

# CS-259

## Data Structures with Java

### *Deterministic Context-Free Languages*

Instructor:

Joel Castellanos

e-mail: [joel@unm.edu](mailto:joel@unm.edu)

Web: <http://cs.unm.edu/~joel/>

Office: Farris Engineering  
Center (FEC) room 321

|            |                    |
|------------|--------------------|
| variables: | f h                |
| constants: | + -                |
| axiom:     | f                  |
| rules:     | f → f-h<br>h → f+h |

11/30/2009

## Context-Free Grammars

- A context-free grammar is a formalism developed by linguist Noam Chomsky in the mid-1950s.
- A context-free grammar provides a simple and precise mechanism for describing the methods by which phrases in some natural language are built from smaller blocks.
- Its simplicity makes the formalism amenable to rigorous mathematical study.
- The "block structure" aspect that context-free grammars capture is so fundamental to grammar that the terms syntax and grammar are often identified with context-free grammar rules.
- Formal constraints not captured by the grammar are then considered to be part of the "semantics" of the language.
- Context-free grammars are powerful enough to be useful yet simple enough to allow the construction of efficient parsing algorithms.

2

## Context-Free Grammars - Definition

A Context-Free Grammar is a formal grammar consisting of 4 parts:

1. **A set of Variables:** Symbols that can be replaced by production rules.
2. **A set of Terminals:** Symbols that do not appear on the predecessor side of any of the system's production rules.
3. **An Axiom:** A string composed of some number of variables and/or constants. The axiom is the initial state of the system.
4. **A set of Production Rules:** Which define the way variables can be replaced with combinations of terminals and other variables.
  - A production consists of two strings: the predecessor and the successor.
  - Being "Context-Free" requires that every predecessor be exactly one variable.

3

## Example Context-Free Grammar

1. **Axiom:** S
2. **Production Rules:**  $\{S \rightarrow aSb, S \rightarrow aSXb, S \rightarrow, X \rightarrow b\}$
3. **Set of Variables:**  $\{S, X\}$
4. **Set of Terminals:**  $\{a, b\}$

Example Productions:

| S         | S          |
|-----------|------------|
| aSb       | aSb        |
| aaSbb     | aaSXbb     |
| aaaSbbb   | aaaSbbb    |
| aaaaSbbbb | aaaaSXbbbb |
| aaaabbbb  | aaaabbbbb  |

4

## Deterministic Context-Free Language

- A language is a set of strings.
- A language  $L$ , is said to be a context-free language (CFL) if there exists a CFG,  $G$ , that recognizes the language  $L$ .
- A grammar recognizes a language iff: Given a string, the grammar can determine whether that string is a member of the language.
- A Deterministic Context-Free Language (DCFL) is one in which each variable appears in the predecessor of exactly one Production Rule.

5

## Simple Context Free Grammars

Which are deterministic and which are nondeterministic:

1.  $S \rightarrow abS \mid ab$
2.  $S \rightarrow SS \mid ab$
3.  $S \rightarrow aX$                        $X \rightarrow bS \mid a$
4.  $S \rightarrow aX$                        $X \rightarrow bS \mid ab$
5.  $S \rightarrow aX \mid ab$                  $X \rightarrow bS$
6.  $S \rightarrow aX$                        $X \rightarrow bS$
7.  $S \rightarrow aX$                        $X \rightarrow aXY$                  $Y \rightarrow ab$

6

## Find a CFG that recognizes $L$

Where  $L$  is the language of even length palindromes:

$L = \{ww' \mid w' \text{ is the reverse of } w, \text{ and } w = w_1w_2w_3\dots w_n \text{ where } w_i \in \{a, b, c, d\}\}$

For example:

$x = abccba$ , where  $w = abc$  and  $w' = cba$

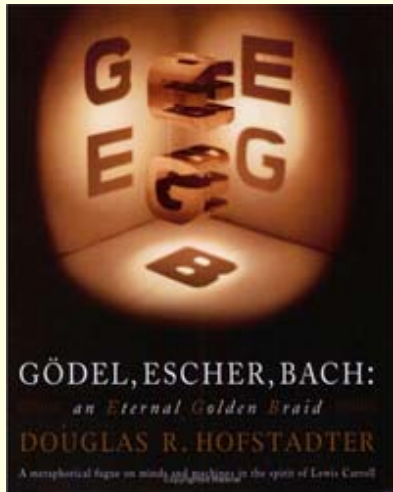
$y = ddaaaadd$ , where  $w = ddaa$  and  $w' = aadd$

$S \rightarrow aSa \mid bSb \mid cSc \mid dSd \mid \emptyset$

example:  $aSa \rightarrow abSba \rightarrow abcScba \rightarrow abccba$

7

## CS Great Book: Gödel, Escher Bach



By  
Douglas R. Hofstadter.

Pulitzer Prize winning,  
metaphorical Fugue  
on Minds and  
Machines in the spirit  
of Lewis Carroll.

8

## MU-Puzzle from Gödel, Escher Bach

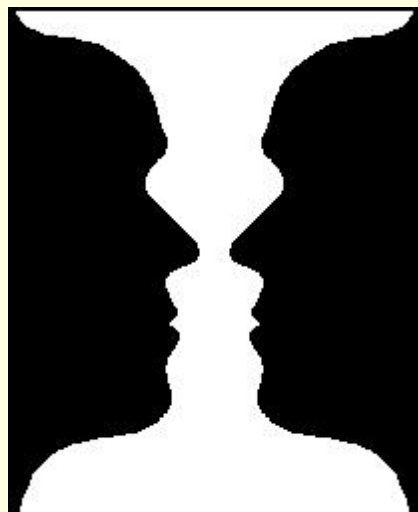
1. Is the System below a Context Free Language?
2. Can the string **MU** be derived?
  - **Alphabet:** {**M U I**}
  - **Axiom:** **MI**
  - **Rule 1:** If you have a theorem whose last letter is **I**, then you may add **U** to the end. Symbolically,  $xI \Rightarrow xIU$ , where  $\Rightarrow$  stands for implies.
  - **Rule 2:** If  $Mx$  is a theorem, so is  $Mxx$ :  $Mx \Rightarrow Mxx$ .
  - **Rule 3:** In any theorem, **III** can be replaced with **U**:  $xIIIy \Rightarrow xUy$ .
  - **Rule 4:** **UU** can be dropped from any theorem:  $xUUy \Rightarrow xy$ .

9

## Inside and Outside of the System

MUI-System:

- Inside:
  - Deriving theorems
- Outside
  - All theorems begin with the letter M.



10

## DCFL Example

|            |  |
|------------|--|
| variables: | f h  |
| terminals: | + -  |
| axiom:     | f  |
| rules:     | f $\rightarrow$ f-h<br>h $\rightarrow$ f+h |

generation 1:

generation 2:

generation 3:

generation 4: f-h - f+h - f-h + f+h - f-h - f+h + f-h + f+h

11

## Is this a DCFL?

|            |  |
|------------|--|
| variables: | s d  |
| terminals: | + -   [ ] f c  |
| axiom:     | sd   |
| rules:     | s $\rightarrow$ sf[-sd][+sd]s<br>d $\rightarrow$ f- ++ ++++ ++ c |

Yes

12

## Is this a DCFL?

|            |  |
|------------|--|
| variables: | <b>f a c</b>                                       |
| terminals: | <b>+ -</b>   |
| axiom:     | <b>f++f++f</b>                                     |
| rules:     | <b>f → af-cf++f-af</b><br><b>a →</b><br><b>c →</b> |

Yes

13

## Is this a DCFL?

|            |  |
|------------|--|
| variables: | <b>f a c</b>   |
| terminals: | <b>+ -</b>   |
| axiom:     | <b>fa++fc++</b>  |
| rules:     | <b>fa → fa++fc++</b><br><b>fc → fa--fc--</b><br><b>c →</b> |

No – It is not context free because production rules contain more than one symbol on the left (predecessor side).

14

## Is this a DCFL?

|            |                                 |
|------------|---------------------------------|
| variables: | f h                             |
| terminals: | + -                             |
| axiom:     | f++f++f                         |
| rules:     | f → f-h<br>f → f+h<br>h → ff+hh |

No – It is not deterministic because there is more than one rule with **f** as the predecessor.

15

## DCFL Project part 1: Due Friday, Nov. 20

### Parsing Deterministic Context-Free Languages

- Create a class called DCFL with a constructor that accepts two arguments:  

```
DCFL(String axiom, String[] rules)
```
- The function must check the arguments to verify that they define a Deterministic Context-Free Language (DCFL).
- If the arguments do not define a DCFL, then display an appropriate error message.
- If the arguments do define a DCFL, then display an affirmation along with a list of the language's variables and a list of the language's terminals.
- Ignore all white space in the input axiom and rules.

16

## DCFL Project part 1: Grading Rubric

Your code must use the given constructor API in order to pass any tests.

**2 Points:** Pass 2 tests of unknown bad input.

**6 Points:** Pass 3 tests of unknown languages that do not define a DCFL.

**8 Points:** Pass 4 tests of unknown DCFL.

**2 Points:** JavaDoc Comments

**2 Points:** Follow CS-259 Coding standard.

17

## DCFL constructor – Syntax Example 1

Given the following statements in main()

```
String axiom = "f++f++f";
String[] rules = {"f=af-cf++f-af",
                 "a=",
                 "c="};
DCFL dcf1 = new DCFL(axiom, rules);
```

Your constructor should output the following:

Yes, this is a deterministic context free language.

```
variables = {f a c}
terminals = {+ -}
```

18

## DCFL constructor – Syntax Example 2

Given the following statements in main()

```
String axiom = "f";
String[] rules = {"f=f-h",
                 "h=f+h"};
DCFL dcf1 = new DCFL(axiom, rules);
```

Your constructor should output the following:

**Yes, this is a deterministic context free language.**

```
variables = {f h}
terminals = {+ -}
```

19

## DCFL constructor – Syntax Example 3

Given the following statements in main()

```
String axiom = "fabbc";
String[] rules = {"f=afg ",
                 "g=bfh",
                 "h = ab"};
DCFL dcf1 = new DCFL(axiom, rules);
```

Your constructor should output the following:

**Yes, this is a deterministic context free language.**

```
variables = {f g h}
terminals = {a b c}
```

20

## DCFL constructor – Syntax Example 4

Given the following statements in main()

```
String axiom = "f++f++f";  
String[] rules = {"f=af-cf++f-af",  
                 "f=c+g",  
                 "c="};  
DCFL dcf1 = new DCFL(axiom, rules);
```

Your constructor should output the following:

```
Error: This is not deterministic because  
there are two rules for variable f.
```

21

## DCFL constructor – Syntax Example 5

Given the following statements in main()

```
String axiom = "f++f++f";  
String[] rules = {"fa=af-cf++f-af",  
                 "fb=c+g",  
                 "c="};  
DCFL dcf1 = new DCFL(axiom, rules);
```

Your constructor should output the following:

```
Error: This is not context free because  
the left side of a rule contains two  
symbols: fa, fb.
```

22

## DCFL helper method: String.trim()

```
//String.trim() removes whitespace from the beginning and end of a
// string. This includes spaces, tabs, return characters and all
// ASCII control characters.
public class HelloWorld
{ public static void main(String[] args)
  { String str = "  white  space  " + '\t' + " ";
    System.out.println("<" + str + ">");
    str.trim();
    System.out.println("<" + str + ">");
    str = str.trim();
    System.out.println("<" + str + ">");
  }
}
```

Output: - what is going on?

```
<  white  space  >
<  white  space  >
<white  space>
```

23

## DCFL helper method: removeWhitespace

```
//First, uses String.trim() to remove all whitespace from the front and
//end of the given str.
//The String str then references the new, trimmed string.
//Then builds a new String, str2, one character at a time from str.
//Each non-blank, non-tab character from str is added to str2.
//The method returns a reference to str2.
```

```
public static String removeWhitespace(String str)
{ str = str.trim(); //Creates a new String object
  String str2 = "";
  for (int i=0; i<str.length(); i++)
  {
    char c = str.charAt(i); //Access in constant time.
    if (c != ' ' && c != '\t') str2 += c;
  }
  return str2;
}
```

24

## Testing: removeWhitespace

```
public static void main(String[] args)
{ String x = "  I contain some whitespace      ";
  String y = "abcdefghijklmnop";
  //The String object referenced by x is passed into
  //  removeWhitespace(x).

  //After the method returns, x is assigned the reference to a new
  //  object created by removeWhitespace(x).

  //The String object that x used to reference is no longer referenced
  //  therefore, Java marks it for garbage collection.

  x = removeWhitespace(x);
  y = removeWhitespace(y);

  //By surrounding the Strings with <> symbols, we can
  //  see if there are leading or trailing spaces.
  System.out.println("<" + x + ">");
  System.out.println("<" + y + ">");
}
25
```

## Why is removeWhitespace Inefficient?

```
public static String removeWhitespace(String str)
{ str = str.trim();
  String str2 = "";

  //For large strings, this loop is VERY inefficient. Why?
  for (int i=0; i<str.length(); i++)
  { char c = str.charAt(i);
    if (c != ' ' && c != '\t')
    { str2 += c;
    }
  }
  return str2;
}
```

The String class is immutable.  
String concatenation creates a new  
String object and marks the original  
for garbage collection.

26

## Improved removeWhitespace

```
public static String removeWhitespace(String str)
{ str = str.trim();
  //The resulting string can be no larger than str.
  char[] result = new char[str.length()];
  int chrIdx = 0; //index into result[] where to add next char.

  for (int i=0; i<str.length(); i++)
  { char c = str.charAt(i);

    if (c != ' ' && c != '\t')
    { result[chrIdx] = c;
      chrIdx++;
    }
  }
  //String.copyValueOf(char[] data, int offset, int count)
  return String.copyValueOf(result, 0, chrIdx);
}
```

Creates new large object  
only 3 times, not  $n$  times.

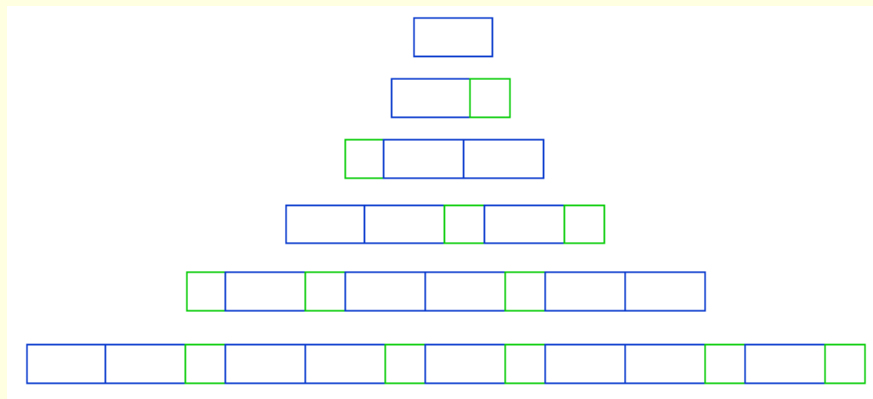
Where are these objects  
created?

27

## Blue-Green Algae

Can this pattern be generated by a Context-Free Grammar?

Can it be generated by a Deterministic Context-Free Grammar?



28

## Correct Context-Free Grammar?

Blue-Green Algae pattern

1. **b**
2. **bg**
3. **gbb**
4. **bbgbg**
5. **gbgbbgbb**
6. **bbgbbgbgbbgbg**

- **b** → **bg** | **gb**
- **g** → **b**

Can the grammar generate every generation of the blue-green algae pattern?

Can the grammar generate a string that is NOT in the blue-green algae pattern?

Bad Production:

**b** → **gb**

29

## Phenotypes and Genotypes

The first **b** production rule must be used on even generations and the second on odd generations.

**b** → **bg** | **gb**

**g** → **b**

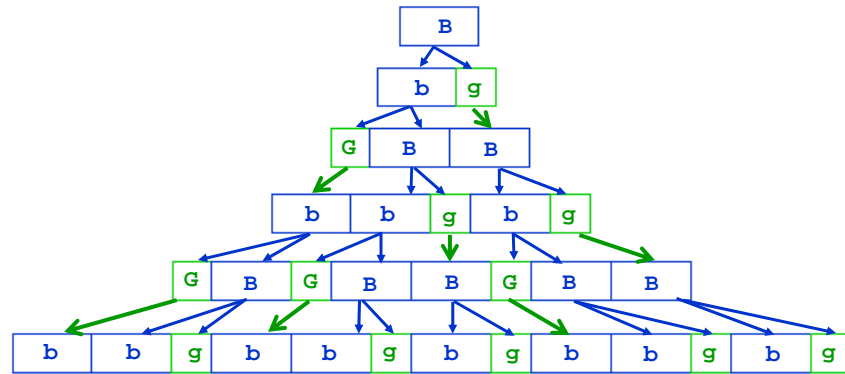
The concepts of Phenotypes and Genotypes borrowed from biology can be used to encode the above "odd" and "even" use of rules in a CFL.

**Phenotype** describes any observed quality of an organism, such as eye color.

**Genotype** describes the allelic genetic composition such as **bb**, **bB**, **Bb**, or **BB** for brown or blue eyes.

30

## DCFG for Blue-Green Algae



B → bg  
b → GB  
G → b  
g → B

31

## DCFL Project Milestone 2: String Generation

Keep the DCFL constructor from Milestone 1.

Add to that class the method:

```
public String produce(int generation)
```

Where:

- **generation** is an integer  $\geq 0$ .
- The method throws an `IllegalArgumentException` when **generation**  $< 0$  or  $> 25$ .
- If **generation** = 0, the method returns the axiom
- If **generation**  $> 0$ , the method returns the string that is **generation** of the DCFL.

Assume `DCFL.produce()` will not be asked to generate a string of more than greater than **10 million** characters in length.

32

## DCFL Project part 2: Example 1

```
String axiom = "f";
String[] rules = {"f=f-h", "h=f+h"};
DCFL dcfl = DCFL(axiom, rules);
String x = dcfl.produce(1);
String y = dcfl.produce(2);
String z = dcfl.produce(3);
System.out.println("Gen 1=" + x);
System.out.println("Gen 2=" + y);
System.out.println("Gen 3=" + z);
```

Output:

```
Gen 1=f-h
Gen 2=f-h-f+h
Gen 3=f-h-f+h-f-h+f+h
```

33

## DCFL Project part 2: Example 2

```
String axiom = "f-f-f-f";
String[] rules = {"f=ff-f--f-f"};
DCFL dcfl = DCFL(axiom, rules);
String x = dcfl.produce(0);
String y = dcfl.produce(1);
System.out.println(x);
System.out.println(y);
```

Output:

```
f-f-f-f
ff-f--f-f-ff-f--f-f-ff-f--f-f-ff-f--f-f
```

34

## DCFL Project part 2: Example 3

```
String axiom = "f++f++f";
String[] rules = {"f=af-cf++f-af", "a=", "c="};
DCFL dcfl = DCFL(axiom, rules);
String x = dcfl.produce(1);
String y = dcfl.produce(2);
System.out.println("x="+x+'\n');
System.out.println("y="+y);
```

Output:

```
x=af-cf++f-af++af-cf++f-af++af-cf++f-af
```

```
y=af-cf++f-af-af-cf++f-af++af-cf++f-af-af-
cf++f-af++af-cf++f-af-af-cf++f-af++af-cf++f-
af-af-cf++f-af++af-cf++f-af-af-cf++f-af++af-
cf++f-af-af-cf++f-af
```

35

## DCFL Project part 2: Grading

- 2 Points: Coding Standard
- 2 Points: JavaDoc
- 16 Points: 8 unknown test cases at 2 points each.

36