San José State University
Department of Computer Science

Ahmad Yazdankhah
ahmad.yazdankhah@sjsu.edu
www.cs.sjsu.edu/~yazdankhah

Grammars

(Part 4)

Lecture 23

Day 27/31

CS 154

Formal Languages and Computability

Spring 2018

Agenda of Day 27

- Summary of Lecture 22
- Quiz 9
- Lecture 23: Teaching ...
  - Grammars (Part 4)

Summary of Lecture 22: We learned ...

Grammars

- A context-free language (CFL) is ...
  - ... a language produced by a CFG.

S-Grammar

- A simple grammar is ...
  - ... a cfg with two restrictions:

1. All production rules are of the form
   $A \to av$
   where $A \in V$, $a \in T$, $v \in V^*$

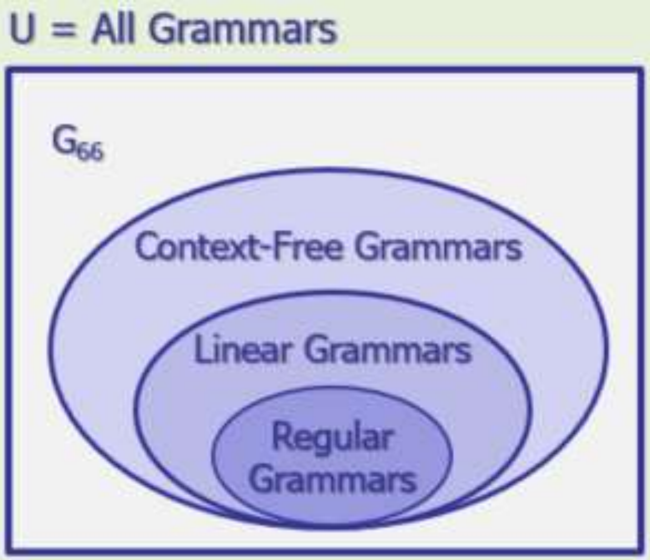
   One terminal as prefix and any number of variables as suffix.

2. Any pair (A, x) occurs only once in all production rules.

Derivation Techniques

- There are two derivation techniques:
  - Leftmost and rightmost derivation.
  - Leftmost is the default method.

Grammars hierarchy

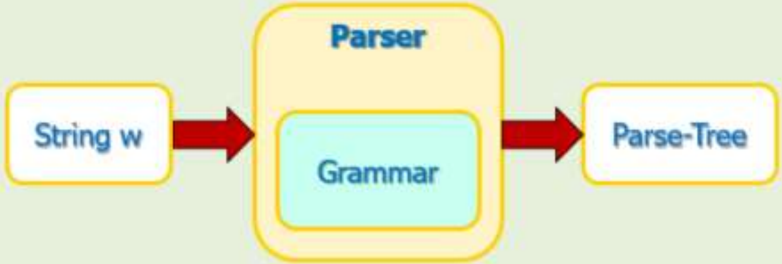


Any Question

Summary of Lecture 22: We learned ...

Parser

- Parser is ...
  - ... a program that gets a string as input and gives the sequence of derivation as the output.



- Every compiler has its own grammar and parser.

Parse-Trees

- Parse-tree is ...
  - ... an ordered-tree that can be constructed for every string by using the grammar.

Any Question

NAME	Alan M. Turing		
SUBJECT	CS 154	TEST NO.	9
DATE	04/26/2018	PERIOD	1 / 2 / 3

TEST RECORD	
PART 1	123
PART 2	
TOTAL	

Your **list #** goes here!

Quiz 9

Use Scantron

Parsing Algorithms

Parsing Algorithms

- There are two main types of algorithms for parsers:
  1. Top-down
  2. Bottom-up

- To see the idea, we'll examine a top-down algorithm called "exhaustive search parsing" (aka "brute force parsing").
  - This algorithm checks all possibilities.

- We'll explain it through an example.

- For more information about other algorithms, you need to take Compiler Course!

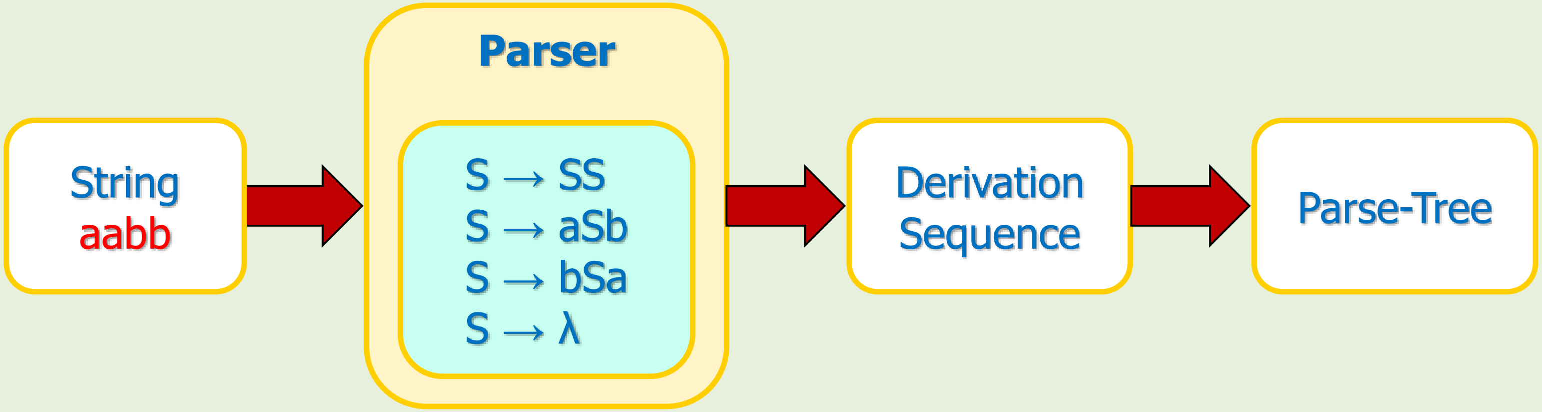
Exhaustive Search Parsing Algorithm: Example

Example 28

- Given the following grammar:

  $S \rightarrow SS \mid a\,S\,b \mid b\,S\,a \mid \lambda$

- Find a derivation sequence for w = aabb.
- Note that if we get the derivation sequence, then drawing the parse-tree would be simple.



Exhaustive Search Parsing Algorithm: Example

Example 28 (cont'd)

$S \rightarrow SS \mid aSb \mid bSa \mid \lambda$
$w = aabb$

- **Round One**
  1. $S \Rightarrow SS$
  2. $S \Rightarrow aSb$
  3. $S \Rightarrow bSa$
  4. $S \Rightarrow \lambda$
- Which production rules can be pruned?
- Number 3 and 4 can be pruned because they will never yield to w.

- **Conclusion of Round One**
  1. $S \Rightarrow SS$
  2. $S \Rightarrow aSb$
  3. ~~$S \Rightarrow bSa$~~
  4. ~~$S \Rightarrow \lambda$~~
- Therefore, 1 and 2 are our starters after the first round.

Exhaustive Search Parsing Algorithm: Example

Example 28 (cont'd)

$S \to SS \mid aSb \mid bSa \mid \lambda$
$w = aabb$

- **Conclusion of Round One**

1. $S \Rightarrow SS$
2. $S \Rightarrow aSb$
3. ~~$S \Rightarrow bSa$~~
4. ~~$S \Rightarrow \lambda$~~

Repeated

- In round 2, we substitute all possibilities for leftmost S in #1 and #2.

- **Round Two**

- Substitute leftmost S of #1 with all possible options:

1.1. $S \Rightarrow SS \Rightarrow SS\ S$

1.2. $S \Rightarrow SS \Rightarrow aSb\ S$

1.3. $S \Rightarrow SS \Rightarrow bSa\ S$

1.4. $S \Rightarrow SS \Rightarrow \lambda\ S$

- Substitute leftmost S of #2 with all possible options:

2.1. $S \Rightarrow a\ S\ b \Rightarrow a\ SS\ b$

2.2. $S \Rightarrow a\ S\ b \Rightarrow a\ aSb\ b$

2.3. $S \Rightarrow a\ S\ b \Rightarrow a\ bSa\ b$

2.4. $S \Rightarrow a\ S\ b \Rightarrow a\ \lambda\ b$

Exhaustive Search Parsing Algorithm: Example

Example 28 (cont'd)

$S \rightarrow SS \mid aSb \mid bSa \mid \lambda$
$w = aabb$

- **Conclusion of Round Two**

1.1. $S \Rightarrow SS \Rightarrow SSS$ **Repeated**

1.2. $S \Rightarrow SS \Rightarrow aSbS$

~~1.3. S ⇒ SS ⇒ bSaS~~

1.4. $S \Rightarrow SS \Rightarrow S$

2.1. $S \Rightarrow aSb \Rightarrow aSSb$

2.2. $S \Rightarrow aSb \Rightarrow aaSbb$

~~2.3. S ⇒ aSb ⇒ abSab~~

~~2.4. S ⇒ aSb ⇒ ab~~

- We continue this process …
- **Round 3**
- … (after a little bit cheating!)
- Substitute leftmost S of #2.2 with all possible options:

2.2.1. $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aa\ SS\ bb$

2.2.2. $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aa\ aSb\ bb$

2.2.3. $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aa\ bSa\ bb$

2.2.4. $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

- So, we got the derivation sequence to derive w = aabb

Exhaustive Search Parsing Algorithm: Complexity

- Exhaustive parsing has two serious problems:
  1. It is extremely inefficient: $O(|P|^{2|w|+1})$
     – Where $|P|$ is the number of production rules, $|w|$ is the size of the string.
  2. It is possible that it never terminates.
     – For example, try to find the derivation sequence for $w = abb$ in the previous example.

- How horrible do you think this efficiency is?

- We'll take a practical examples under the "Complexity" part of this course.

Exhaustive Search Parsing Algorithm: Good News

1. **Theorem**

   For every CFG G, there exists an algorithm that parses any $w \in L(G)$ in $O(|w|^3)$ steps.

2. **Using S-Grammar**

   If the grammar is s-grammar, then the parser would be much, much faster.

   – The efficiency would be: $O(|w|)$

- Let's see this through an example.

Exhaustive Search Parsing Algorithm: S-Grammar

Example 29

- Given the following grammar:
  1. $S \to aS$
  2. $S \to bSS$
  3. $S \to c$
- Is this an s-grammar?
- Derive w = abcc

- Yes, because both conditions of s-grammars are satisfied.
- Now Let's derive abcc:

$$S \overset{1}{\Rightarrow} aS \overset{2}{\Rightarrow} abSS \overset{3}{\Rightarrow} abcS \overset{3}{\Rightarrow} abcc$$

- Note that we are still using "exhaustive search parsing".
- The point is that each string has a unique derivation.
- That's why s-grammar is extensively used in the programming languages.

Exhaustive Search Parsing Algorithm: S-Grammar

Theorem

- If G is an s-grammar, then any string $w \in L(G)$ can be parsed with $O(|w|)$.

Proof

- Let's assume $w = a_1 a_2 \dots a_n$
- There can be at most one rule with S on the left and starting with a_1 on the right: $S \Rightarrow a_1 A_1 A_2 \dots A_m$
- Again, there can be at most one rule with A_1 on the left and starting a_2 on the right: $A_1 \Rightarrow a_2 B_1 B_2 \dots B_k$
- So, $S \Rightarrow a_1 a_2 B_1 B_2 \dots B_k \; A_2 \dots A_m$
- It means that after $|w|$ we can derive w.

Ambiguity in Grammars

Introduction

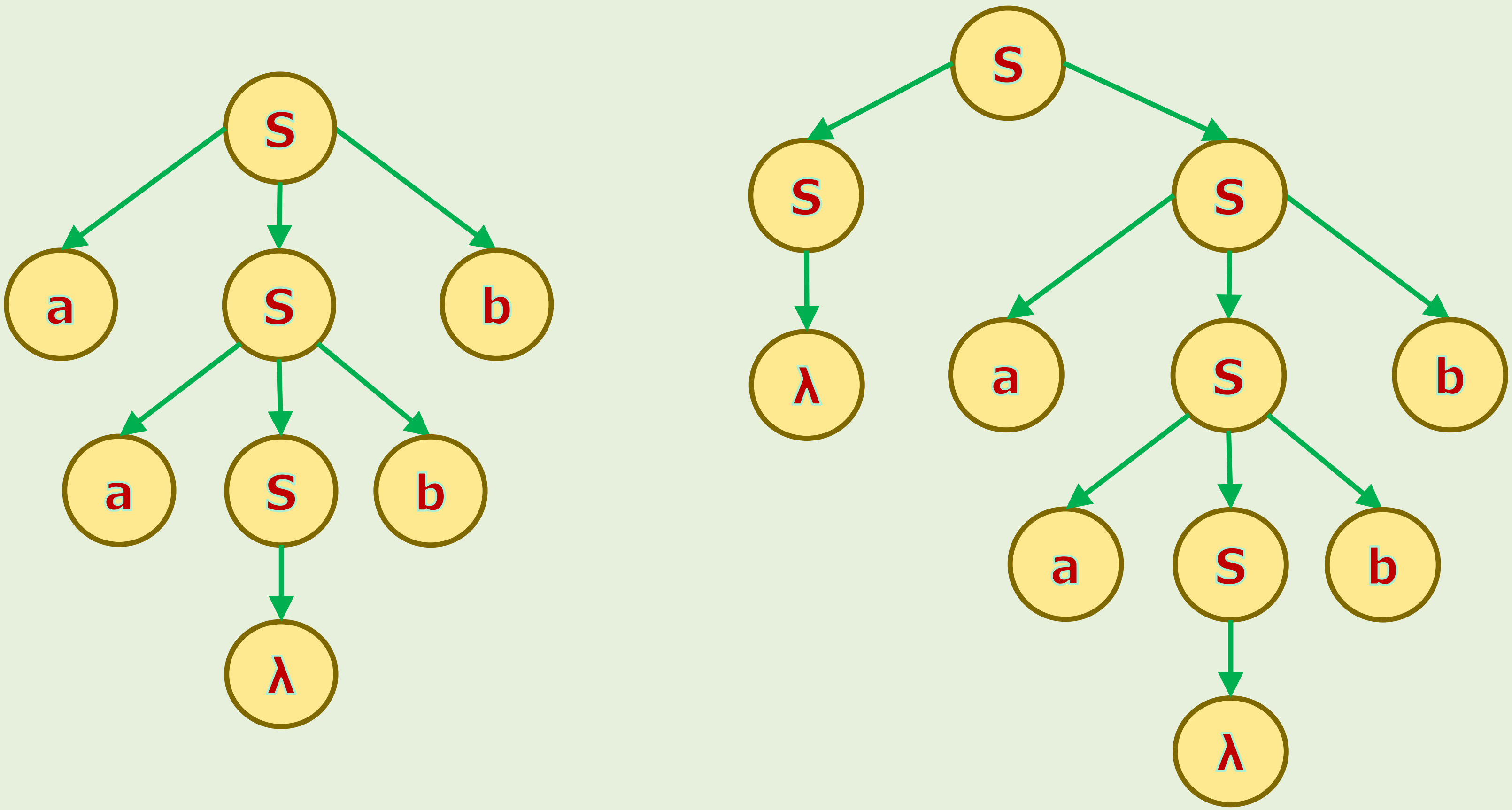
- We learned parsers produce a parse-tree for every $w \in L(G)$.
- But the point is that the parse-tree is NOT always UNIQUE.
  - In other words, in some cases, for a given $w \in L(G)$, there are more than one parse-tree.
- First, let's see this through an example!
- Then, we show what could be the consequence in practice!

When Parse-Tree is NOT Unique

Example 30

Given grammar G as: $S \rightarrow aSb \mid SS \mid \lambda$

Draw possible parse-trees for driving w = aabb.

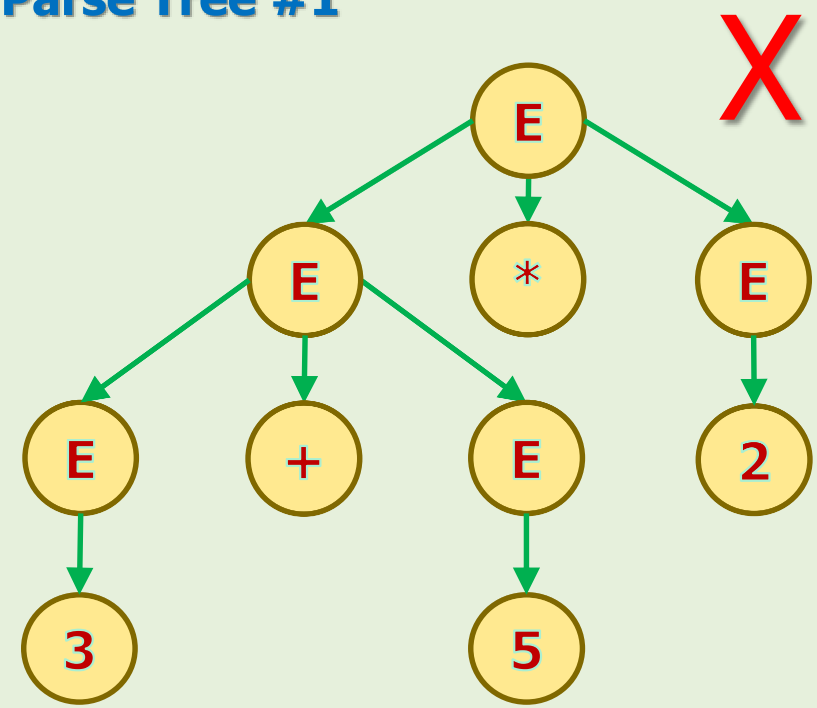


Non-Uniqueness of Parse-Trees in Practice

Example 31

- Given grammar G as:
  1. $E \rightarrow E * E$
  2. $E \rightarrow E + E$
  3. $E \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
- E is starting variable.
- Construct a parse-tree for the mathematical expression: 3 + 5 * 2
- This grammar is a simplified version of arithmetic expressions in the programming languages.

Parse Tree #1



- Is this a good parsing?
- No, because * should have more priority than + but this parse-tree is calculating 3 + 5 first.

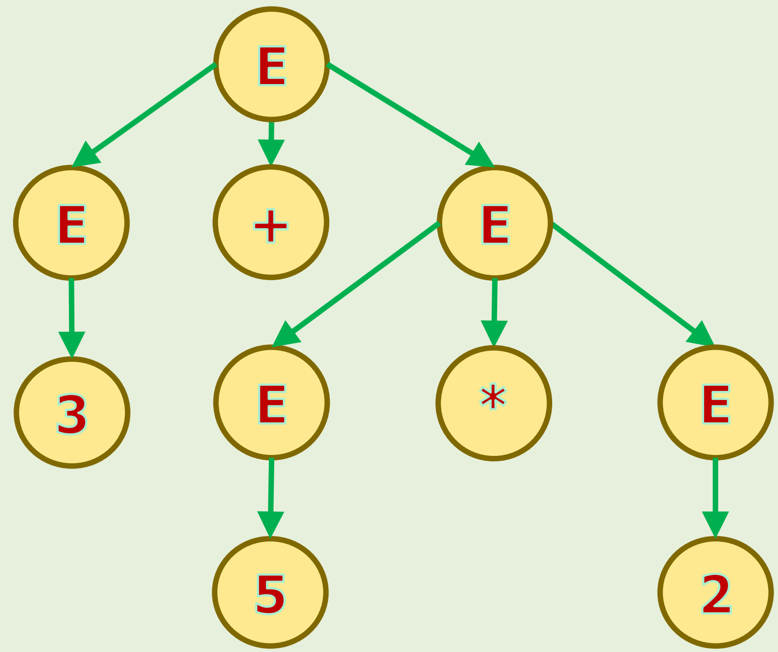
Non-Uniqueness of Parse Trees in Practice

Example 31 (cont'd)

Repeated

- Given grammar G as:
  1. $E \rightarrow E * E$
  2. $E \rightarrow E + E$
  3. $E \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
- E is starting variable.
- Construct a parse-tree for the mathematical expression: 3 + 5 * 2

Parse Tree #2



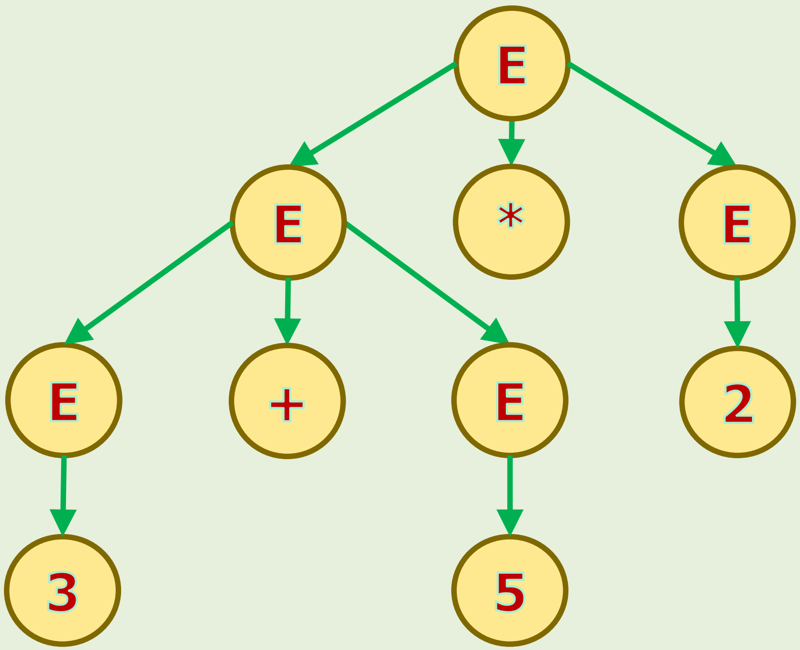
- Is this a good parsing?
- Yes!

Non-Uniqueness of Parse Tree in Practice

Example 31 (cont'd)

Parse Tree #1

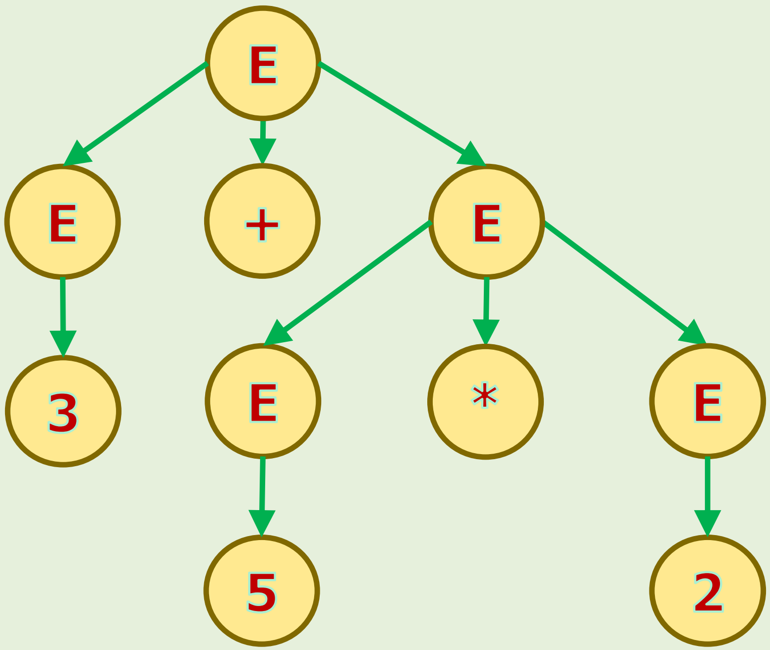
X



- Bad Parse Tree

Parse Tree #2

✓



- Good Parse Tree

Ambiguity in Grammars

Definition

- A grammar G is said to be ambiguous if there exists some $w \in L(G)$ that has at least two different parse-trees.

- In some cases, we can convert an ambiguous grammar to non-ambiguous one.
- But most of the time, it is hard and needs more knowledge.
- You might learn these skills in "Compiler Course".

- Let's rewrite the grammar of our previous example and remove the ambiguity.

Ambiguity in Grammars

Example 32

- Convert the following grammar to an unambiguous grammar.

  1. $E \rightarrow E * E$
  2. $E \rightarrow E + E$
  3. $E \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

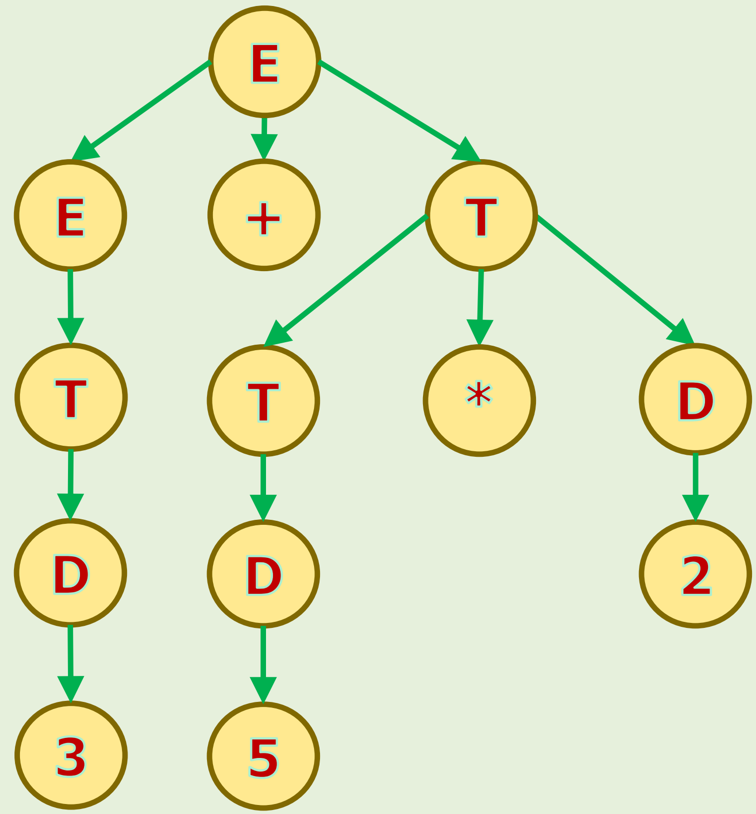
- E is starting variable.

Solution

1. $E \rightarrow E + T \mid T$
2. $T \rightarrow T * D \mid D$
3. $D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

- Construct a parse-tree for:
  $3 + 5 * 2$

Parse Tree

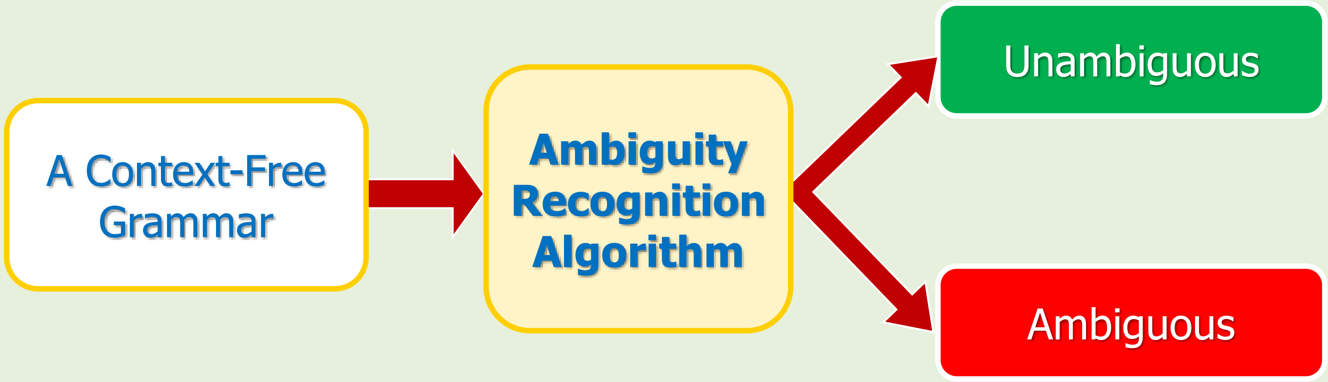


- There is no other parse-tree for this string.

Two Open Questions

1. Given a context-free grammar G.

- Is there an efficient algorithm to find out whether G is ambiguous or not?

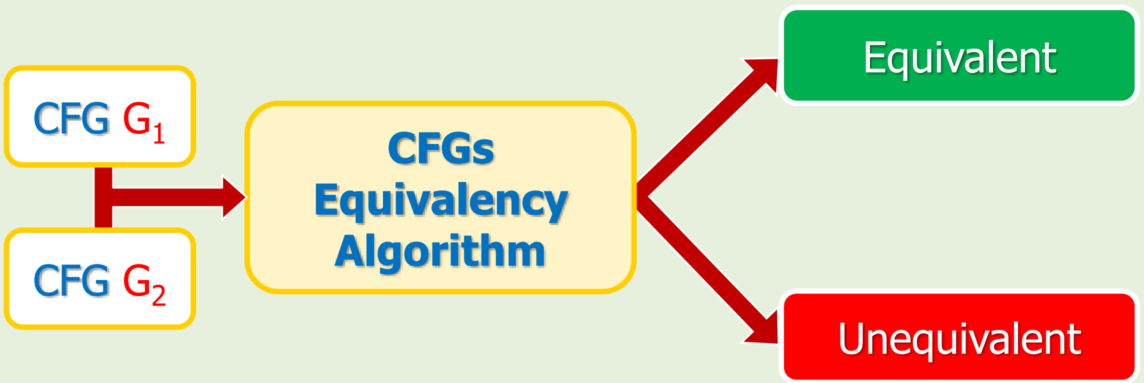


- As of this moment, there is no general algorithm to answer this question.

Two Open Questions

2. Are two given context-free grammars G_1 and G_2 equivalent?

- Is there an efficient algorithm to answer this question?



- Again, as of this moment, there is no general algorithm to answer this question.

References

1. Linz, Peter, "An Introduction to Formal Languages and Automata, 5th ed.," Jones & Bartlett Learning, LLC, Canada, 2012
2. Michael Sipser, "Introduction to the Theory of Computation, 3rd ed.," CENGAGE Learning, United States, 2013
   ISBN-13: 978-1133187790
3. The ELLCC Embedded Compiler Collection, available at: http://ellcc.org/