

# CS-257L

## Nonimperative Programming: Scheme!

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

Deterministic Context-Free  
Languages

variables:	f h
constants:	+ -
axiom:	f
rules:	f → f-h h → f+h

# Project – HW5 – Due Wednesday – Feb. 20

---

Write a function called DCFL-checker that accepts 5 arguments:

1. axiom
2. rules
3. initialAngle
4. turnAngle
5. shrinkage

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.

# Software Specifications

---

The name of the function shall be: **DCF~~L~~-checker** .

The number of arguments shall be 5.

Not 4.

Not 6.

Seven is way out.

Five shall be the number.

The number shall be 3 and 2 combined by addition into a single value: five.

# 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.

# Context-Free Grammars - Definition

---

An Context-Free 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.

# 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.

# DCFL Example

variables:	<b>f h</b>
terminals:	<b>+ -</b>
axiom:	<b>f</b>
rules:	<b>f</b> → <b>f-h</b> <b>h</b> → <b>f+h</b>

generation 1:

**f-h**

generation 2:

**f-h - f+h**

generation 3:

**f-h - f+h - f-h + f+h**

generation 4:

**f-h - f+h - f-h + f+h - f-h - f+h + f-h + f+h**

# Is this a DCFL?

variables:	<b>s d</b>
terminals:	<b>+ -   [ ] f c</b>
axiom:	<b>sd</b>
rules:	<b>s → sf[-sd][+sd]s</b> <b>d → f- ++ ++++ ++ c</b>

Yes

# 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> <b>a →</b> <b>c →</b>

Yes

# Is this a DCFL?

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

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

# Is this a DCFL?

variables:	<b>f h</b>
terminals:	<b>+ -</b>
axiom:	<b>f++f++f</b>
rules:	<b>f → f-h</b> <b>f → f+h</b> <b>h → ff+hh</b>

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

# 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  $w' = aadd$

$S \rightarrow aSa \mid bSb \mid cSc \mid dSd \mid ()$

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

# HW5 – Scheme Scope

---

- This programming assignment can be accomplished just with the Scheme features that we have thus far covered.
- My solution uses many of the user functions defined in chapters 1 through 5 of The Little Schemer. These may be used or modified without reference.
- For this assignment, you may use any features of Scheme supported by DrScheme's R5RS.

# HW5 – DCFL-checker – Input & output

(DCFL-checker

`axiom ; a list of atoms`

`rules ; a list of lists of atoms.`

`initialAngle ;real number [0, 360]`

`turnAngle ;real number [0, 180]`

`shrinkage ;real number [0.01, 100.0]`

)

- If the arguments do not define a DCFL, then display an appropriate error message.
- If the arguments do define a DCFL, then display:
  1. An affirmation,
  2. A list of the language's variables, and
  3. A list of the language's terminals.

# DCFL-checker – Syntax Example

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

(DCFL-checker

(f + + f + + f) ;axiom

((f a f - c f + + f - a f) (a) (c)) ;rules

90 ;initialAngle

90 ;turnAngle

0.5 ;shrinkage

)

# DCFL-checker – Output Example

```
(DCFL-checker
  (f + + f + + f)
  ;axiom
  ((f a f - c f + + f - a f) (a)
  (c)) ;rules
  90           ;initialAngle
  90           ;turnAngle
  0.5         ;shrinkage
)
```