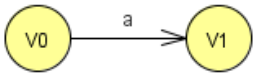
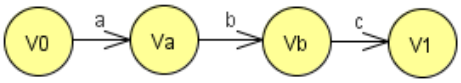
Converting Regular Grammar to DFA

Assume that a regular grammar is given in its right-linear form, this grammar may be easily converted to a DFA. A right-linear grammar, defined by G = (V, T, S, P), may be converted to a DFA, defined by M = (Q, Σ, δ, q_0, F) by:

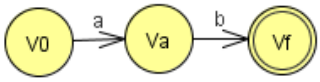
1. Create a state for each variable.
2. Convert each production rule into a transition.
    a. If the production rule is of the form $V_i \rightarrow aV_j$, where $a \in T$, add the transition $\delta(V_i, a) = V_j$ to M. For example, $V_0 \rightarrow aV_1$ becomes:



    b. If the production rule is of the form $V_i \rightarrow wV_j$, where $w \in T^*$, create a series of states which derive w and end in V_j. Add the states in between to Q. For example, $V_0 \rightarrow abcV_1$ becomes:



    c. If the production rule is of the form $V_i \rightarrow w$, where $w \in T^*$, create a series of states which derive w and end in a final state. For example, $V_0 \rightarrow a$ becomes:



To practice this algorithm, we focus on a set of strings which are arithmetic expression involving the addition or subtraction of two binary numbers where each operand can be any binary number of at least one bit and the expression contains exactly one operator. Example valid strings include 01110+110001, 1101011-01110111, and 0-101.

The regular grammar for this language:

$$S \rightarrow 0A \mid 1A,$$

$$A \rightarrow 0A \mid 1A \mid +B \mid -B,$$

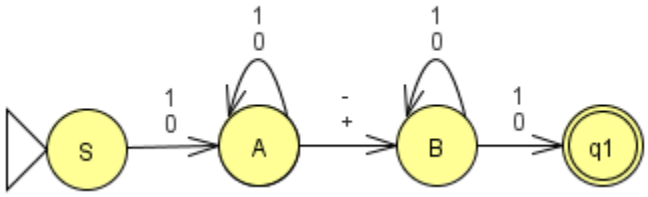
$$B \rightarrow 0B \mid 1B \mid 0 \mid 1.$$

We define the resulting DFA, M = ((Q, Σ, δ, S, F), where δ (right column) is derived from the algorithm above for each of the production rules (left column) of the table below, $C \in F$.

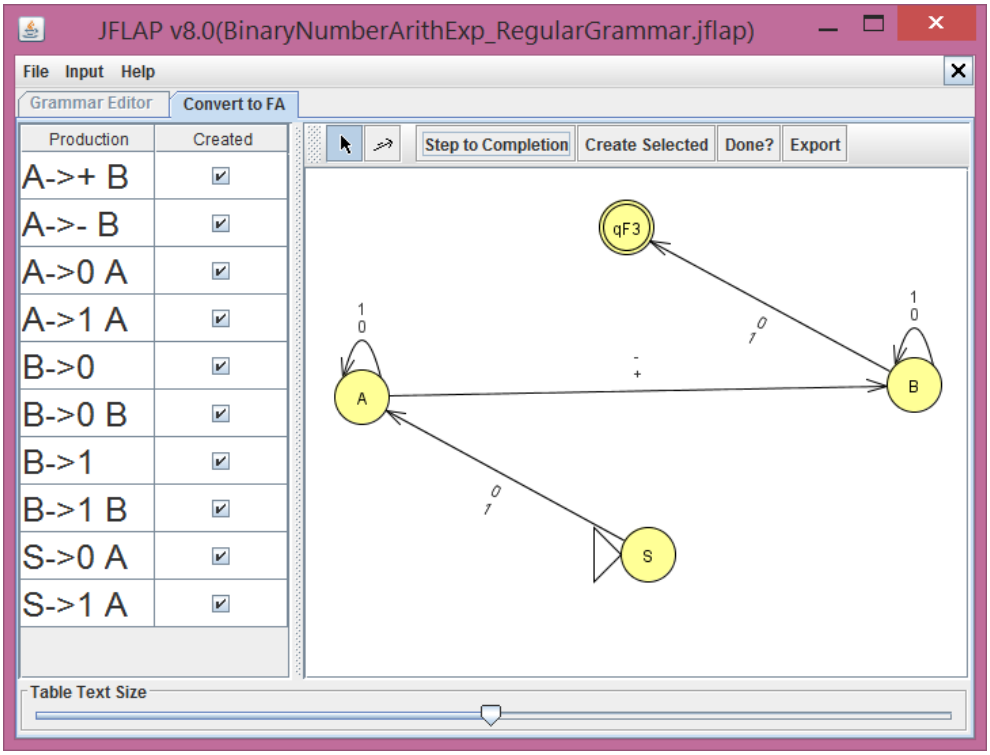
$S \rightarrow 0A$	$\delta(S, 0) = A$
$S \rightarrow 1A$	$\delta(S, 1) = A$
$A \rightarrow 0A$	$\delta(A, 0) = A$
$A \rightarrow 1A$	$\delta(A, 1) = A$
$A \rightarrow +B$	$\delta(A, +) = B$
$A \rightarrow -B$	$\delta(A, -) = B$

B → 0B	δ(B, 0) = B
B → 1B	δ(B, 1) = B
B → 0	δ(B, 0) = C
B → 1	δ(B, 1) = C

The DFA for this language is shown below.



Try It! Start JFLAP and create a grammar with the rules given earlier for this example. Once the productions rules are entered, convert the grammar to a DFA using *Convert > Convert Regular Grammar to FSA*. A new JFLAP screen shows with the grammar on the left-hand pane. Click *Step to Completion* to get a screen similar to this:



Questions to think about

1. Could the resulting DFA for any given regular grammar have more than one final state?

   Answer: Yes. The final states are introduced by converting rules in the $V_i \rightarrow w$ format. If more than one production rule have this format, more than one final state is created.

2. Will the resulting DFA for any given regular grammar have exactly one start state?

   Answer: Yes. A grammar is required to have one start variable and since this start variable becomes the start state for the DFA, there will be exactly one start state.

Reference:

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