Converting NPDA to CFG

Pre-requisite knowledge: non-deterministic pushdown automata and context-free grammars.

Recall that a nondeterministic pushdown accepter (NPDA) is defined by:

M = ($Q, \Sigma, \Gamma, \delta, q_0$, z, F) where

Q – set of states in CU

Σ - input alphabet

Γ - stack alphabet

δ - $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$

- the transition function

q_0 – initial state of the CU, $q_0 \in Q$

z – stack start symbol, $z \in \Gamma^*$

F – set of final states, $F \subseteq Q$

In this module, we will look at how an NPDA is converted to a CFG in JFLAP. We consider the language over the alphabet {a, b} where the number of a's is the same as the number of b's. Sample strings in the language include abab, aabbba, bbabbaaa. One approach to building the NPDA is the following. For every a read, push a 0 into the stack if there is no matching b; for every b read, push a 1 into the stack if there is no matching a. Remaining unmatched letters are recorded in the stack. After parsing the string, the stack should be empty.

One initial approach to this is to use the following transitions over the state q_0:

M = ($Q, \Sigma, \Gamma, \delta, q_0, \lambda$, F) where

$Q = \{q_0, q_f\}$

$\Sigma = \{a,b\}$

$\Gamma = \{0,1\}$

$F = \{ q_f \}$

where δ is defined

$\delta(q_0, \lambda, Z) = \{(q_f, \lambda)\}$

$\delta(q_0, a, \lambda) = \{(q_0, 0Z)\}$

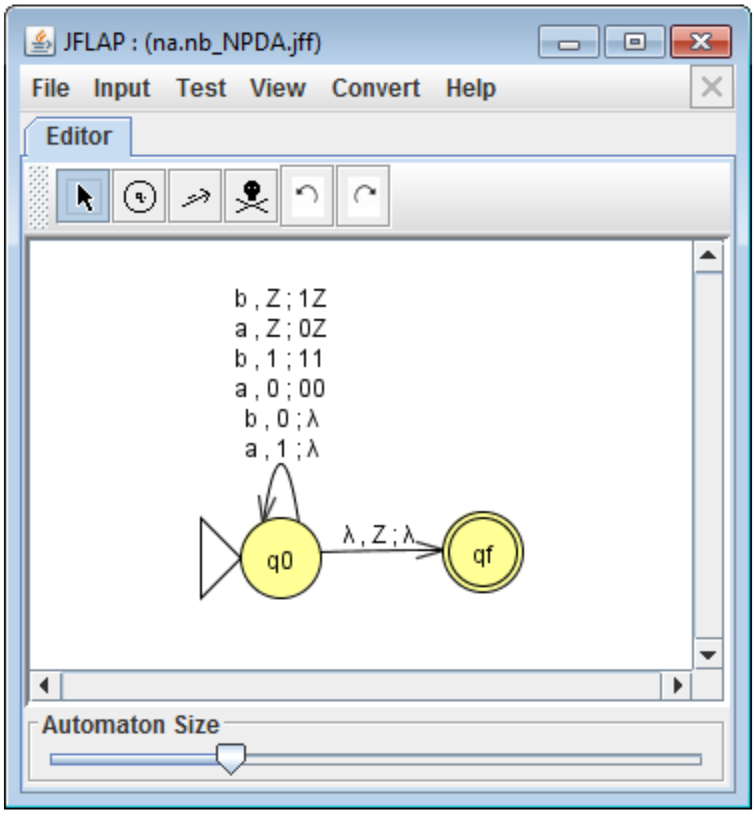
$\delta(q_0, a, 0) = \{(q_0, 00)\}$

$\delta(q_0, a, 1) = \{(q_0, \lambda)\}$

$\delta(q_0, b, \lambda) = \{(q_0, 1Z)\}$

$\delta(q_0, b, 1) = \{(q_0, 11)\}$

$\delta(q_0, b, 0) = \{(q_0, \lambda)\}$

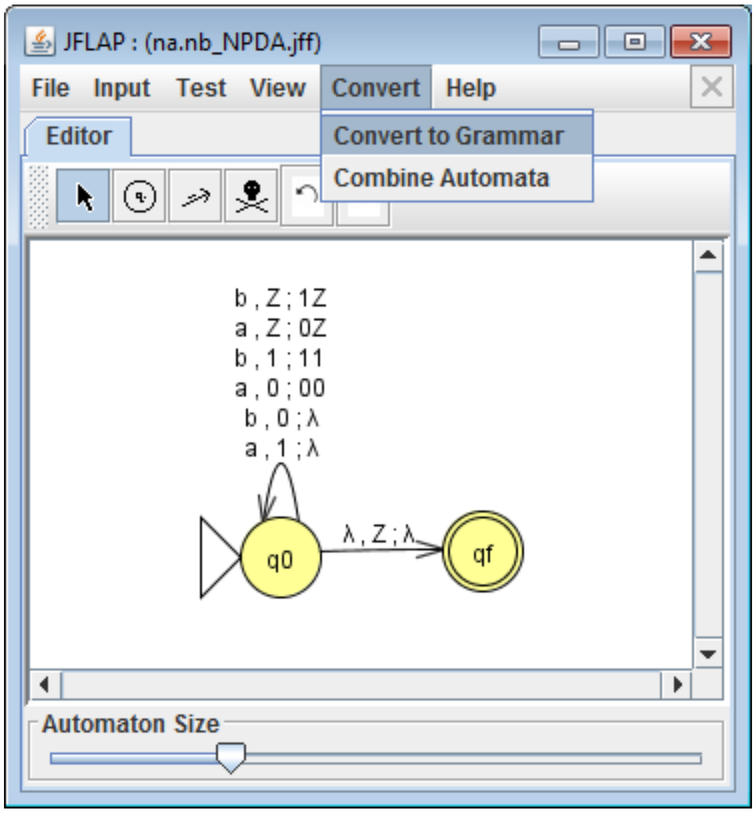


Try It! Open the na.nb_NPDA.jff. Run sample strings including valid and invalid strings such as ba, abba, aabbb, bbabbabb.

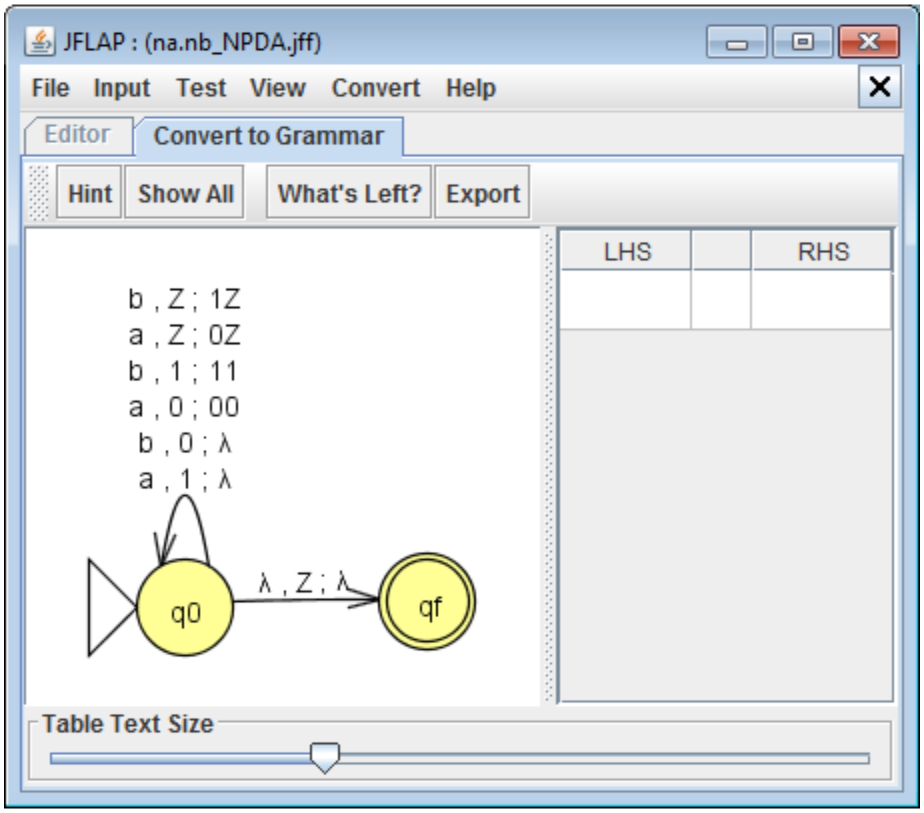
Next, we will convert the NPDA to a CFG. Before we do that, it is important to check that the NPDA adheres to JFLAP's rules for converting to a grammar. These are:

1. All transitions must pop exactly one symbol off the stack and push onto it either none or exactly two symbols.
2. There can be only one final state, and any transition into the final state must pop Z off the stack.

We can examine the NPDA above and see that the NPDA adheres to these rules. Next, select *Convert > Convert to Grammar*.

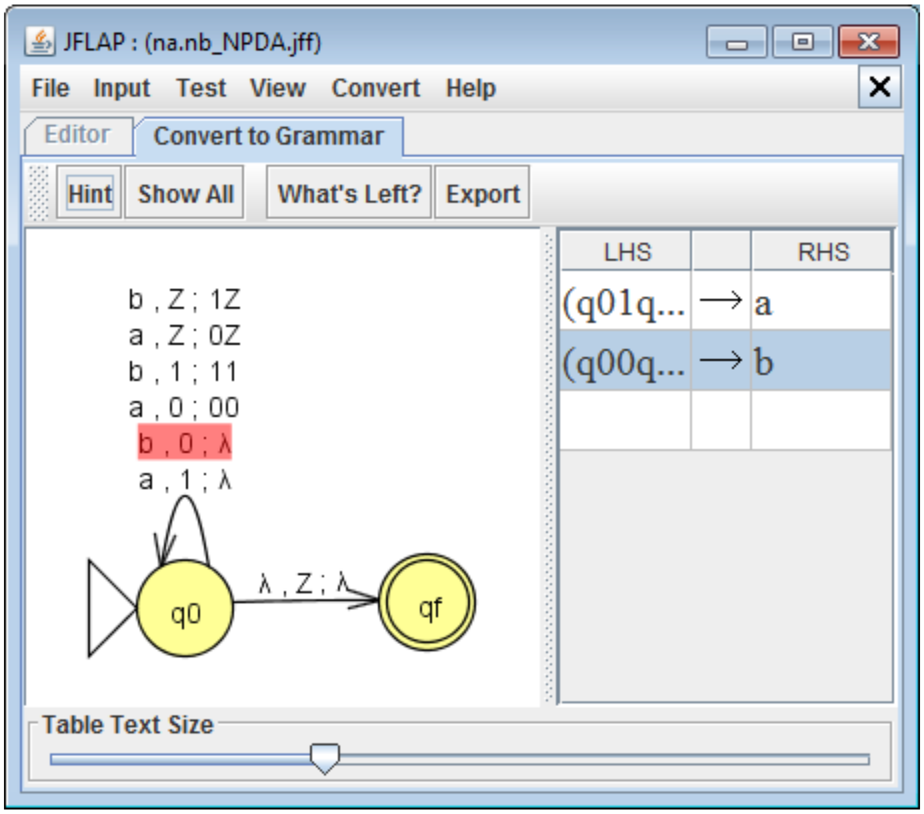


A window shows an extra pane on the right hand side ready to fill with production rules for the grammar.



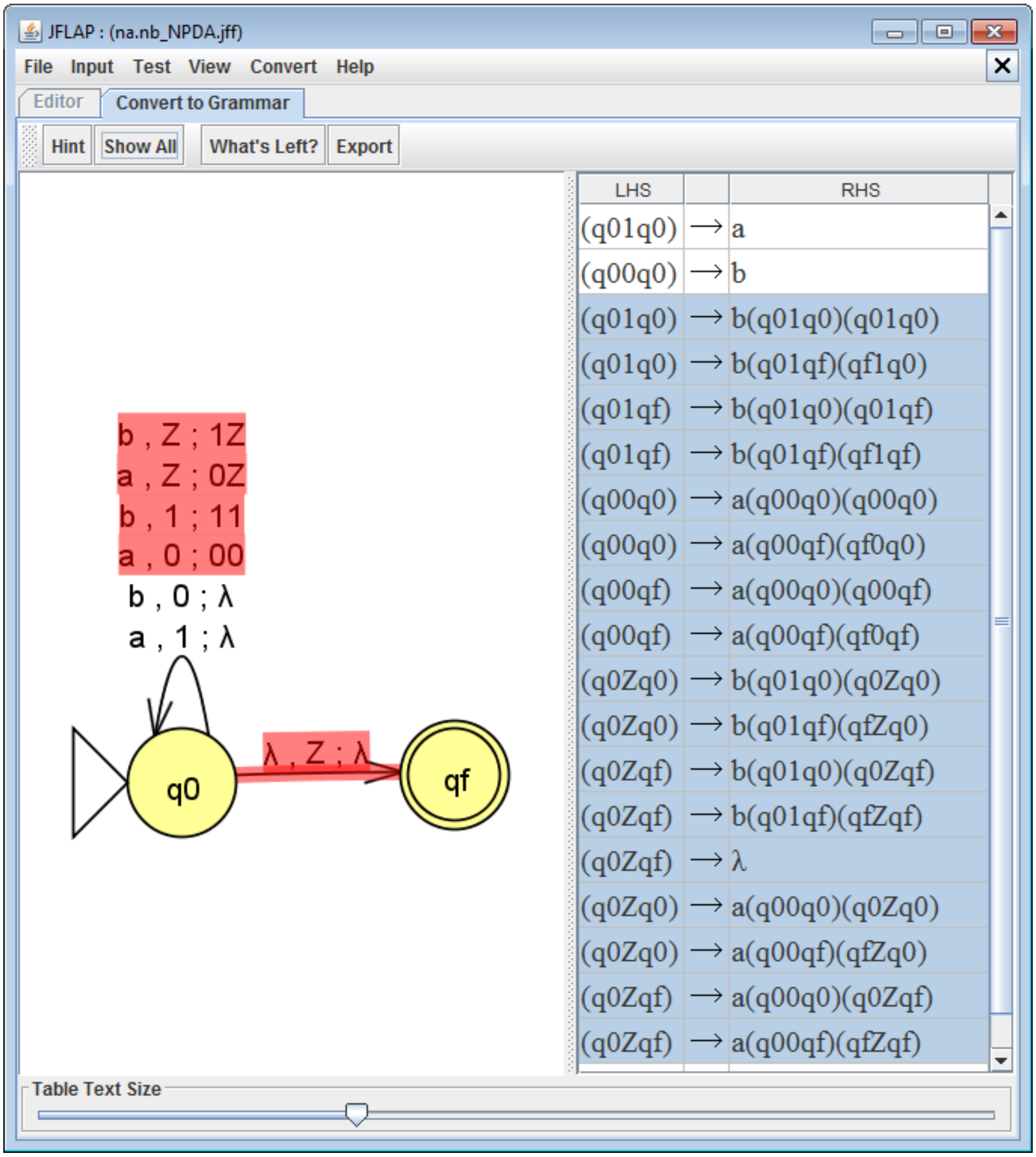
LHS		RHS

Clicking Hint successively will show the progression of how the grammar is built.



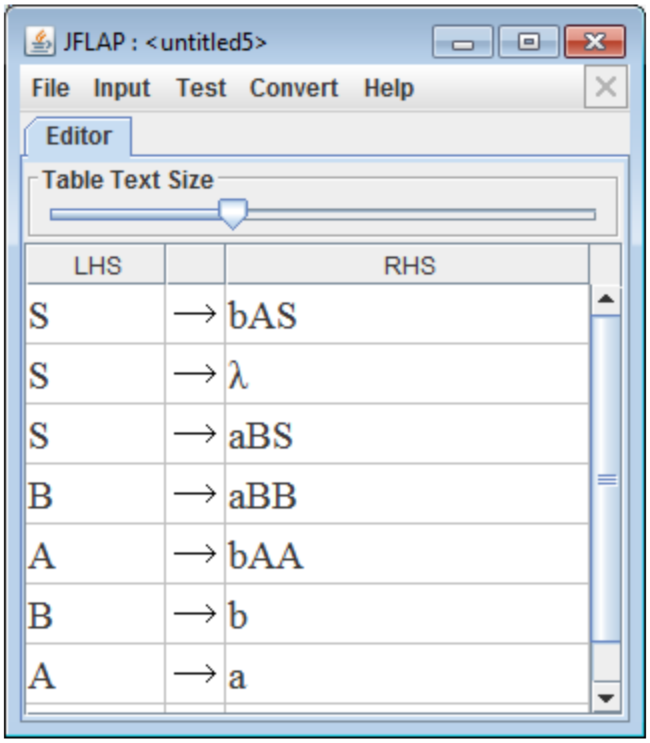
LHS		RHS
(q01q...	→	a
(q00q...	→	b

The JFLAP window above shows the first two NPDA transitions being converted to grammar rules. Click *Hint* until no more hints are there or click *Show All* to complete the transformation.



LHS		RHS
(q01q0)	→	a
(q00q0)	→	b
(q01q0)	→	b(q01q0)(q01q0)
(q01q0)	→	b(q01qf)(qf1q0)
(q01qf)	→	b(q01q0)(q01qf)
(q01qf)	→	b(q01qf)(qf1qf)
(q00q0)	→	a(q00q0)(q00q0)
(q00q0)	→	a(q00qf)(qf0q0)
(q00qf)	→	a(q00q0)(q00qf)
(q00qf)	→	a(q00qf)(qf0qf)
(q0Zq0)	→	b(q01q0)(q0Zq0)
(q0Zq0)	→	b(q01qf)(qfZq0)
(q0Zqf)	→	b(q01q0)(q0Zqf)
(q0Zqf)	→	b(q01qf)(qfZqf)
(q0Zqf)	→	λ
(q0Zq0)	→	a(q00q0)(q0Zq0)
(q0Zq0)	→	a(q00qf)(qfZq0)
(q0Zqf)	→	a(q00q0)(q0Zqf)
(q0Zqf)	→	a(q00qf)(qfZqf)

The LHS variables all contain parentheses (()) to indicate that these are variable names generated by JFLAP. This grammar may now be exported to its own JFLAP work area; do this by clicking on *Export*. A message will pop up notifying you that the rules are trimmed. Next, the trimmed grammar should show up.



LHS		RHS
S	$\longrightarrow$	bAS
S	$\longrightarrow$	λ
S	$\longrightarrow$	aBS
B	$\longrightarrow$	aBB
A	$\longrightarrow$	bAA
B	$\longrightarrow$	b
A	$\longrightarrow$	a

This may now be saved as its own JFLAP file using *File > Save As*.

Questions to think about

1. Verify that the CFG generated produces the same set of strings as the original NPDA.
2. Is the CFG generated as short as possible? Can it be trimmed further?
3. Construct an NPDA for the language below and convert to a CFG.

$$L = \{w : n_a(w) + n_b(w) = n_c(w)\}.$$

Reference:

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