Regular Grammars

Pre-requisite knowledge: deterministic finite automata, non-deterministic finite automata, regular expressions, regular languages, and grammars.

A regular language may be expressed using a deterministic or non-deterministic finite automaton, a regular expression, or a regular grammar. A regular grammar is one that is either right-linear or left-linear.

Def. A grammar G = (V,T,S,P) is said to be <u>right-linear</u> if all productions are of the form:

$A \rightarrow xB,$

$A \rightarrow x$

where $A, B \in V$, $x \in T^*$.

Def. A grammar is said to be <u>left-linear</u> if all productions are of the form:

$A \rightarrow Bx,$

$A \rightarrow x$

Def. A <u>regular grammar</u> is one that is either right-linear or left-linear.

Example: Simple Arithmetic Expression

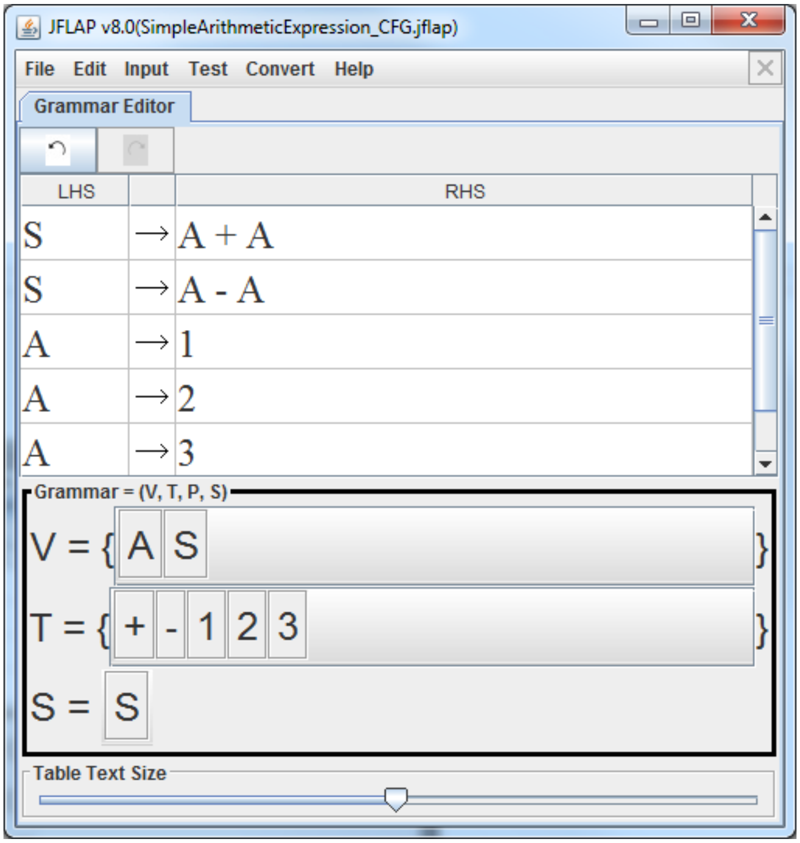
We start by generating the grammar for a simple arithmetic expression where we may use an addition or a subtraction operator (+ or -). The operands may be single digits using 1, 2, or 3. Example arithmetic expressions include 1 + 2 and 2 - 3. Invalid strings include 1 + 2 + 3, 4 + 5, 8 – 1, and 0 + 6.

To develop a regular grammar, we first start with one that *might* be used to represent a simple arithmetic expression:

$$S \rightarrow A + A \mid A - A$$

$$A \rightarrow 1 \mid 2 \mid 3.$$

Try it! Start JFLAP and select File/Open to load the file SimpleArithmeticExpression_CFG.jflap. It should open with this window:



To try an input string of 1 + 2, choose *Input* > *Brute Force Parse* and type 1 + 2 into the *Input* textbox, press the *Enter* key and press *Complete*. You should see *Input Accepted!* You may ignore the brute parse table for now. Check the grammar type using *Test* > *Test for Grammar Type*. JFLAP should report this as a context-free grammar.

Try It! Next, try to input a string which is not part of the language such as 1 – 4 and 5 + 8. What does JFLAP return? How do you know whether or not the input string is part of the language generated by the grammar?

Questions to think about

1. Does the grammar accept 1 + 2 + 3? Why or why not?

   Answer: No, the grammar does not accept 1 + 2 + 3.

2. Is the grammar a regular grammar?

   Answer: To find out, we need to be able to say that the grammar is either right- or left-linear. So we ask ourselves, is it right-linear? No. The first rule (S → A + A) does not satisfy the definition of a right-linear grammar since the right hand side of the rule contains a terminal in between two non-terminals. Is it left-linear? It is not for similar reasons above.

Next, we re-work the grammar into a right-linear grammar for the same language expressed. To eliminate the terminal in between two non-terminals, we need to use a new non-terminal to represent the “+ A” or “- A” portion of the string. We represent this with a new non-terminal, B. The right-linear grammar could be:

$$S \rightarrow 1B \mid 2B \mid 3B$$

$$B \rightarrow +C \mid -C$$

$$C \rightarrow 1 \mid 2 \mid 3.$$

Try It! Enter this grammar containing 8 production rules into JFLAP. Run some valid and invalid input strings. Check its grammar type. Note that all right-linear grammars are also context-free grammars.

Try It! Convert the grammar into a left-linear grammar. Enter the production rules into JFLAP. Run some test strings. Check its grammar type.

Questions to think about

1. How many strings can be generated by this language?

   Answer: 3 x 2 x 3 = 18. There are 18 different strings accepted by the grammar.

2. What is the relationship between context-free grammar and regular grammars? For example, are all context-free grammars right- or left-linear?

   Answer: A context-free grammar is not always right- or left-linear as shown with the first grammar in this module but all right-linear and left-linear grammars are context-free.

3. How would the right-linear grammar be extended to include multiple digits in each operand? Example valid strings include 321123 + 2233 and 111 - 33.

   Answer: Add the following rules to the grammar: $B \rightarrow 1B \mid 2B \mid 3B$ and $C \rightarrow 1C \mid 2C \mid 3C$.

Reference:

Peter Linz, “An Introduction to Formal Languages and Automata” 5th edition, Jones and Bartlett, 2011.