



## Grammatical Forms

Recall in the last chapter, context-free grammars have no restrictions on the right-hand side of production rules. All of the following are valid context-free grammar production rules:

S  $\rightarrow$  aSbS  
S  $\rightarrow$  c  
S  $\rightarrow$   $\lambda$   
S  $\rightarrow$  xyz  
S  $\rightarrow$  ABCDEF  
S  $\rightarrow$  MZK

Notice that there can be a  $\lambda$ , one or more terminals, one or more non-terminals and any combination of terminals and non-terminals on the right-hand side of a production rule.

In this chapter, we will be looking at some special grammar forms. These special forms place restrictions on what is allowed on the right-hand side of a production rule. The reason for placing restrictions on the rules is implementation efficiency. The computational complexity of a parser is based on the amount of work it takes to match input with grammar production rules. If the grammar is put into a form that will minimize the amount of backtracking and choices that a program must make, then we can build more efficient parsers/compiler.

Let us first look at a special class of grammars, known as the regular grammar. Regular grammar is capable of specifying rules for regular languages. Regular grammar is a subset of context-free grammar.

### Regular Grammars:

- For every regular language, there is a regular grammar for it.
- A regular grammar is either **left-linear** or **right-linear**.
- Left-linear grammar -  
A  $\rightarrow$  Bx  
A  $\rightarrow$  x  
where B is a non-terminal and x can be one or more terminals
- Right-linear grammar -  
A  $\rightarrow$  xB  
A  $\rightarrow$  x  
where B is a non-terminal and x can be one or more terminals
- For a grammar to be regular, all of the productions in the grammar would have to be **ALL** left-linear or **ALL** right-linear.

### How to construct Regular Grammars:

- From a finite automata or a regular expression
  - Use a new nonterminal for every new character
  - Each loop state turns into a recursive definition on a non-terminal

Example:

R.E. = ab\*ab

Regular Grammar: S  $\rightarrow$  aA

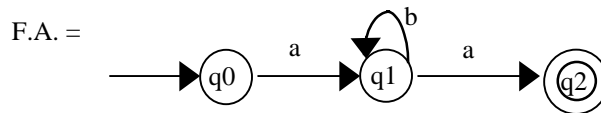
$A \rightarrow bA$   
 $A \rightarrow aB$   
 $B \rightarrow b$

Example:

R.E. =  $a(a+b)^*b$

Regular Grammar:  $S \rightarrow aA$   
 $A \rightarrow aA$   
 $A \rightarrow bA$   
 $A \rightarrow b$

Example:



Regular Grammar =

$S \rightarrow aA$   
 $A \rightarrow bA$   
 $A \rightarrow a$

- From a context-free Grammar that is not regular

Since every regular grammar defines a regular language and every regular language must have a regular expression, it would be easier to go ahead and figure out what the regular expression is and convert it to the regular grammar.

Example:

Context-free Grammar that is not regular:

$S \rightarrow XYX$   
 $X \rightarrow aX \mid bX \mid \lambda$   
 $Y \rightarrow aa \mid bb$

The equivalent regular expression for it is:  $(a+b)^*(aa + bb)(a+b)^*$

Convert it to regular grammar:

$S \rightarrow aS \mid bS \mid X$   
 $X \rightarrow aaY \mid bbY$   
 $Y \rightarrow aY \mid bY \mid \lambda$

### Remove unit and lambda productions:

Unit productions are those productions with only one non-terminal on the right-hand side. Example:  $S \rightarrow A$ . Unit productions simply replace one non-terminal with another. It doesn't process any letters from the input. Unit productions are usually regarded as wasted steps in a parser.

Lambda productions are those productions which derive a  $\lambda$ . Example:  $S \rightarrow \lambda$ . Lambda productions are also troublesome for a parser. They usually introduce choices and require the parser to backtrack.

If a set of production rules contain lambda and/or unit productions, you must remove them before constructing a parser based on the grammar.

The process of removing lambda and unit productions is simple substitution.

Example 1: Remove the lambda production from the following grammar:

```
S -> aAa
S -> bAb
A -> λ
A -> c
```

Our task is to remove the third rule while maintaining the same set of words recognizable by this grammar.

For the first production rule,  $S \rightarrow aAa$ , if the  $A$  on the right-hand side could go to  $\lambda$ , then the production could also be just  $S \rightarrow aa$ . Similarly, the second production could just be  $S \rightarrow bb$ . These two modified rules are **additional** rules, not replacement for the original grammar.

The new grammar becomes:

```
S -> aAa
S -> bAb
S -> aa          <- new rule
S -> bb          <- new rule
A -> c
```

Removing lambda will always result in additional rules.

Example 2: Remove the unit production in the following grammar:

```
S -> a          <- This is NOT a unit production. A single terminal is ok.
S -> aA
S -> B          <- This is an unit production with one non-terminal.
B -> Aa
B -> cCD
B -> Bc
```

To remove the unit production in this case, simply add all of  $B$ 's productions to  $S$ . Since  $B$  has 3 productions and  $S \rightarrow B$ , so replace the  $B$  on the right-hand side with 3 additional rules:

```
S -> Aa
S -> cCD
S -> Bc
```

Again, these are **additional** rules. Leave all non-unit productions alone.

The new grammar:

```
S -> a          B -> Bc
S -> aA          S -> Aa    <- new rule
B -> Aa          S -> cCD   <- new rule
B -> cCD          S -> Bc    <- new rule
```

Example 3: Remove the unit production from the following grammar:

```
S -> Aa
S -> B
B -> A
```

$B \rightarrow bb$   
 $A \rightarrow a$   
 $A \rightarrow bc$   
 $A \rightarrow B$

Notice the circular reference in this grammar. Since  $S \rightarrow B \rightarrow A \rightarrow B$ , substitutions in the previous example won't get rid of all unit productions.

In this case, we want to do couple things:

- 1) Figure out the unit production circular reference:  $S \rightarrow B \rightarrow A \rightarrow B$
- 2) Add all non-unit productions from B to S, from A to B and from B to A.
- 3) Get rid of all unit productions.

The new grammar:

$S \rightarrow Aa$   
 $S \rightarrow bb$   
 $S \rightarrow a$   
 $S \rightarrow bc$   
 $B \rightarrow bb$   
 $B \rightarrow a$   
 $B \rightarrow bc$   
 $A \rightarrow a$   
 $A \rightarrow bc$   
 $A \rightarrow bb$

It's clear to see that after we perform the modification above, B is no longer needed. You don't have to worry about removing useless productions in this case. Just remove the unit productions.

Example 4: If a grammar contains both lambda and unit productions, remove the lambda first.

$S \rightarrow A \mid BAabB$   
 $A \rightarrow abA \mid ab$   
 $B \rightarrow bB \mid \lambda$

Remove lambda first:

$S \rightarrow A \mid BAabB \mid BAab \mid AabB \mid Aab$   
 $A \rightarrow abA \mid ab$   
 $B \rightarrow bB \mid b$

Now remove the unit production  $S \rightarrow A$

$S \rightarrow abA \mid ab \mid BAabB \mid BAab \mid AabB \mid Aab$   
 $A \rightarrow abA \mid ab$   
 $B \rightarrow bB \mid b$