Converting a Context-Free Grammar to a Nondeterministic Pushdown Automaton

Jay Bagga

1 Introduction

By now you are familiar with context-free grammars and nondeterministic pushdown automata. They are equivalent in the sense that both generate the class of context-free languages. In this lesson we study two algorithms that convert a given CFG to an equivalent NPDA. We'll see an example and use JFLAP to practice this conversion.

We'll work with the following CFG:

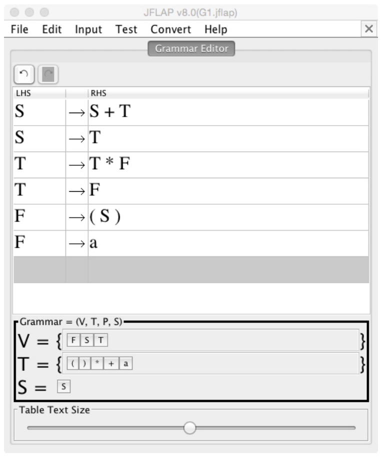
$S \to S + T \mid T$
$T \to T * F \mid F$
$F \to (S) \mid a$

It is not hard to see that this grammar generates arithmetic expressions such as $(a + a) * a$.

The derivation of this expression is as follows:

$S \to T \to T * F \to F * F \to (S) * F \to (S + T) * F \to (T + T) * F \to (F + T) * F \to (a + T) * F \to (a + F) * F \to (a + a) * F \to (a + a) * a$
Below we use JFLAP to implement two algorithms to convert the CFG to an NPDA. These are with the LL parsing and with LR parsing.

2 CFG to NPDA (LL)

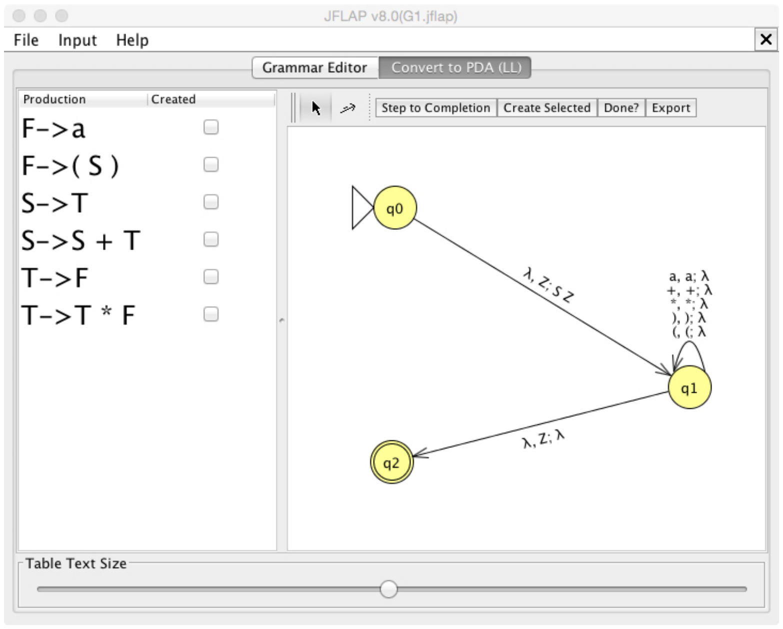


LHS		RHS
S	→	S + T
S	→	T
T	→	T * F
T	→	F
F	→	(S)
F	→	a

Grammar = (V, T, P, S)
V = { F S T }
T = { ( ) * + a }
S = S

Figure 1: Input CFG

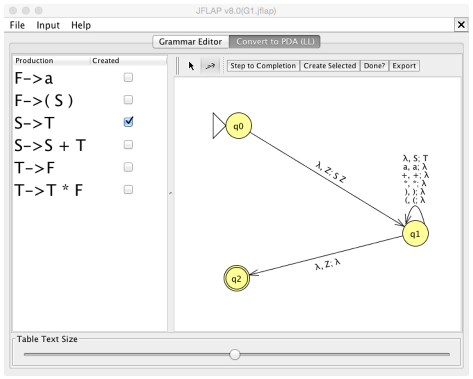
Input the above CFG. See Figure 1. For the LL parsing algorithm, we begin by pushing the start variable onto the stack. When a variable X is popped from the stack, it is replaced by the right side of an X-production. When a terminal is popped, it is matched with the input terminal. The symbol Z is a special symbol used as a "bottom-of-stack" marker.



Production	Created
F−>a	☐
F−>(S)	☐
S−>T	☐
S−>S + T	☐
T−>F	☐
T−>T * F	☐

Figure 2: CFG to NPDA Step 1

Choose Convert to NPDA (LL). See Figure 2. JFLAP creates the NPDA with some initial transitions. The transition from q_0 pushes S. The transitions at q_1 (one for each terminal) match input terminals with those popped. Finally the transition for q_1 to q_2 pops Z and accepts.

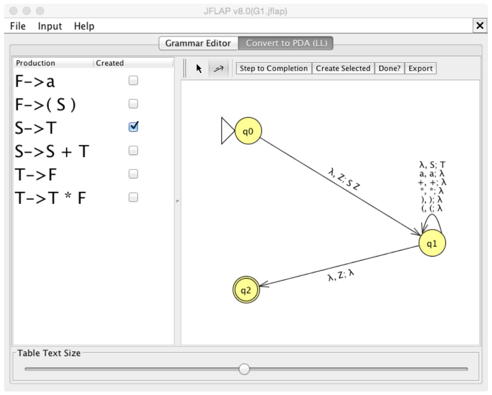


Production	Created
F−>a	☐
F−>(S)	☐
S−>T	✓
S−>S + T	☐
T−>F	☐
T−>T * F	☐

Figure 3: Adding loop transitions

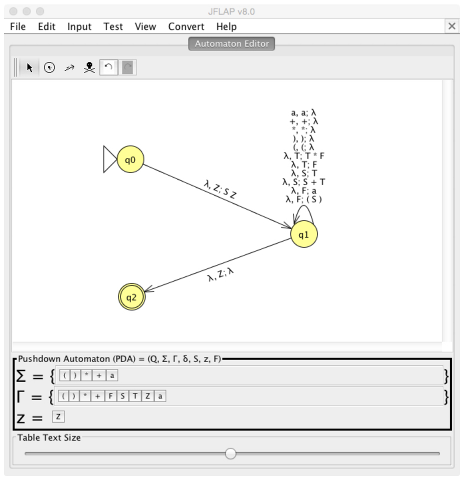
Now we need to add transitions for each production, as described above. All these transitions are loop transitions at q_1. We select q_1 and click on "Create Selected" to create the transition corresponding to the production $S \to T$. See Figure 3.

This process can be repeated for each production. Or you can click “Step to Completion” to get the result as shown in Figure 4. Click “Export” to put this NPDA in a new window from where you can save it. See Figure 5. Simulate this NPDA to find the transitions corresponding to the derivation of the expression $(a + a) * a$ shown above in Section 1.



Production	Created
F->a	☐
F->(S)	☐
S->T	✓
S->S + T	☐
T->F	☐
T->T * F	☐

Figure 4: CFG to NPDA completed



Pushdown Automaton (PDA) = (Q, Σ, Γ, δ, S, z, F)

Σ = { ( ) * + a }

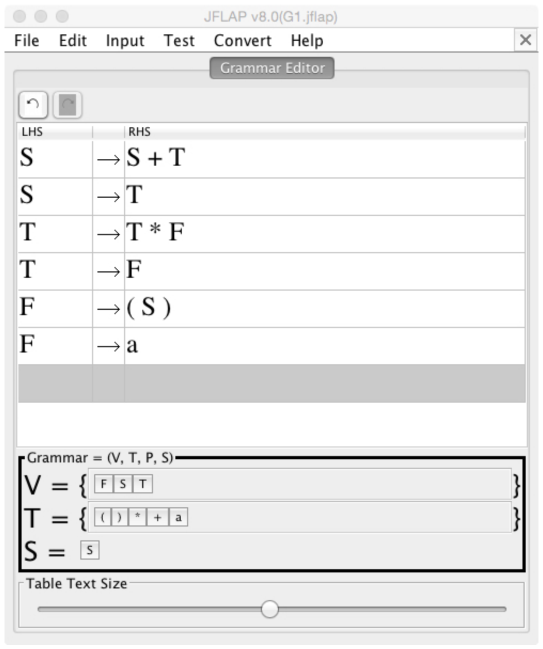
Γ = { ( ) * + F S T Z a }

z = Z

Figure 5: Final NPDA

3 CFG to NPDA (LR)

Input the above CFG again. See Figure 6.



LHS		RHS
S	→	S + T
S	→	T
T	→	T * F
T	→	F
F	→	(S)
F	→	a

Grammar = (V, T, P, S)

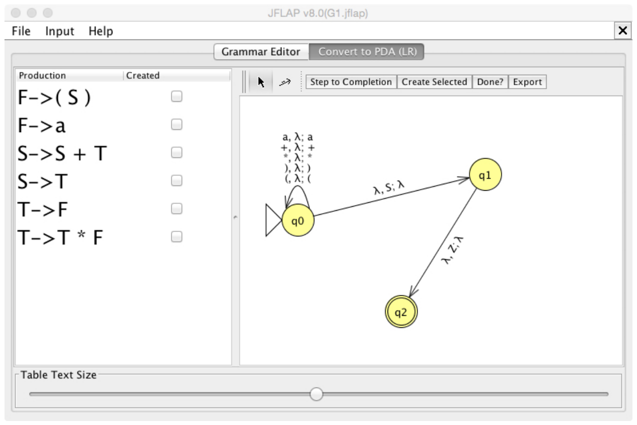
V = { F S T }

T = { ( ) * + a }

S = S

Figure 6: Input CFG

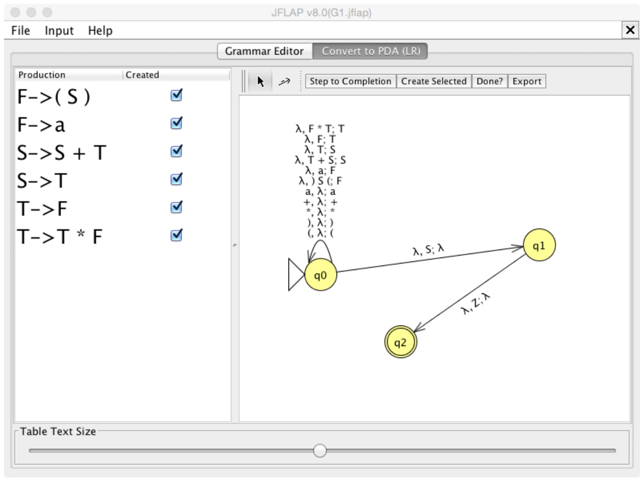
Choose Convert to NPDA (LR). In the first step, when a variable X is popped from the stack, it is replaced by the right side of an X-production. For the LR parsing algorithm, the NPDA will again have three states. In the state q_0, when a terminal is read from the input, it is pushed on the stack. The only transition for q_0 to q_1 pops S and finally we pop Z from q_1 to the accept state q_2. See Figure 7.



Production	Created
F->(S)	☐
F->a	☐
S->S + T	☐
S->T	☐
T->F	☐
T->T * F	☐

Figure 7: CFG to NPDA Step 1

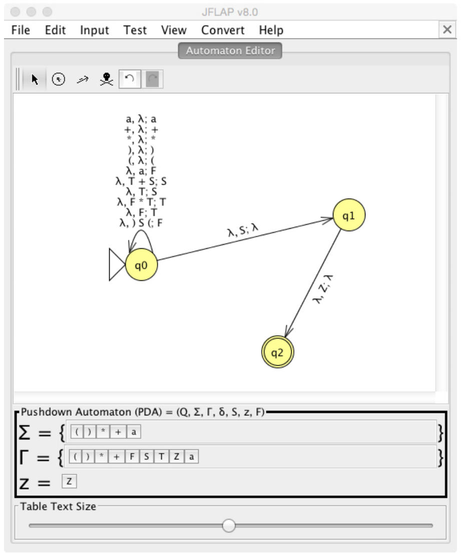
The other part of the work in LR parsing is done as follows: We pop the right hand side of each production and push the left hand side. See the completed NPDA in Figure 8. Match the productions with the corresponding transitions at q_0.



Production	Created
F->(S)	✓
F->a	✓
S->S + T	✓
S->T	✓
T->F	✓
T->T * F	✓

Figure 8: CFG to NPDA completed

See the final NPDA in Figure 9. We test this with the input $(a+a)*a$. Choose "Input" and select "Step by state". After clicking "Step" a few times, you should see a green highlighted step that is accepted in q_2. A trace of this run is shown in the table below. Study this carefully.



Pushdown Automaton (PDA) = (Q, Σ, Γ, δ, S, z, F)

Σ = { ( ) * + a }

Γ = { ( ) * + F S T Z a }

z = Z

Figure 9: Final NPDA

Input read	Input remaining	From state	To state	Pop	Push	Stack
	$(a+a)*a$	q_0	q_0	λ	Z	Z
$($	$a+a)*a$	q_0	q_0	λ	$($	$(Z$
$(a$	$+a)*a$	q_0	q_0	λ	a	$a(Z$
$(a$	$+a)*a$	q_0	q_0	a	F	$F(Z$
$(a$	$+a)*a$	q_0	q_0	F	T	$T(Z$
$(a$	$+a)*a$	q_0	q_0	T	S	$S(Z$
$(a+$	$a)*a$	q_0	q_0	λ	$+$	$+S(Z$
$(a+a$	$)*a$	q_0	q_0	λ	a	$a+S(Z$
$(a+a$	$)*a$	q_0	q_0	a	F	$F+S(Z$
$(a+a$	$)*a$	q_0	q_0	F	T	$T+S(Z$
$(a+a$	$)*a$	q_0	q_0	$T+S$	S	$S(Z$
$(a+a)$	$*a$	q_0	q_0	λ	$)$	$)S(Z$
$(a+a)$	$*a$	q_0	q_0	$)S)$	F	FZ
$(a+a)$	$*a$	q_0	q_0	F	T	TZ
$(a+a)*$	a	q_0	q_0	λ	$*$	$*TZ$
$(a+a)*a$		q_0	q_0	λ	a	$a*TZ$
$(a+a)*a$		q_0	q_0	a	F	$F*TZ$
$(a+a)*a$		q_0	q_0	$F*T$	T	TZ
$(a+a)*a$		q_0	q_0	T	S	SZ
$(a+a)*a$		q_0	q_1	S	λ	Z
$(a+a)*a$		q_1	q_2	Z	λ	λ
ACCEPT						

4 References

1. Introduction to the Theory of Computation (Third Edition), Michael Sipser. Cengage Learning. 2013.

2. JFLAP - An Interactive Formal Languages and Automata Package, Susan H. Rodger and Thomas W Finley. Jones and Bartlett Publishers. 2006