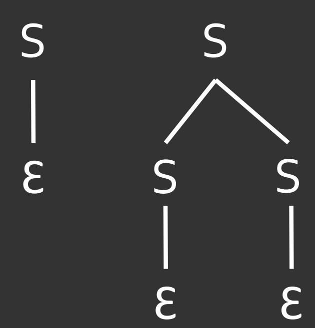
It is the grammar that is ambiguous

Ambiguous Grammars

- Definition. A grammar is ambiguous if and only if some string (sequence of tokens) has two different parse tree

Example

$$S \rightarrow (S) \mid SS \mid \varepsilon$$



the grammar in this example is ambiguous because there are two parse trees for ε

It is the grammar that is ambiguous

- it is not the string that is ambiguous

Ambiguous Grammars

- Definition. A grammar is ambiguous if and only if some string (sequence of tokens) has two different parse tree

Example

$$S \rightarrow \quad (S) \mid SS \mid \varepsilon$$

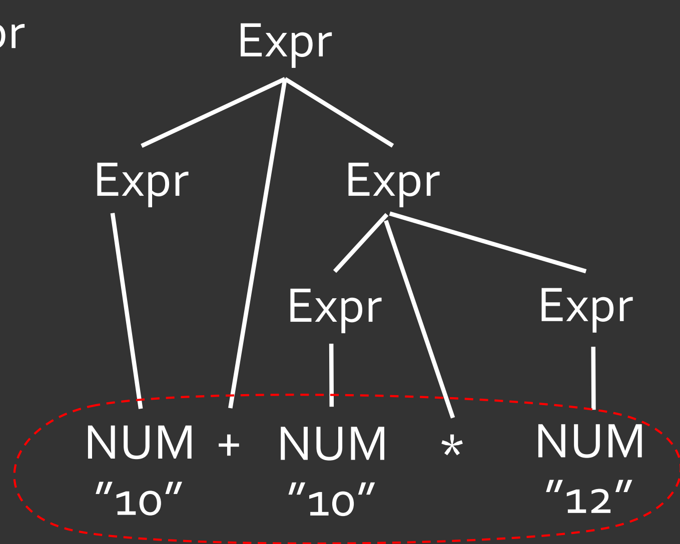
the grammar in this example is ambiguous because there are two parse trees for ε

It is the grammar that is ambiguous

- it is not the string that is ambiguous
- it is not the language of the grammar that is ambiguous

Ambiguous Grammars: another example

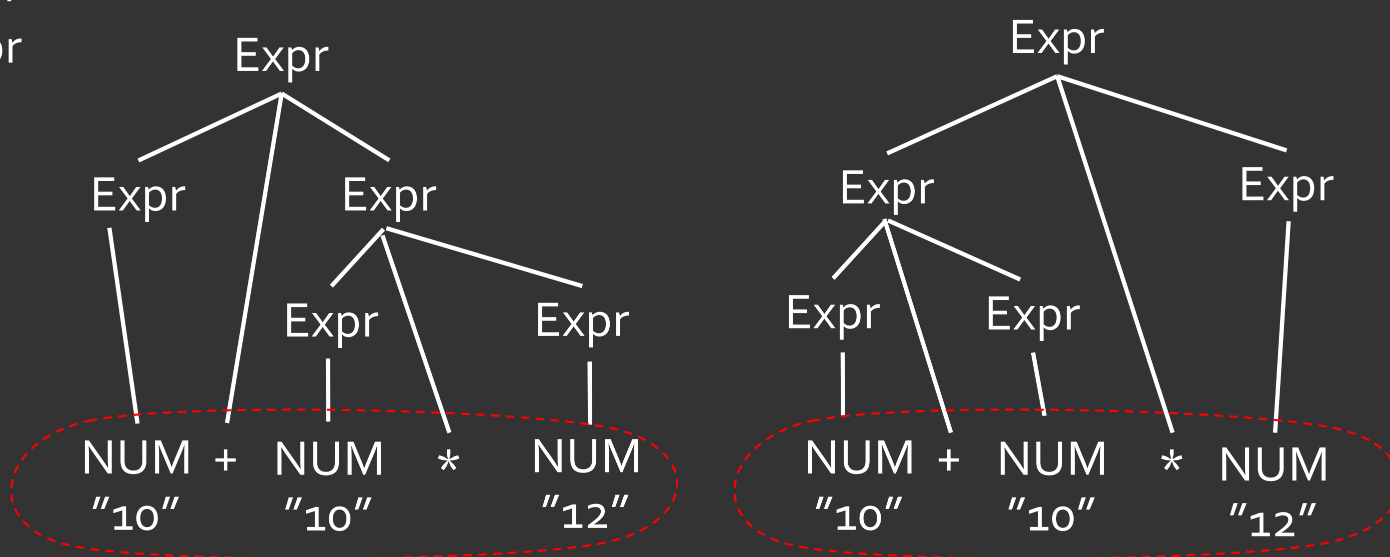
Expr → Expr + Expr
Expr → Expr * Expr
Expr → NUM



`10 + 10 * 12` : NUM + NUM * NUM

Ambiguous Grammars: another example

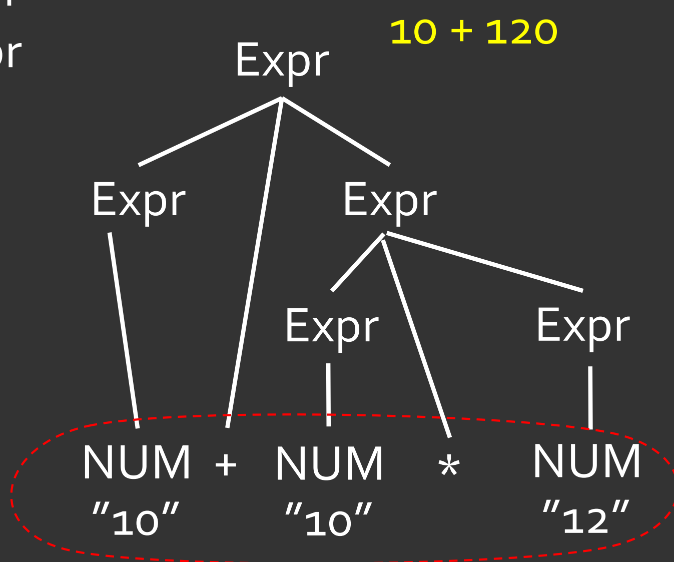
Expr → Expr + Expr
Expr → Expr * Expr
Expr → NUM

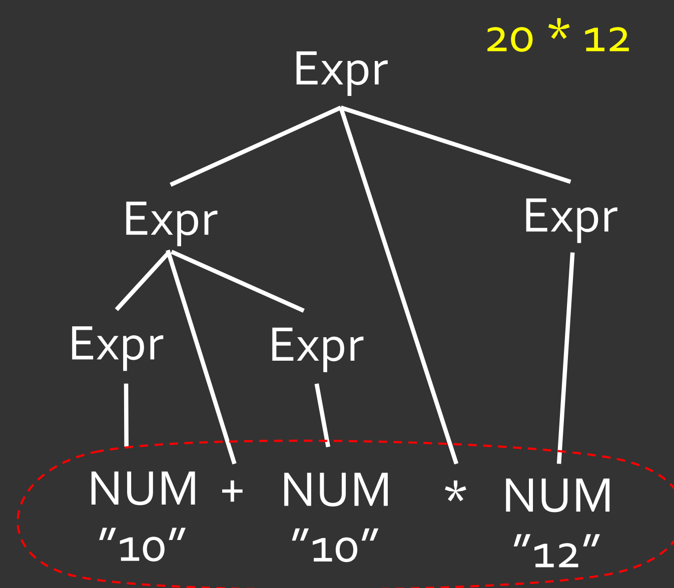


`10 + 10 * 12` : NUM + NUM * NUM

Ambiguous Grammars: another example

Expr → Expr + Expr
Expr → Expr * Expr
Expr → NUM





This is a problem because syntax gives meaning:

10 + 120 vs.
20 * 12

10 + 10 * 12 : NUM + NUM * NUM

Dealing with ambiguous grammars

Dealing with ambiguous grammars

- As we saw in the previous example, the parse tree captures some information about the meaning of the input

Dealing with ambiguous grammars

- As we saw in the previous example, the parse tree captures some information about the meaning of the input
- In the case of an expression, we want only the parse tree that captures the correct operator precedence

Dealing with ambiguous grammars

- As we saw in the previous example, the parse tree captures some information about the meaning of the input
- In the case of an expression, we want only the parse tree that captures the correct operator precedence
- There are two ways to deal with ambiguity

Dealing with ambiguous grammars

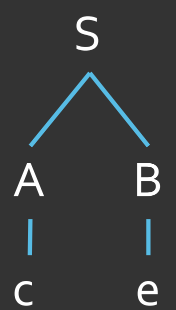
- As we saw in the previous example, the parse tree captures some information about the meaning of the input
- In the case of an expression, we want only the parse tree that captures the correct operator precedence
- There are two ways to deal with ambiguity
  1. modify the grammar to obtain another unambiguous grammar for the same language. We have seen that for the expression grammar when we studied the expression grammar with expr, tern, and factor

Dealing with ambiguous grammars

- As we saw in the previous example, the parse tree captures some information about the meaning of the input
- In the case of an expression, we want only the parse tree that captures the correct operator precedence
- There are two ways to deal with ambiguity
  1. modify the grammar to obtain another unambiguous grammar for the same language. We have seen that for the expression grammar when we studied the expression grammar with expr, tern, and factor
  2. keep the grammar as is and add extra semantic disambiguation rules that specify the preference that the parser should make when there are more than one parse tree (operator precedence for example)

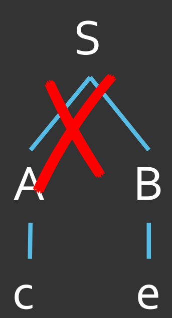
Non Parse trees!

S → A | B
A → a A b | c
B → b B d | e



Non Parse trees!

S → A | B
A → a A b | c
B → b B d | e



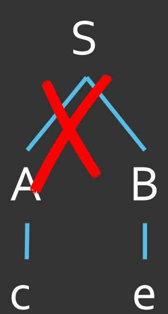
S → A B is not
a grammar rule

Non Parse trees!

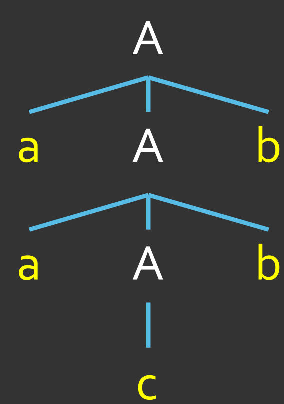
$S \rightarrow A \mid B$

$A \rightarrow a\,A\,b \mid c$

$B \rightarrow b\,B\,d \mid e$

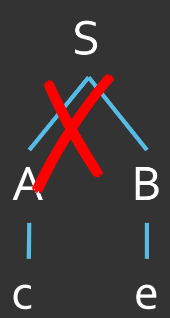


S → A B is not a grammar rule

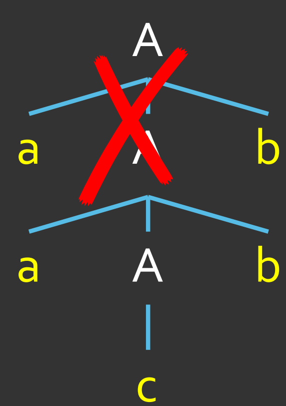


Non Parse trees!

S → A | B
A → a A b | c
B → b B d | e



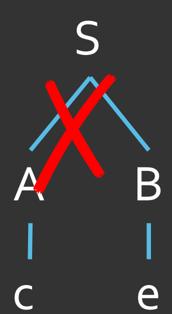
S → A B is not a grammar rule



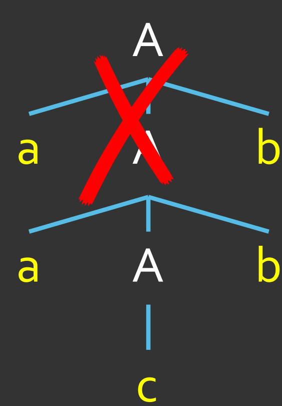
A is not the start symbol

Non Parse trees!

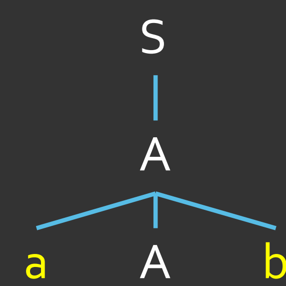
S → A | B
A → a A b | c
B → b B d | e



S → A B is not a grammar rule

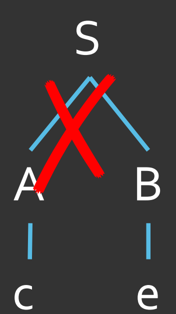


A is not the start symbol

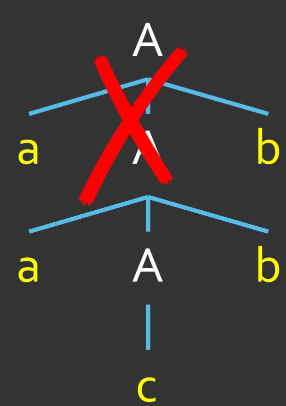


Non Parse trees!

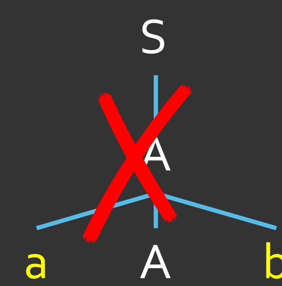
S → A | B
A → a A b | c
B → b B d | e



S → A B is not a grammar rule



A is not the start symbol



A is a leaf and not a terminal or epsilon

What is ε ?

What is ε ?

- Going back to the definition of a grammar rule, we notice that the righthand side of a rule is an element of $(T \cup NT)^*$

What is ε ?

- Going back to the definition of a grammar rule, we notice that the righthand side of a rule is an element of $(T \cup NT)^*$
- This means that the righthand side of a rule is a sequence of zero or more terminals and non-terminals

What is ε ?

- Going back to the definition of a grammar rule, we notice that the righthand side of a rule is an element of $(T \cup NT)^*$
- This means that the righthand side of a rule is a sequence of zero or more terminals and non-terminals
- When this sequence is empty, the rule has the form

$$A \rightarrow \varepsilon$$

What is ε ?

- Going back to the definition of a grammar rule, we notice that the righthand side of a rule is an element of (T ∪ NT)*
- This means that the righthand side of a rule is a sequence of zero or more terminals and non-terminals
- When this sequence is empty, the rule has the form

$$A \rightarrow \varepsilon$$

It has been my experience that ε might be confusing to someone studying the material for the first time. In the next couple of slides, I will try to clarify the concept

What is ε ?

ε represents an empty sequence of tokens

ε itself is not a token

```
What is ε ?
```

ε represents an empty sequence of tokens

ε **is not** a token

What is ε ?

ε represents an empty sequence of tokens

ε is **NOT** a token

EVER

What is ε ?

ε represents an empty sequence of tokens

ε itself is not a token

So, what does it mean when we have a rule of the form A → ε ?

To answer, the question, I will consider the following grammar

program	→	decl_section stmt_list	
decl_section	→	decl decl_section	| ε
stmt_list	→	stmt stmt_list	| stmt
stmt	→	ID EQUAL ID	
decl	→	type_name ID SEMICOLON	
type_name	→	ID	

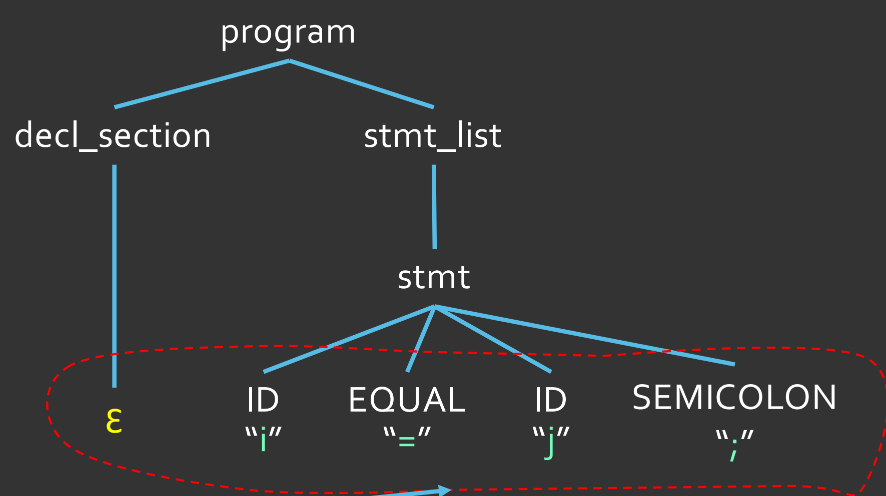
What is ε ?

program	→	decl_section stmt_list	
decl_section	→	decl decl_section	| ε
stmt_list	→	stmt stmt_list	| stmt
stmt	→	ID EQUAL ID SEMICOLON	
decl	→	type_name ID SEMICOLON	
type_name	→	ID	

Let us consider the following input

i = j;

the parse tree is given on the right



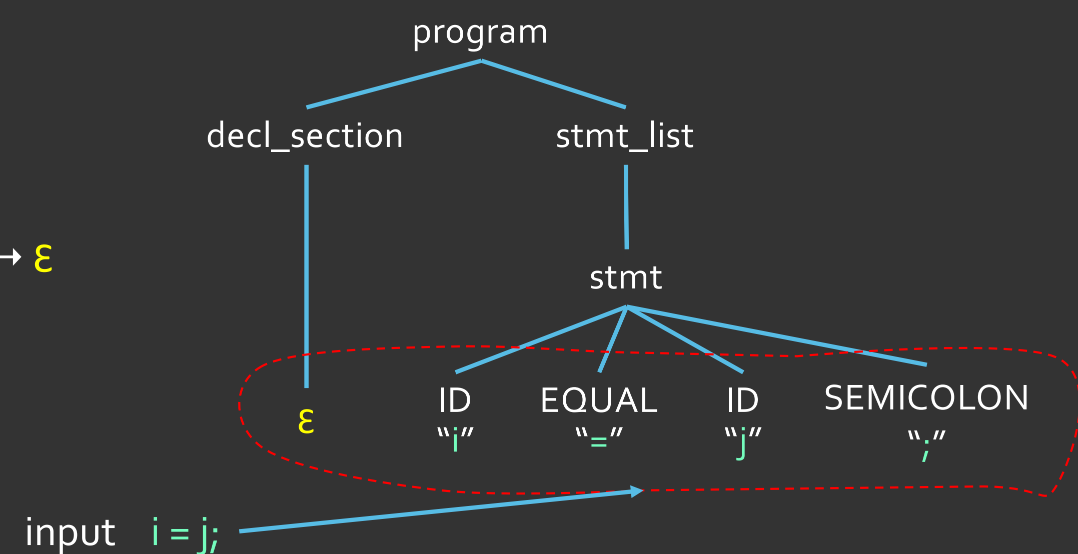
input i = j;

What is ε ?

program	→	decl_section stmt_list	
decl_section	→	decl decl_section	| ε
stmt_list	→	stmt stmt_list	| stmt
stmt	→	ID EQUAL ID SEMICOLON	
decl	→	type_name ID SEMICOLON	
type_name	→	ID	

In the example, decl_section matches an empty sequence of tokens.

The parse tree is valid because decl_section → ε is a grammar rule

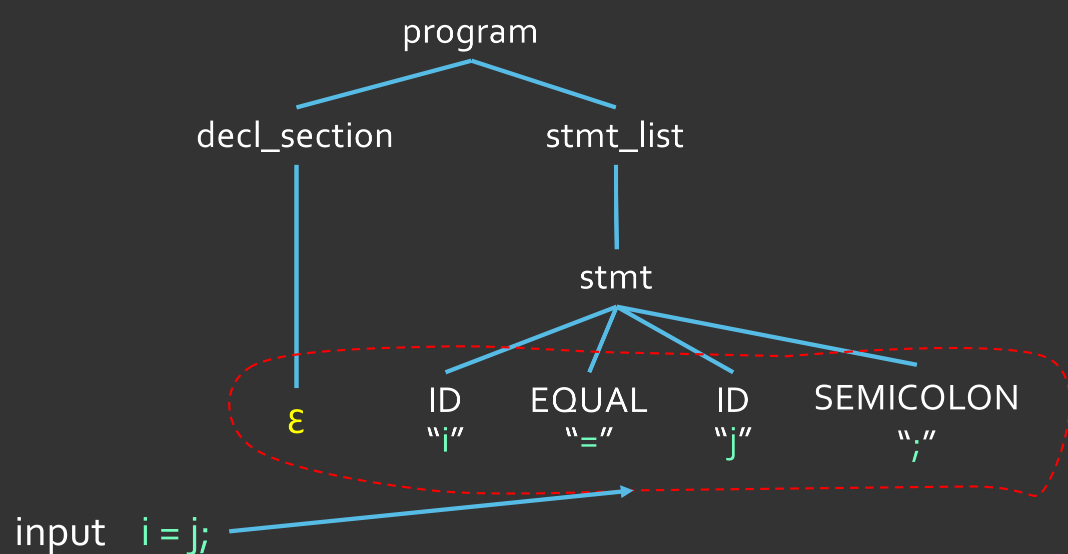


What is ε ?

program	→	decl_section stmt_list	
decl_section	→	decl decl_section	| ε
stmt_list	→	stmt stmt_list	| stmt
stmt	→	ID EQUAL ID SEMICOLON	
decl	→	type_name ID SEMICOLON	
type_name	→	ID	

in a sense, the rule decl_section → ε
is kind of making decl_section optional

But we have to be careful! We cannot have a
parse tree without a decl_section



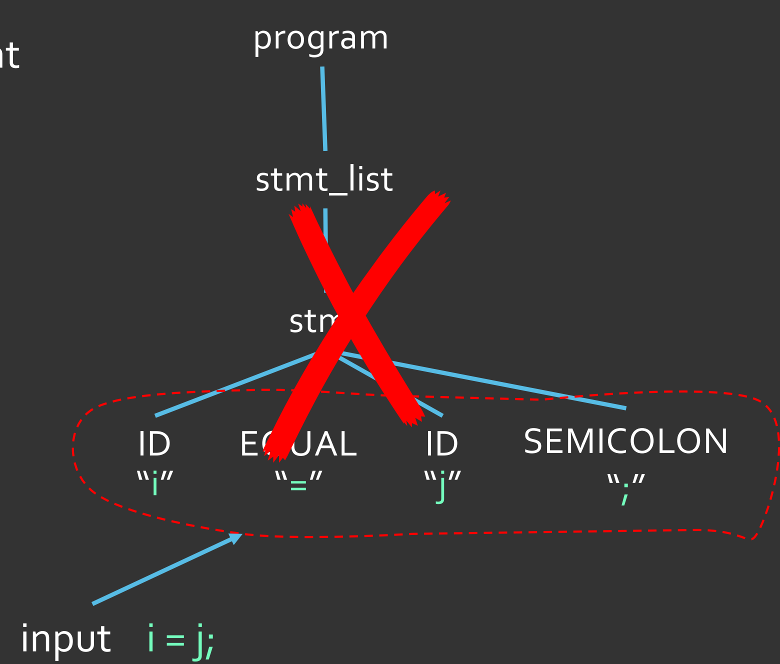
What is ε ?

program	→	decl_section stmt_list	
decl_section	→	decl decl_section	| ε
stmt_list	→	stmt stmt_list	| stmt
stmt	→	ID EQUAL ID SEMICOLON	
decl	→	type_name ID SEMICOLON	
type_name	→	ID	

in a sense, sense the rule decl_section → ε
is kind of making decl_section optional

But we have to be careful! We cannot have a
parse tree without a decl_section

The modified tree on the right is not valid because
program → stmt_list is not a grammar rule



Why have epsilon?

Epsilon allows us to write more compact grammars. I give an example grammar with 9 rules

A → B C D E

B → b | Ɛ

C → c | Ɛ

D → d | Ɛ

E → e | Ɛ

If we want to write an equivalent grammar without epsilon, we should consider all the possibilities in which B, C, D and E can be epsilon. We obtain the following grammar with 16 rules

A → A B C D	// B, C, D and E not epsilon
A → B C D | B C E | B E D | C D E	// exactly one is epsilon
A → B C | B D | B E | C D | C E | E D	// exactly two are epsilon
A → B | C | D | E	// exactly three are epsilon
A → Ɛ	// all of them are epsilon

Derivations

A derivation is another way to represent how a sequence of tokens can be parsed according to a given grammar

Derivations

A derivation is another way to represent how a sequence of tokens can be parsed according to a given grammar

We first define derives in one step:

x A y derives x β y in one step, written $x\ A\ y \Rightarrow x\ \boldsymbol{\beta}\ y$ if and only if

> **x** and **y** are strings of terminals and non-terminals
> **A → β** is a grammar rule

Derivations

A derivation is another way to represent how a sequence of tokens can be parsed according to a given grammar

We first define derives in one step:

$x\,A\,y$ derives $x\,\beta\,y$ in one step, written $x\,A\,y \Rightarrow x\,\beta\,y$ if and only if

x and y are strings of terminals and non-terminals
$A \rightarrow \beta$ is a grammar rule

In other words, given a string of terminals and non terminals $x\,A\,y$, we can replace the non-terminal A with the right hand side β of one of its rules and obtain a new string $x\,\beta\,y$

Example derivation in one step

Expression Grammar

$$\begin{aligned} \text{Expr} &\rightarrow \text{Expr} + \text{Expr} \\ \text{Expr} &\rightarrow \text{Expr} * \text{Expr} \\ \text{Expr} &\rightarrow \text{NUM} \end{aligned}$$

$$\text{Expr} + \text{Expr} + \text{Expr} \Rightarrow \text{Expr} + \text{Expr} * \text{Expr} + \text{Expr}$$

Example derivation in one step

Expression Grammar

$$\begin{aligned} Expr &\rightarrow Expr + Expr \\ Expr &\rightarrow Expr * Expr \\ Expr &\rightarrow NUM \end{aligned}$$

$$\underbrace{Expr\ +}_{x}\ \underbrace{Expr}_{A}\ \underbrace{+\ Expr}_{y} \Rightarrow \underbrace{Expr\ +}_{x}\ \underbrace{Expr * Expr}_{\beta}\ \underbrace{+\ Expr}_{y}$$

Example derivation in one step

Expression Grammar

$$
\begin{aligned}
\text{Expr} &\rightarrow \text{Expr} + \text{Expr} \\
\text{Expr} &\rightarrow \text{Expr} * \text{Expr} \\
\text{Expr} &\rightarrow \text{NUM}
\end{aligned}
$$

$$\underbrace{\text{Expr} +}_{x}\ \underbrace{\text{Expr}}_{A}\ \underbrace{+ \text{Expr}}_{y} \Rightarrow \underbrace{\text{Expr} +}_{x}\ \underbrace{\text{Expr} * \text{Expr}}_{\beta}\ \underbrace{+ \text{Expr}}_{y}$$

$$\text{Expr} \rightarrow \text{Expr} * \text{Expr}$$

$$A \rightarrow \beta$$

Example derivation in one step

Expression Grammar

$$\begin{aligned} Expr &\rightarrow Expr + Expr \\ Expr &\rightarrow Expr * Expr \\ Expr &\rightarrow NUM \end{aligned}$$

$$Expr + Expr \Rightarrow NUM + Expr$$

Example derivation in one step

Expression Grammar

$$\begin{aligned} Expr &\rightarrow Expr + Expr \\ Expr &\rightarrow Expr * Expr \\ Expr &\rightarrow NUM \end{aligned}$$

$$\underbrace{\varepsilon}_{x} \ \underbrace{Expr}_{A} \ \underbrace{+ \ Expr}_{y} \Rightarrow \underbrace{\varepsilon}_{x} \ \underbrace{NUM}_{\beta} \ \underbrace{+ \ Expr}_{y}$$

Example derivation in one step

Expression Grammar

$$\begin{aligned} \text{Expr} &\rightarrow \text{Expr} + \text{Expr} \\ \text{Expr} &\rightarrow \text{Expr} * \text{Expr} \\ \text{Expr} &\rightarrow \text{NUM} \end{aligned}$$

$$\underbrace{\varepsilon}_{x}\ \underbrace{\text{Expr}}_{A}\ \underbrace{+\ \text{Expr}}_{y} \Rightarrow \underbrace{\varepsilon}_{x}\ \underbrace{\text{NUM}}_{\beta}\ \underbrace{+\ \text{Expr}}_{y}$$

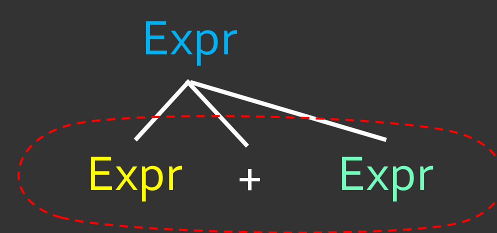
$\text{Expr} \rightarrow \text{NUM}$

$A \rightarrow \beta$

Example derivation for 10+11*12

Expr ⇒ Expr + Expr

Expr → Expr + Expr

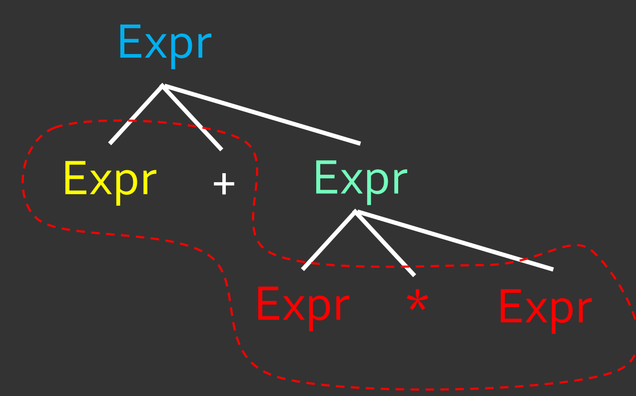


Example derivation for 10+11*12

Expr → Expr * Expr

$$\text{Expr} \Rightarrow \text{Expr} + \text{Expr} \Rightarrow \text{Expr} + \text{Expr} * \text{Expr}$$

Expr → Expr + Expr



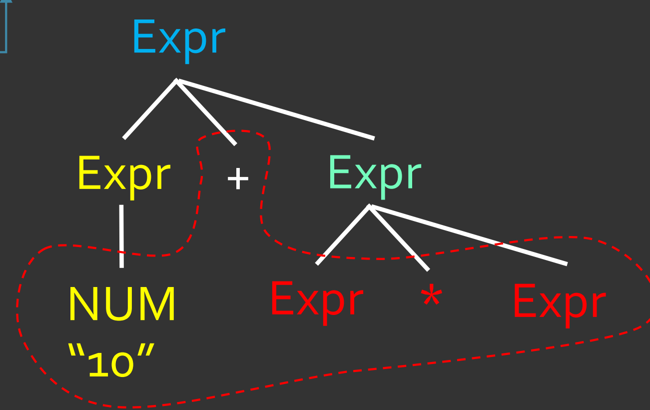
Example derivation for 10+11*12

Expr → Expr * Expr

Expr ⇒ Expr + Expr ⇒ Expr + Expr * Expr ⇒ NUM + Expr * Expr

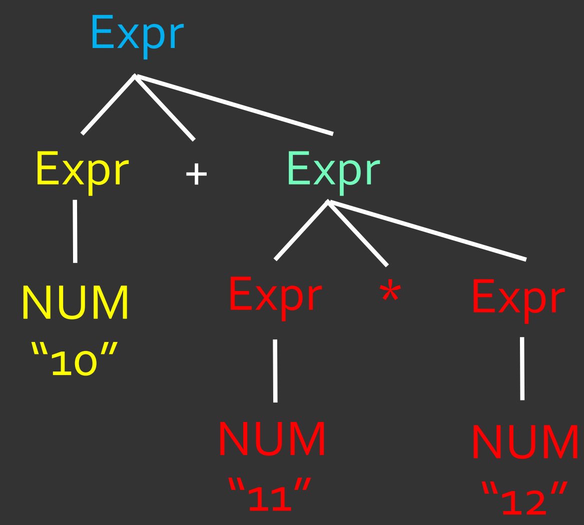
Expr → Expr + Expr

Expr → NUM



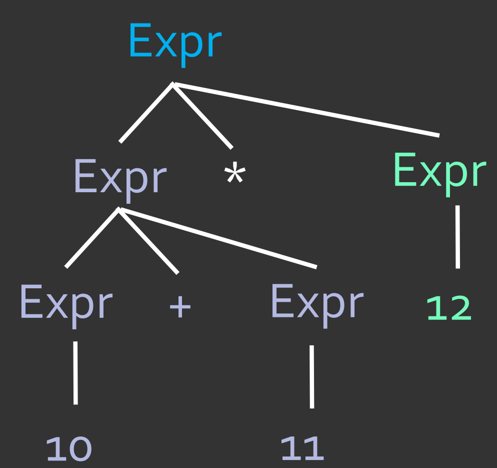
Example derivation for 10+11*12

Expr ⇒ Expr + Expr
⇒ Expr + Expr * Expr
⇒ NUM + Expr * Expr
⇒ NUM + NUM * Expr
⇒ NUM + NUM * NUM



Example derivation for 10+11*12

$$
\begin{aligned}
Expr &\Rightarrow Expr * Expr \\
&\Rightarrow Expr + Expr * Expr \\
&\Rightarrow NUM + Expr * Expr \\
&\Rightarrow NUM + NUM * Expr \\
&\Rightarrow NUM + NUM * NUM
\end{aligned}
$$



Derivations

Let *w* and *w′* be sequences of terminals and non-terminals

We say that ***w*** derives ***w′*** in zero or more steps and we write $w \overset{*}{\Rightarrow} w'$ if

$w = w'$ (derivation in 0 step)

Derivations

Let *w* and *w'* be sequences of terminals and non-terminals

We say that ***w*** derives ***w'*** in zero or more steps and we write $w \overset{*}{\Rightarrow} w'$ if

$w = w'$ (derivation in 0 step)

there exists z such that $w \Rightarrow z$ and $z \overset{*}{\Rightarrow} w'$

Derivations

Let w and w' be sequences of terminals and non-terminals

We say that ***w*** derives ***w'*** in zero or more steps and we write $w \overset{*}{\Rightarrow} w'$ if

$w = w'$ (derivation in 0 step)

there exists $w_1, w_2, \ldots, w_k$ ($k \geq 2$) such that

$w = w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \ldots \Rightarrow w_k = w'$ (derivation in k-1 steps)

Language of a Grammar

- The language of a grammar G with start symbol S is the set of strings that can be derived from S

$$L(G) = \{ w \in T^* : S \overset{*}{\Rightarrow} w \}$$

Language of a Grammar

- The language of a grammar G with start symbol S is the set of strings that can be derived from S

$$L(G) = \{ w \in T^* : S \overset{*}{\Rightarrow} w \}$$

How do we read this?

Language of a Grammar

- The language of a grammar G with start symbol S is the set of strings that can be derived from S

$$L(G) = \{\ w \in T^* \ :\ S \overset{*}{\Rightarrow} w\ \}$$

language of G = set of strings of terminals that can be derived from S

If T is a set of symbols, T* is the set of sequences of symbols from T, including the empty sequence

Leftmost and Rightmost derivations

- A derivation is a leftmost derivation if at every step the **leftmost non-terminal** is replaced with a right hand side of one of its rules

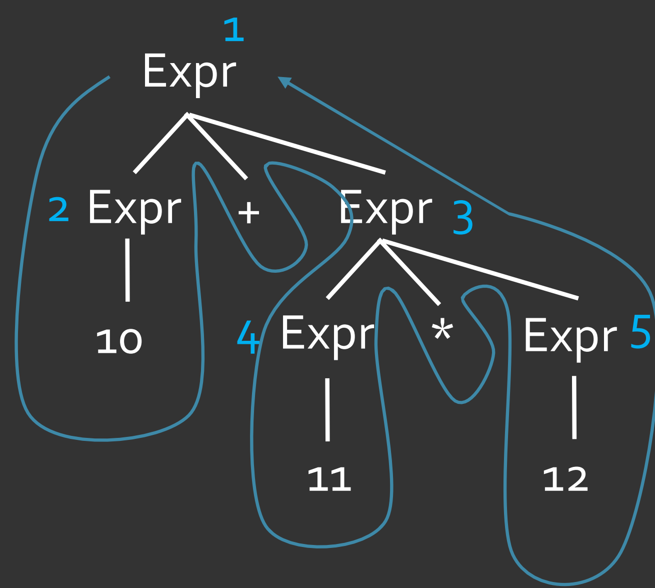
Leftmost and Rightmost derivations

- A derivation is a leftmost derivation if at every step the **leftmost non-terminal** is replaced with a right hand side of one of its rules

- A derivation is a rightmost derivation if at every step the **rightmost non-terminal** is replaced with a right hand side of one of its rules

Example leftmost derivation for 10+11*12

$$
\begin{aligned}
\text{Expr}^{1} &\Rightarrow \text{Expr}^{2} + \text{Expr} \\
&\Rightarrow \text{NUM} + \textbf{Expr}^{3} \\
&\Rightarrow \text{NUM} + \textbf{Expr}^{4} * \text{Expr} \\
&\Rightarrow \text{NUM} + \text{NUM} * \textbf{Expr}^{5} \\
&\Rightarrow \text{NUM} + \text{NUM} * \text{NUM}
\end{aligned}
$$

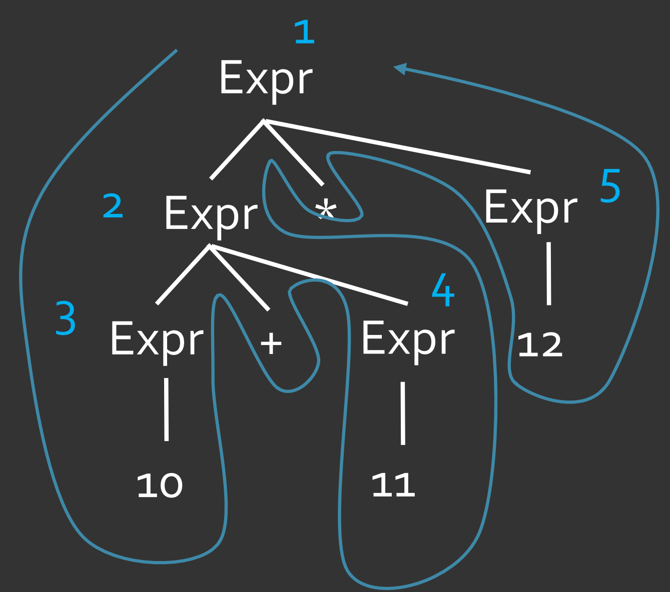


This derivation is not a rightmost derivation because

Expr + Expr ⇒ NUM + **Expr** is not part of a rightmost derivation

Another leftmost derivation for 10+11*12

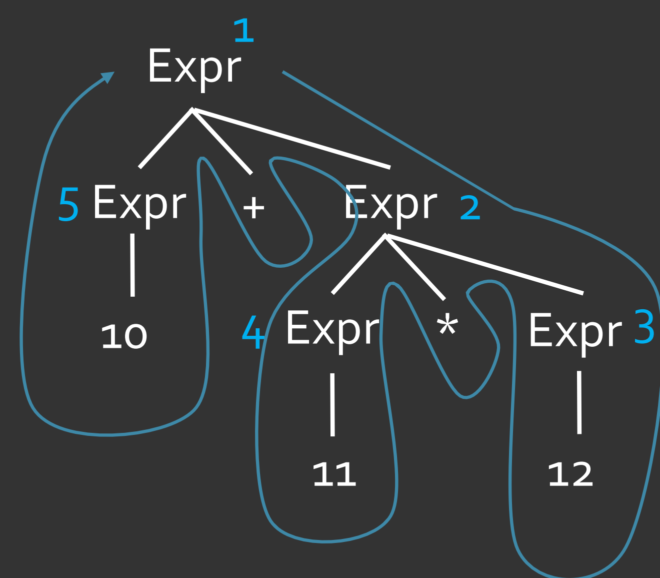
Expr ⇒ Expr * Expr
⇒ Expr + Expr * Expr
⇒ NUM + **Expr** * Expr
⇒ NUM + NUM * **Expr**
⇒ NUM + NUM * NUM



Example rightmost derivation for 10+11*12

$$
\begin{aligned}
\text{Expr}^{1} &\Rightarrow \text{Expr} + \textbf{Expr}^{2} \\
&\Rightarrow \text{Expr} + \text{Expr} * \textbf{Expr}^{3} \\
&\Rightarrow \text{Expr} + \textbf{Expr}^{4} * \text{NUM} \\
&\Rightarrow \textbf{Expr}^{5} + \text{NUM} * \text{NUM} \\
&\Rightarrow \text{NUM} + \text{NUM} * \text{NUM}
\end{aligned}
$$

The rightmost non-terminal is replaced in each step



Parse trees and leftmost and rightmost derivations

- For every parse tree, there exists one unique leftmost derivation which corresponds to a depth-first traversal from left to right

Parse trees and leftmost and rightmost derivations

- For every parse tree, there exists one unique leftmost derivation which corresponds to a depth-first traversal from left to right
- For every parse tree, there exists a unique rightmost derivation that corresponds to a depth-first traversal from right to left

Parse trees and leftmost and rightmost derivations

- For every parse tree, there exists one unique leftmost derivation which corresponds to a depth-first traversal from left to right
- For every parse tree, there exists a unique rightmost derivation that corresponds to a depth-first traversal from right to left

It follows that

- A grammar is ambiguous if and only if some string has two different leftmost derivations

Parse trees and leftmost and rightmost derivations

- For every parse tree, there exists one unique leftmost derivation which corresponds to a depth-first traversal from left to right
- For every parse tree, there exists a unique rightmost derivation that corresponds to a depth-first traversal from right to left

It follows that

- A grammar is ambiguous if and only if some string has two different leftmost derivations
- A grammar is ambiguous if and only if some string has two different rightmost derivations

Ambiguous Grammar (again)

- We can show that a grammar is ambiguous by showing that some string has two different parse trees

Ambiguous Grammar (again)

- We can show that a grammar is ambiguous by showing that some string has two different parse trees
- We can show that a grammar is ambiguous by showing that some string has two derivation leftmost derivations

Ambiguous Grammar (again)

- We can show that a grammar is ambiguous by showing that some string has two different parse trees
- We can show that a grammar is ambiguous by showing that some string has two derivation leftmost derivations
- We can show that a grammar is ambiguous by showing that some string has two different rightmost derivations

All three conditions are EQUIVALENT

Ambiguous Grammar (again)

- We **cannot show** that a grammar is ambiguous by showing that some string has a leftmost and a rightmost derivations that are different!

Ambiguous Grammar (again)

- We **cannot show** that a grammar is ambiguous by showing that some string has a leftmost and a rightmost derivations that are different!

Example

$S \rightarrow A\ B$
$A \rightarrow a$
$B \rightarrow b$

clearly not ambiguous

Ambiguous Grammar (again)

- We **cannot show** that a grammar is ambiguous by showing that some string has a leftmost and a rightmost derivations that are different!

Example

$S \rightarrow A\ B$
$A \rightarrow a$
$B \rightarrow b$ clearly not ambiguous

Leftmost derivation of ab: $S \Rightarrow A\ B \Rightarrow a\ B \Rightarrow a\ b$

Rightmost derivation of ab: $S \Rightarrow A\ B \Rightarrow A\ b \Rightarrow a\ b$

They are different!