Compilers

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

- Grammar

Today's Lecture

Grammar

- Grammar - Describes how to form strings from a language's alphabet that are valid according to the language's syntax.
- A grammar does not describe the meaning of strings.

- This presentation describes context-free grammars.
- A context-free grammar is a set of rules that defines how to form sentences.

- Taken from the following sources:
  https://en.wikipedia.org/wiki/Formal_grammar
  Engineering a Compiler by Cooper and Torczon 2nd edition

Why Do We Need a Grammar?

- We can already use regular expressions and finite automata to recognize strings so why bother with a grammar?
- Regular expressions and finite automata are not powerful enough to recognize programming languages (programming languages are too complicated for them).
- We need a context-free grammar in order to recognize a programming language.

- Regular expressions and finite automata ONLY RECOGNIZE regular languages.
- Context-free grammars can recognize regular languages as well as other types of languages.
- So, regular expressions and finite automata only recognize a subset of the languages that a context-free grammar can recognize.
- If we wanted to, we could write a context-free grammar that is equivalent to a given regular expression or finite automata (the opposite is not true though).

Regular Expressions/Finite Automata vs Context-free Grammars

Grammar Description

- A grammar G consists of four components.

- G = (N, T, P, S)
  - N – Set of nonterminals (these are kind of like variables)
  - T – Set of terminals (these are kind of like constants)
  - P – Set of productions (rewrite rules)
  - S – Start state (this is a member of N (the set of nonterminals))

Example Context-free Grammar

- The following grammar recognizes strings of only the character a of any length.
- Here are strings from the language: a, aa, aaa, aaaa, ….

- G = (N, T, P, S)
  - N = { S } ← **Nonterminals (only S in this example)**
  - T = { a } ← **Terminals (only a in this example)**
  - P = {
    - S → aS,
    - S → a

    } ← **Productions (2 productions in this example). Only nonterminals are allowed on the left side of a production in a context-free grammar.**
  - S = { S } ← **Start symbol (must be a nonterminal)**

Recognizing Strings

- To check if a string belongs to the language the grammar recognizes you must apply productions.
- You begin with the start symbol and apply productions from there.
- If you can derive the target string by applying productions, then the string is in the language.
- For example, is the following string recognized by the grammar: a
- G = (N, T, P, S)
  - N = { S }
  - T = { a }
  - P = {

    S → aS,

    S → a

    }
  - S = { S }

Check next slide for answer

String Derivation

- G = (N, T, P, S)
  - N = { S }
  - T = { a }
  - P = { S → aS,
    S → a }
  - S = { S }
- Derive the string a.

1. Start with S

S => a

2. Apply S→a. Replace the S on the left side with a.

Success! The string a has been derived using the grammar.

- Note: Different arrows should be used for production definition (→) and string derivation (=>).

- G = (N, T, P, S)
  - N = { S }
  - T = { a }
  - P = { S → aS,
S → a }
  - S = { S }
- Derive the string aa.

Check next slide for answer

String Derivation

- G = (N, T, P, S)
  - N = { S }
  - T = { a }
  - P = { S → aS,
    S → a }
  - S = { S }
- Derive the string aa.

1. Start with S

S => aS

=> a**a**

2. Apply S→aS. Replace the S on the left side with aS.

3. Apply S→ a. Replace the S from the line above with a.

Success! The string aa has been derived using the grammar.

String Derivation

String Derivation

- G = (N, T, P, S)
  - N = { S }
  - T = { a }
  - P = { S → aS,
    S → a }
  - S = { S }

- Derive the string aaa.

Check next slide for answer

String Derivation

- G = (N, T, P, S)
  - N = { S }
  - T = { a }
  - P = { S → aS,
    S → a }
  - S = { S }
- Derive the string aaa.

1. Start with S

S => aS

=> a**<u>aS</u>**

=> aa**<u>a</u>**

2. Apply S→aS. Replace the S on the left side with aS.

3. Apply S→aS. Replace the S from the line above with aS.

4. Apply S→ a. Replace the S from the line above with a.

Success! The string aaa has been derived using the grammar.

String Derivation

- G = (N, T, P, S)
  - N = { S }, T = { a, b }, S = { S }
  - P = { S → aS,
S → a,
S → b }
- Derive the string ab.

Check next slide for answer

- G = (N, T, P, S)
  - N = { S }, T = { a, b }, S = { S }
  - P = { S → aS,
    S → a,
    S → b }
- Derive the string ab.

1. Start with S

S => aS

=> a**b**

2. Apply S→ aS. Replace the S on the left side with aS.

3. Apply S→ b. Replace the S from the line above with b.

Success! The string ab has been derived using the grammar.

String Derivation

String Derivation

- G = (N, T, P, S)
  - N = { S }, T = { a, b }, S = { S }
  - P = { S → aS,
S → a,
S → b }
- Derive the string b.

String Derivation

- G = (N, T, P, S)
  - N = { S }, T = { a, b }, S = { S }
  - P = { S → aS,
    S → a,
    S → b }
- Derive the string b.

Success! The string b has been derived using the grammar.

1. Start with S

2. Apply S→b. Replace the S on the left side with a.

S => b

String Derivation

- G = (N, T, P, S)
  - N = { S }, T = { a, b }, S = { S }
  - P = { S → aS,
    S → a,
    S → b }
- Derive the string bb.

Check next slide for answer

String Derivation

- G = (N, T, P, S)
  - N = { S }, T = { a, b }, S = { S }
  - P = { S → aS,
    S → a,
    S → b }
- Derive the string bb.

UNSUCCESSFUL! **The string bb cannot be recognized by this grammar.**

1. Start with S

S => ???

There is no way to recognize a string that has more than one character that starts with a b

- There is no production that starts with b on the right side that also has a nonterminal.
- We need a nonterminal somewhere in the right side to recognize strings that have more than one character.

String Derivation

- G = (N, T, P, S)
  - N = { S }, T = { a, b }, S = { S }
  - P = { S → aS,
S → bS,
S → a,
S → b }
- Derive the string bb.

Different productions are being used in this grammar

- G = (N, T, P, S)
  - N = { S }, T = { a, b }, S = { S }
  - P = { S → aS,
    S → bS,
    S → a,
    S → b }
- Derive the string bb.

1. Start with S

S => bS

=> b**b**

2. Apply S→ bS. Replace the S on the left side with bS.

3. Apply S→ b. Replace the S from the line above with b.

Success! The string bb has been derived using the grammar.

String Derivation

String Derivation

- G = (N, T, P, S)
  - N = { S }, T = { a, b }, S = { S }
  - P = { S → aS,
    S → bS,
    S → a,
    S → b }
- Derive the string abb.

String Derivation

- G = (N, T, P, S)
  - N = { S }, T = { a, b }, S = { S }
  - P = { S → aS,
    S → bS,
    S → a,
    S → b }
- Derive the string abb.

Success! The string abb has been derived using the grammar.

1. Start with S

S => aS

2. Apply S→ aS. Replace the S on the left side with aS.

=> a**bS**

3. Apply S→ bS. Replace the S from the line above with bS.

=> ab**b**

4. Apply S→ b. Replace the S from the line above with b.

Grammars with More Than One Nonterminal

- Grammars can have more than one nonterminal.
- The start symbol can be any of the nonterminals.
- For example:

- G = (N, T, P, S)
  - N = { E, S, V }
  - T = { 0, 1, + }
  - S = { E }
  - P = { E → S,
S → V + V,
V → 0,
V → 1 }

There are three nonterminals in this grammar

Start symbol is E

- G = (N, T, P, S)
  - N = { E, S, V }, T = { 0, 1, + }, S = { E }
  - P = { E → S,
    S → V + V,
    V → 0,
    V → 1 }
- Derive the string 0+1.

String Derivation

- G = (N, T, P, S)
  - N = { E, S, V }, T = { 0, 1, + }, S = { E }
  - P = { E → S,
    S → V + V,
    V → 0,
    V → 1 }
- Derive the string 0+1.

Success! The string 0+1 has been derived using the grammar.

1. Start with E

E => S

=> **V+V**

=> **0**+V

=> a+**1**

2. Apply E→ S. Replace the E on the left side with S.

3. Apply S→ V+V. Replace the S from the line above with V+V.

4. Apply V→ 0. Replace the V from the line above with a.

5. Apply V→ 1. Replace the V from the line above with 1.

String Derivation

- G = (N, T, P, S)
  - N = { E, S, T, V }, T = { 0, 1, +, * }, S = { E }
  - P = { E → S,
    S → T + S,
    S → T,
    T → V * T,
    T → V,
    V → 0, V → 1 }
- Derive the string 0*1.

String Derivation

String Derivation

- G = (N, T, P, S)
  - N = { E, S, T, V }, T = { 0, 1, +, * }, S = { E }
  - P = { E → S,
    S → T + S,
    S → T,
    T → V * T,
    T → V,
    V → 0, V → 1 }
- Derive the string 0*1.

  E => S
  => T (apply S→T)
  => V*T (apply T→V*T)
  => 0*T (apply V→0)
  => 0*V (apply T→V)
  => 0*1 (apply V→1)

Success! The string 0*1 has been derived using the grammar.

String Derivation

- G = (N, T, P, S)
  - N = { E, S, T, V }, T = { 0, 1, +, * }, S = { E }
  - P = { E → S,
    S → T + S,
    S → T,
    T → V * T,
    T → V,
    V → 0, V → 1 }
- Derive the string 0*1+1.

- G = (N, T, P, S)
  - N = { E, S, T, V }, T = { 0, 1, +, * }, S = { E }
  - P = { E → S,
    S → T + S,
    S → T,
    T → V * T,
    T → V,
    V → 0, V → 1 }
- Derive the string 0*1+1.

```
E => S
  => T+S      (apply S→T+S)
  => V*T+S    (apply T→V)
  => 0*T+S    (apply V→0)
  => 0*V+S    (apply T→V)
  => 0*1+S    (apply V→1)
  => 0*1+T    (apply S→T)
  => 0*1+V    (apply T→V)
  => 0*1+1    (apply V→1)
```

Success! The string 0*1+1 has been derived using the grammar.

String Derivation

Context-free Grammar with Empty Production

- The following grammar recognizes strings of only the character a of any length.
- Here are strings from the language: a, aa, aaa, aaaa, ….

- G = (N, T, P, S)
  - N = { S } ← **Nonterminals (only S in this example)**
  - T = { a } ← **Terminals (only a in this example)**
  - P = {

    S →aS,

    S → ε

    }

    ← **Productions (2 productions in this example). Only nonterminals are allowed on the left side of a production in a context-free grammar.**
  - S = { S } ← **Start symbol (must be a nonterminal)**

The production S→ ε is the empty production. It basically goes to nothing. Applying this production eliminates the nonterminal (S in this case).

Empty Production in String Derivation

- G = (N, T, P, S)
  - N = { S }
  - T = { a }
  - P = { S →aS,
    S → ε }
  - S = { S }
- Derive the string a.

Success! The string a has been derived using the grammar.

1. Start with S

S => aS

=> a**ε**

=> a

2. Apply S→aS. Replace the S on the left side with aS.

3. Apply S→ ε. Replace the S from the line above with ε.

ε is empty string so it disappears

- Note: Different arrows should be used for production definition (→) and string derivation (=>).

- G = (N, T, P, S)
  - N = { S }
  - T = { a }
  - P = { S →aS,
    S → ε }
  - S = { S }

- Derive the string aa.

String Derivation

String Derivation

- G = (N, T, P, S)
  - N = { S }
  - T = { a }
  - P = { S →aS,
    S → ε }
  - S = { S }

Success! The string a has been derived using the grammar.

- Derive the string aa.

1. Start with S

S => aS

2. Apply S→aS. Replace the S on the left side with aS.

=> a**<u>aS</u>**

3. Apply S→aS. Replace the S from the line above with aS.

=> aa**<u>ε</u>**

4. Apply S→ ε. Replace the S from the line above with ε.

=> aa

ε is empty string so it disappears

- The following grammar defines strings that have a's followed by b's where there are the **same number** of a's and b's.

- $a^n b^n$ for some n>=0.

- Here are some strings from the language: ab, aabb, aaabbb, aaaabbbb, and so on.

- G = (N, T, P, S)
  - N = { S }
  - T = { a, b }
  - P = { S → aSb, S → ε }
  - S = { S }

- This language is NOT regular! There is no finite automata or regular expression that can recognize it. Note: The regular expression a*b* will not work because it allows strings that contain different numbers of a's and b's.

Example Grammar

- G = (N, T, P, S)
  - N = { S }
  - T = { a, b }
  - P = { S → aSb, S → ε }
  - S = { S }
- Derive the string aabb.

String Derivation Showing Productions

String Derivation Showing Productions

- G = (N, T, P, S)
  - N = { S }
  - T = { a, b }
  - P = { S → aSb, S → ε }
  - S = { S }
- Derive the string aabb.

1. Start with S

2. Apply S→aSb. Replace the S on the left side with aSb.

S => aSb

=> a**aSb**b

3. Apply S→aSb. Replace the S from the line above with aSb

=> aa**ε**bb

4. Apply S→ ε. Replace the S from the line above with ε.

=> aabb

ε is empty string so it disappears

String Derivation Tree

- You can also derive a string by creating a string derivation tree (next slide).
- The root of the tree is the start symbol. You create levels in the tree by applying productions.
- Derive the string aabb using a string derivation tree.

String Derivation Tree

- G = (N, T, P, S)
  - N = { S }
  - T = { a, b }
  - P = { S → aSb, S → ε }
  - S = { S }
- Derive the string aabb.
- Start with the start symbol and apply productions.

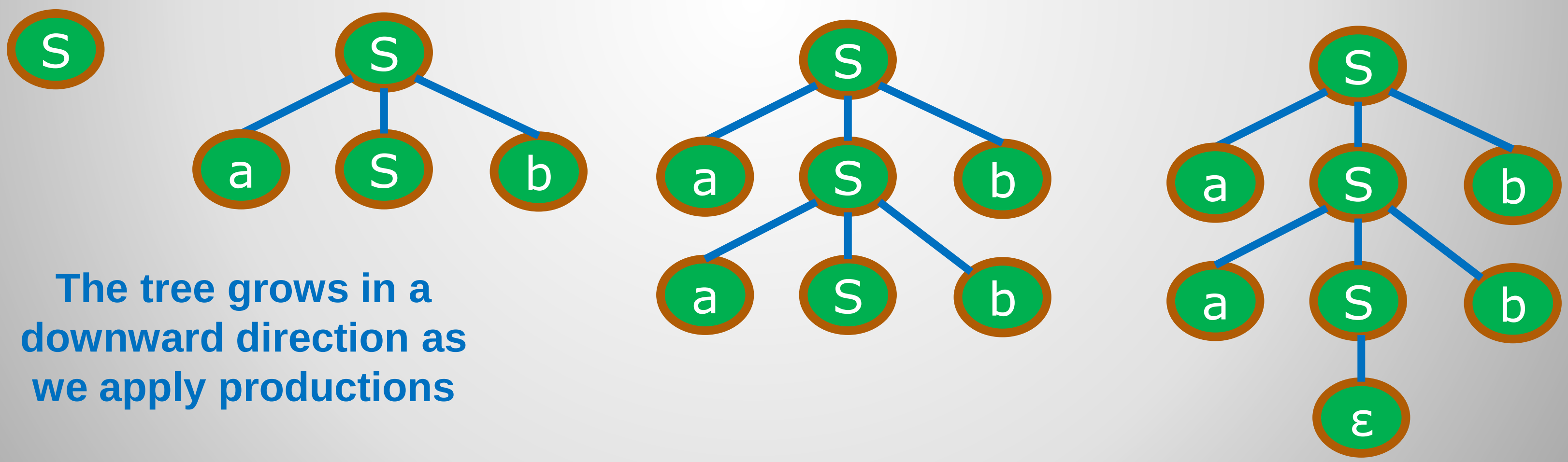
The leaf nodes in the tree correspond to the string that we derived. The derived string can be constructed from the tree by visiting leaf nodes going from left to right (see next slide).

1. Start with S

2. Apply S→aSb

3. Apply S→aSb

4. Apply S→ ε



The tree grows in a downward direction as we apply productions

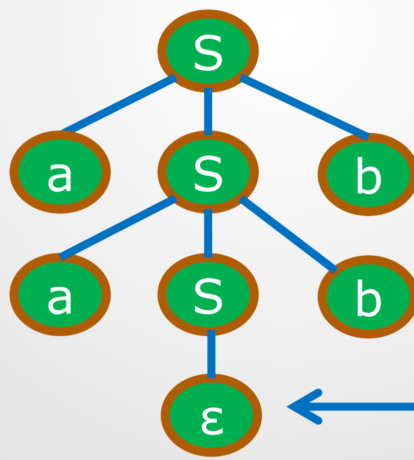
String Derivation Tree

- G = (N, T, P, S)
  - N = { S }
  - T = { a, b }
  - P = { S → aSb, S → ε }
  - S = { S }
- Derive the string aabb.

Visit leaf nodes from left to right to see the string that was recognized

This a is the <u>first</u> letter in the string

This a is the <u>second</u> letter in the string (this a is to the right of the other a in the tree)



This b is the <u>forth</u> letter in the string

This b is the <u>third</u> letter in the string (this b is to the left of the other b in the tree)

ε is the empty string

Leftmost vs Rightmost Derivations

- **Leftmost Derivation** - Always expand the leftmost nonterminal in the production.
- **Rightmost Derivation** - Always expand the rightmost nonterminal in the production.

Leftmost Derivation Example

- G = (N, T, P, S)
  - N = { S }
  - T = { 1, a, + }
  - P = { S→S+S, S→1, S→a }
  - S = { S }
- Do leftmost derivation for the string:1+1+a

Leftmost Derivation Example

Leftmost Derivation Example

- G = (N, T, P, S)
  - N = { S }
  - T = { 1, a, + }
  - P = { S→S+S, S→1, S→a }
  - S = { S }
- Do a leftmost derivation for the string:1+1+a

S => S+S
↓
=> **S+S**+S
↓
=> **1**+S+S
↓
=> 1+**1**+S
↓
=> 1+1+**a**

A leftmost derivation means we should always expand the leftmost nonterminal symbol

Leftmost Derivation Example

Rightmost Derivation Example

- G = (N, T, P, S)
  - N = { S }
  - T = { 1, a, + }
  - P = { S→S+S, S→1, S→a }
  - S = { S }
- Do rightmost derivation for the string:1+1+a

Rightmost Derivation Example

Rightmost Derivation Example

- G = (N, T, P, S)
  - N = { S }
  - T = { 1, a, + }
  - P = { S→S+S, S→1, S→a }
  - S = { S }
- Do a leftmost derivation for the string:1+1+a

S => S+S

=> S+**S+S**

=> S+S+**a**

=> S+**1**+a

=> **1**+1+a

A rightmost derivation means we should always expand the rightmost nonterminal symbol

Rightmost Derivation Example

Ambiguous Grammar

- **Ambiguous Grammar** – If one of the following is true then the grammar is ambiguous:
  - There is more than one leftmost derivation of a string OR
  - There is more than one rightmost derivation of a string.

Backus Naur Form

- Grammars can be described in slightly different formats.
- Format used in all previous slides was:
  - Nonterminals start with a capital letter.
  - Use → for productions.
  - P = { S → aS,
  - S → a }
- Backus-Naur Form (BNF)
  - Surround nonterminals with < >.
  - Surround terminals with " ".
  - Use ::= instead of → in productions.
  - If the same nonterminal left side is used for multiple productions, you can put all the right side on one line and separate them with |.
  - P = {
  - <s> ::= "a" <s> | "a" }
- Note: Some descriptions of BNF use ' ' or underline or italic for terminals instead.

- **End of Slides**

End of Slides