# **Section 3.3 Regular Grammars**

Regular grammars is another way to describe regular languages.

## **English Grammar**

Rules include:

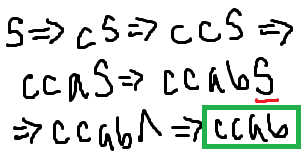
* Sentence -> subject, predicate
* Subject -> article, adjective, noun
* Predicate -> verb, object

Sometimes, you draw a parse tree (shown right) for the sentence. All the terminal nodes (leaves of the tree) are actual words in English.

## **Grammar for an Arbitrary Language (i.e. set of strings)**

If L is a language over an alphabet A, then a grammar for L consists of a set of grammar rules of the form

where and denote strings of symbols taken from A and from a set of grammar symbols disjoint from A. This is called a “production rule” or “grammar rule”. For example,



**Example:** Let A = {a, b, c} and S be our “start symbol”. Can you derive ccab from these grammar rules for A\*?:

* S -> ^
* S -> aS
* S -> bS
* S -> cS

The left hand size of the grammar rules are called “non-terminals”, or the internal nodes on the tree. They’re not part of the output, so you have to use the lambda.

Every grammar has a special grammar symbol called the start symbol.

There must be at least one production rule that only has a start symbol on the left hand side.

You can do parse trees for these too!

**Formal Definition of a Grammar**

A grammar consists of

* A finite set *N* of grammar symbols called non-terminals
  + Things not in the alphabet and on the LHS of rules
* A finite set *T* of symbols called terminals, where *N* ∩ T = Ø
* A non-terminal, not necessarily always S, called the start symbol
* A finite set of production rules of the form where they are both strings over the alphabet *N* ∪ *T* with the restrictions that
  + is not the empty string
  + There is at least one production with S alone on the left hand side
  + Each non-terminal must appear on the LHS of some production

**Some Notes**

Historically, ^ can never be part of an alphabet. It continues to represent the empty string and is a special character. HOWEVER, if ^ is part of a grammar, then it is one of the symbols in the set of terminals.

* For now, all grammars will just have one non-terminal on the LHS.
* To write rules, you can shorthand it with a vertical bar
  + Rules from the example can be simplified like so: S -> ^ | aS | bS | cS

**Building Grammars**

Find a grammar for the following languages: {a*n* | n ∈ N}, {a*n*b*n* | n ∈ N}, {(ab)*n* | n ∈ N}.

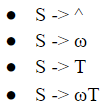
This is the same as finding the grammar for the language a\*. So, you can do S -> aS | ^, as in you can either go to an a (and continue getting a’s) or ^ for the time when *n* = 0 or when you want to end the string.

This isn’t a regular grammar. You can’t do something like abS, because the language requires that there should be the same number of a’s and b’s grouped together at the beginning and end. It should be S -> aSb | ^. The ^ is for obvious reasons, and the aSb allows you to put as many ab’s in between, which effectively lays out the string. This is a context-free grammar.

The language represents a sequence of 0 or more copies of ab. Thus, the answer is

S -> abS | ^.

## **Regular Grammars**

A grammar is called a regular grammar if each production takes one of the following forms where the uppercase letters are nonterminals and ω is a *non-empty* string of terminals (shown right):

What’s special about regular grammars?

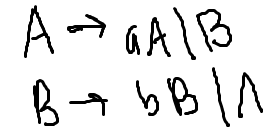
* Only one non-terminal on RHS
* It must appear on the far right side of the RHS
* Grammars are NOT unique (there can be 2 different grammars for the same language)
* The same can be said for regular expressions and NFAs

Example: Which of the following are regular?

The first one isn’t regular because the non-terminal (B) isn’t on the far right side of the RHS. The second isn’t regular because there are two non-terminals (T and U) on the RHS. However, the third IS regular because it follows all the rules.

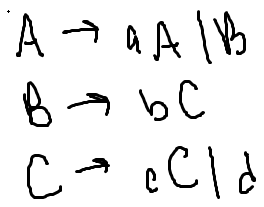
**Concatenating Regular Grammars**

Hard because the requirement to only have one non-terminal on RHS; for example: L = a\*b\*

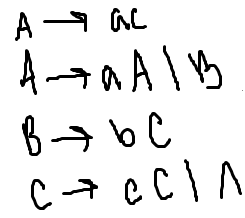
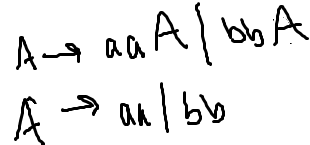
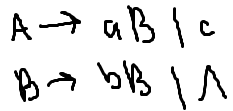


Grammar for a\*: Answer:

Grammar for b\*:

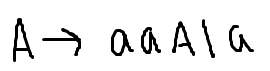
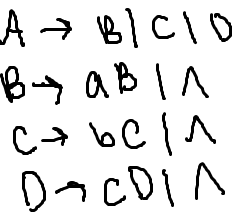
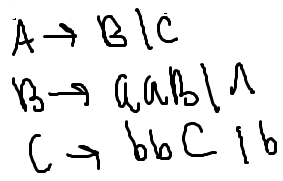


Problem: Find a regular grammar for the language a\*bc\*d. Answer:

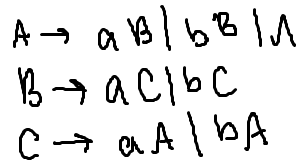
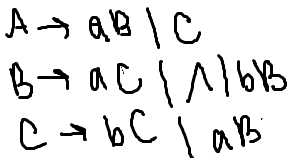
**Problem from book (#1):** Find a regular grammar for each of the following expressions: a + bc, ab\* + c, a\*bc\* + ac, (aa + bb)(aa + bb)\*

**Problem from the book (#2):** Find a regular grammar to describe each of the following languages: {aa, ab, ac}, {a, aaa, aaaaa, …, a2*n* + 1, … | n ∈ N},

{^, a, b, c, aa, bb, cc, …, a*n*, b*n*, c*n*, … | n ∈ N}, {a2*k* | k ∈ N}∪{b2*k* + 1a | k ∈ N}.



**Problem from the book (#3):** Find a regular grammar for each of the following languages over the alphabet {a, b}: All strings have length that is a multiple of 3, all strings have an odd number of a’s.

## **Regular Grammars in JFLAP**

You can’t have the shorthand vertical bars in there.

Just enter the grammar rules in the table.

“Test” -> “Test for grammar type” will tell you if it’s a regular grammar or not.

“Input” -> “Brute Parser” will give you a tree for a given string depending on the grammar

“Input” -> “Multiple Brute Force Parse” will let you enter strings to test out the grammar

## **Context-Free Grammar**

A grammar is called a context-free grammar if each production takes the following form

where S is non-terminal and is a non-empty string of terminals and non-terminals. They’re less stricter. However, something important to note:

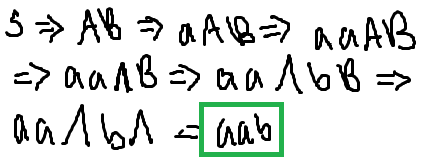
* Every regular grammar is also context-free
* Some context-free grammars are NOT regular grammars

**Other things to know-ish, not learn**

* You can turn a NFA into a DFA
* Every regular expression has a unique minimum-state DFA (minimum # of states, so layout and names are the only differences)

## **More Information on Grammars**

A sentential form is a string made up of terminals and non-terminals. Ex. a^B, a^b^

* If *x* and *y* are sentential forms and is a production rule, then the replacement of by in *x* *y* is called a derivation and you denote it by writing
* More notation:
  + derives in one step
  + derives in 1 or more steps
  + derives in 0 or more steps
* Example: .
  + S aab? NO
  + S aab? YES
  + S aab? YES

A leftmost derivation is a derivation that at each step of the leftmost non-terminal of the sentential form is reduced by some production (i.e. always replace the leftmost non-terminal first). There’s also the rightmost derivation.

If , then L(*G*) = {6, a, aa, aaa, aaaa, …}, while, formally, it’s L(*G*) = {x | x ∈ T\* and S x}, where *x* is a string of terminals.

**Grammar for a Finite Language (All Regular)**

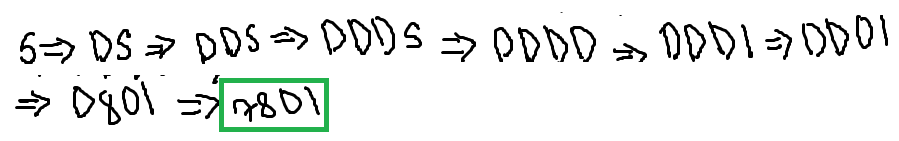
Problem: Suppose L = {a, b, c, abc}. Find a grammar for L.

**Grammar for an Infinite Language**

Theorem: If a language is infinite, there must be a way to loop in your production rules.

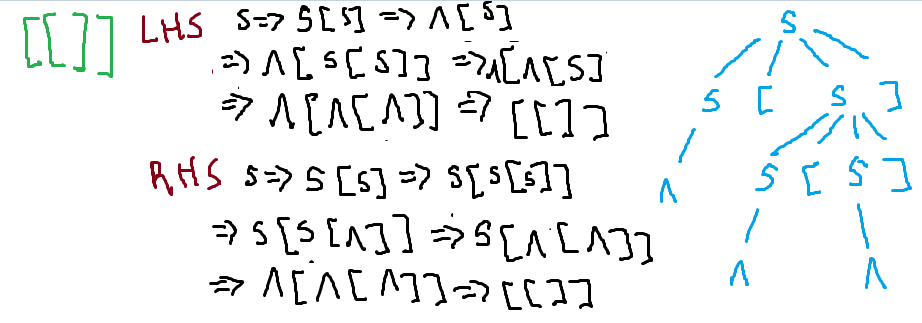
* Think in the back of your head the Pigeonhole principle, where you cannot have 28 pigeon holes and 29 pigeons, because there is guaranteed to have 2 in one.

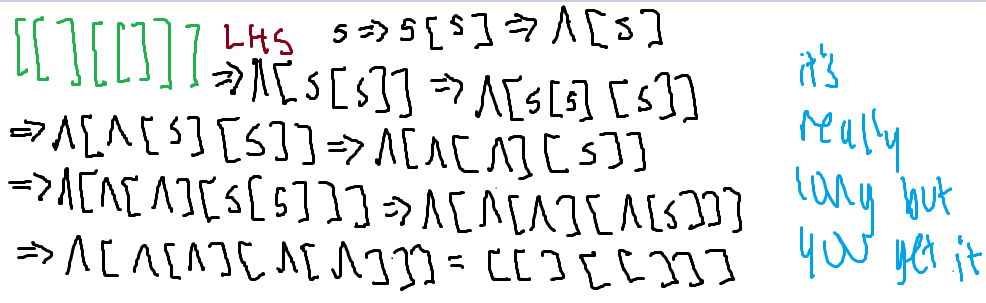
**Problem from the book (pg 199, #1b):** Given the grammar where , find a rightmost derivation of string 7801.



**Problem from the book (pg 199):** Given the grammar where , for each of the following strings, construct a leftmost derivation, a rightmost derivation, and a parse tree:

[[]], [[][[]]].





**Recursive Productions**

A production is recursive if its left side appears on the right side. Ex.

A production is indirectly recursive if A derives a sentential form that contains A. Ex. .

**Recursive Grammars**

A grammar is recursive if it contains a production that is recursive or indirectly recursive.

Are all grammars for infinite languages recursive? YES

## **Building Grammars Practice**

Find a grammar for each of these following languages (pg. 199):

* {a, ba, bba, bbba, …} (regular)

* {ba*n*b | n ∈ N} (regular)

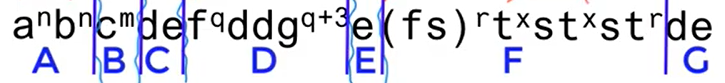
* {a*n*bc*n* | n ∈ N}
* {a2*n* | n ∈ N} (regular)
* {a*m*bc*n* | m, n ∈ N} (regular)
* {a*m*b*n* | m, n ∈ N and n > 0} (regular)

* The odd palindromes over {a, b, c} A palindrome is a string that is symmetrical,

it reads the same letters forwards and back.

Create a grammar for the following language:

Firstly, group the terms by exponents. a & b are together, f & g are together, (fs) & t are together, and t & st are together. Split down everything. Then, label each by a non-terminal letter.



**Combining Grammars: Union**



Problem from book (pg 195, #5d): Find a grammar for

**Combining Grammars: Concatenation**



Find a grammar for

**Combining Grammars: Closure (aka \*)**

Find a grammar for {ab}\*.