JFLAP -Java Formal Language and Automata Package

Introduction - Working with Grammars

2009-2010

Formal Languages and Automata Theory
http://www.jflap.org/tutorial/

JFLAP

(Java Formal Language and Automata Package)

JFLAP is software for experimenting with formal languages topics including nondeterministic finite automata, nondeterministic pushdown automata, multi-tape Turing machines, several types of grammars, parsing, Mealy and Moore sequential machines, and L-systems. In addition to constructing and testing examples for these, JFLAP allows one to experiment with construction proofs from one form to another, such as converting an NFA to a DFA to a minimal state DFA to a regular expression or regular grammar.

Languages	Operations

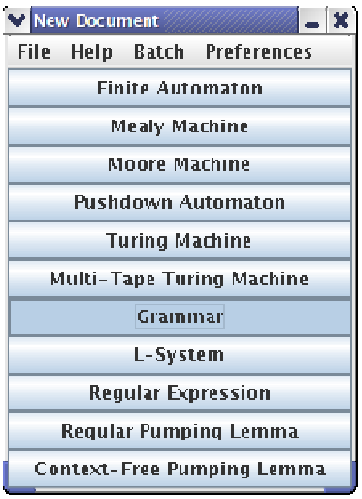
Regular Languages	• DFA.
	• NFA.
	• Regular Grammars.
	• Regular Expressions.
	• Conversions:
	  ○ NFA → DFA → Minimal DFA.
	  ○ NFA <-> Regular Expression.
	  ○ NFA <-> Regular Grammar.

Context-free Languages	• Push-Down Automaton.
	• Context-Free Grammar.
	• Transform:
	  ○ PDA → CFG
	  ○ CFG→ PDA
	  ○ CFG → CNF

Recursively Enumerable languages	• Turing Machine (1-tape). • Turing Machine (multi-tape). • Turing Machine (building blocks). • Unrestricted Grammar.

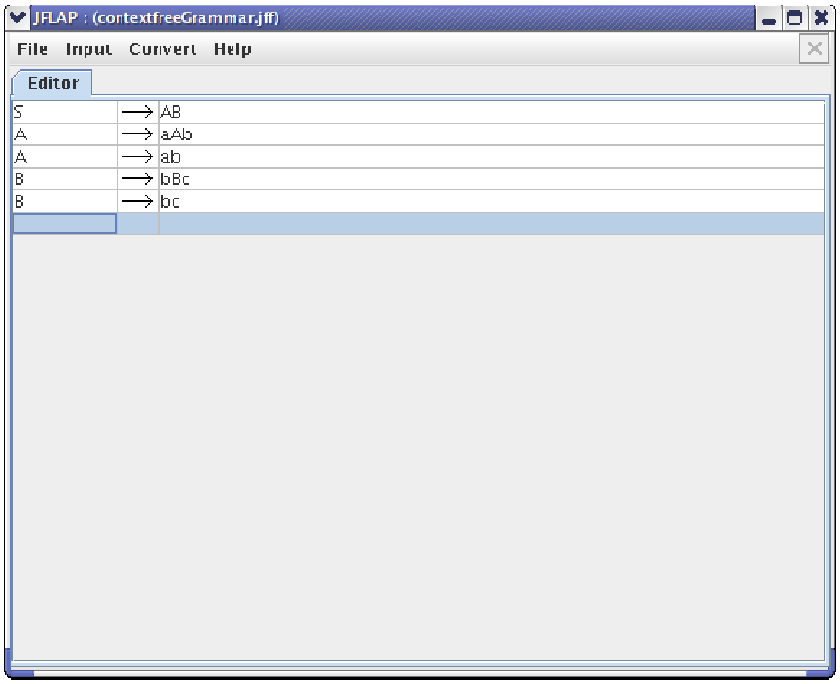
1. How to Enter a Grammar: Grammar Editor

We will begin by entering a context-free grammar. To start a new grammar, start JFLAP (download it from www.jflap.org or Aula Global 2). Run it using *java –jar JFLAP.jar* in Linux or Windows. It requires at least Java SE 1.4. Using the initial screen, click the ***Grammar*** option from the menu, as shown below:



You should be able to see a grammar window that looks like the screen below. You can enter your start variable and its production on the first row. The variable should be entered on the leftmost column and the right side of the production on the rightmost column. The middle column is going to be an arrow indicating the production. You can enter the grammar as shown below.

When you are finished, your grammar editor window should look like this.



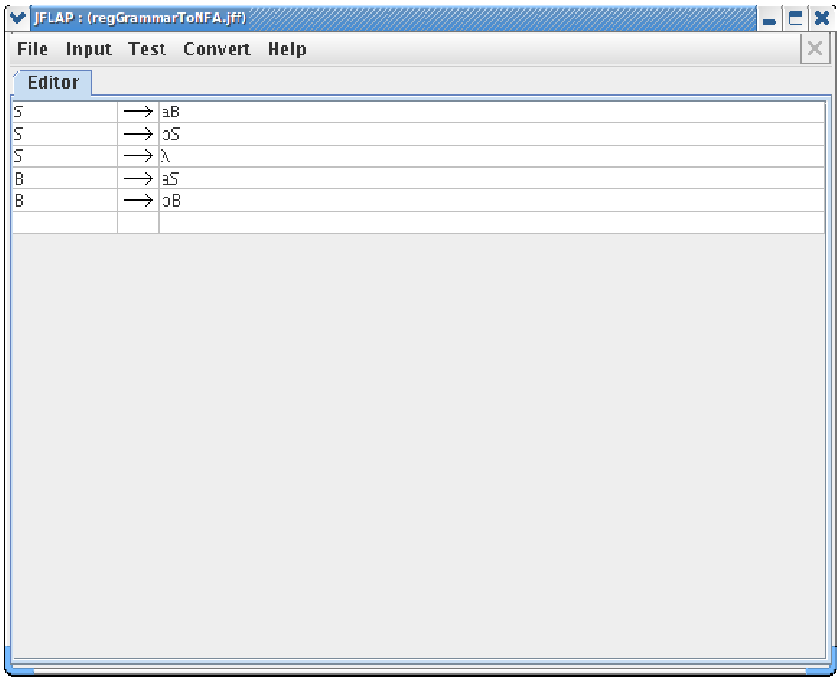
Using the File option, you can save, print and download grammars in XML format. Remember than lowercase is used to express terminal symbols and uppercase for nonterminal symbols. The

axiom is always the nonterminal introduced on the left side of the first production rule. Backus notation is not accepted. Do not enter blanks in the grammar.

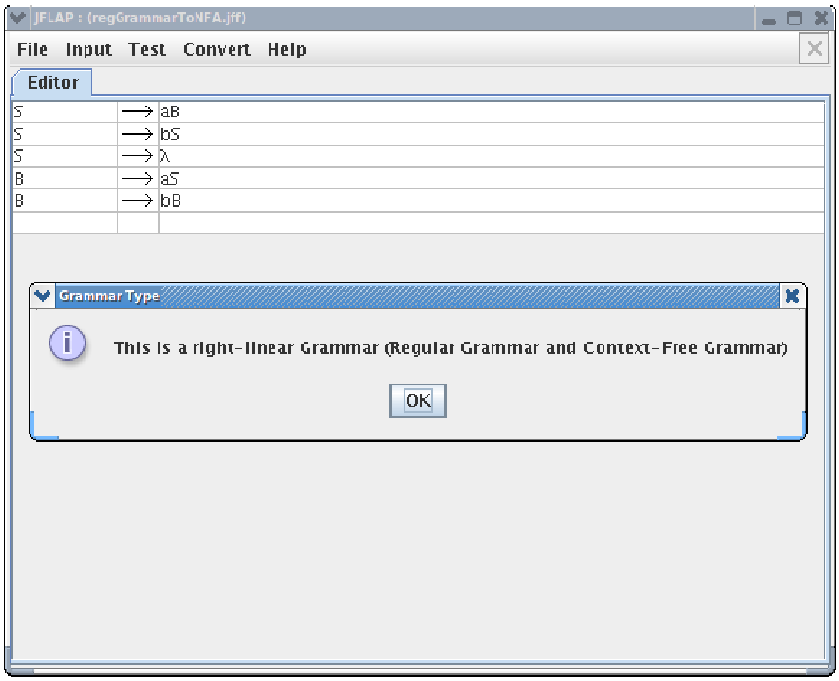
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><!--Created with JFLAP 6.0.--><structure>&#13;
  <type>grammar</type>&#13;
  <!--The list of productions.-->&#13;
  <production>&#13;
    <left>S</left>&#13;
    <right>aA</right>&#13;
  </production>&#13;
  <production>&#13;
    <left>S</left>&#13;
    <right>bA</right>&#13;
  </production>&#13;
  <production>&#13;
    <left>S</left>&#13;
    <right>aC</right>&#13;
  </production>&#13;
  <production>&#13;
    <left>A</left>&#13;
    <right>B</right>&#13;
  </production>&#13;
  <production>&#13;
    <left>B</left>&#13;
    <right>qvC</right>&#13;
  </production>&#13;
  <production>&#13;
    <left>C</left>&#13;
    <right>x</right>&#13;
  </production>&#13;
  <production>&#13;
    <left>B</left>&#13;
    <right/>&#13;
  </production>&#13;
  <production>&#13;
    <left>A</left>&#13;
    <right/>&#13;
  </production>&#13;
</structure>
```

2. *Test for Grammar Type*

JFLAP allows users to check what type of grammar they are entering. We will begin by entering a regular grammar.

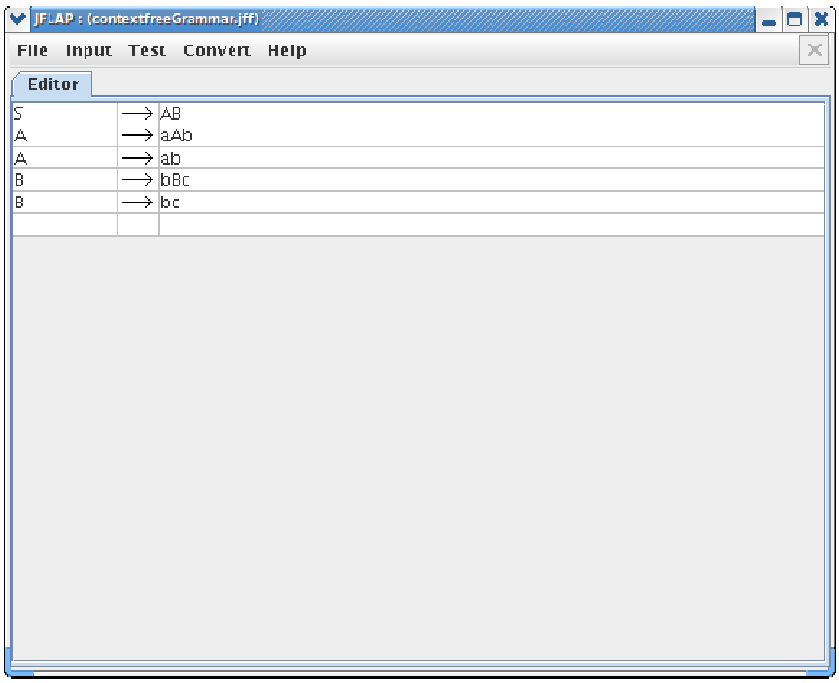


Now let's test for what type of grammar it is. Click on ***Test*** and click ***Test for Grammar Type***. JFLAP will now notify you about the grammar type.

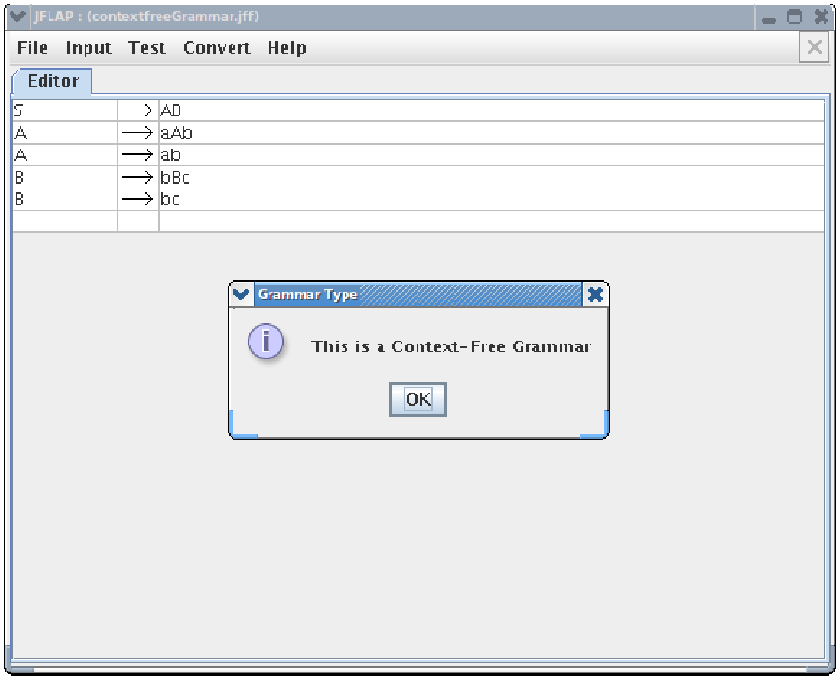


It tells user that the grammar we entered is right-linear grammar, which is also regular and context-free grammar.

Now, let's enter another grammar. Enter the grammar below.



By clicking same options, we can confirm that this grammar is context-free grammar from JFLAP.



This feature in JFLAP can recognize right and left linear grammar, CNF grammar, GNF grammar, context-sensitive grammar, and unrestricted grammar.

3. Verifying if an input is accepted or not: Brute Force Parser

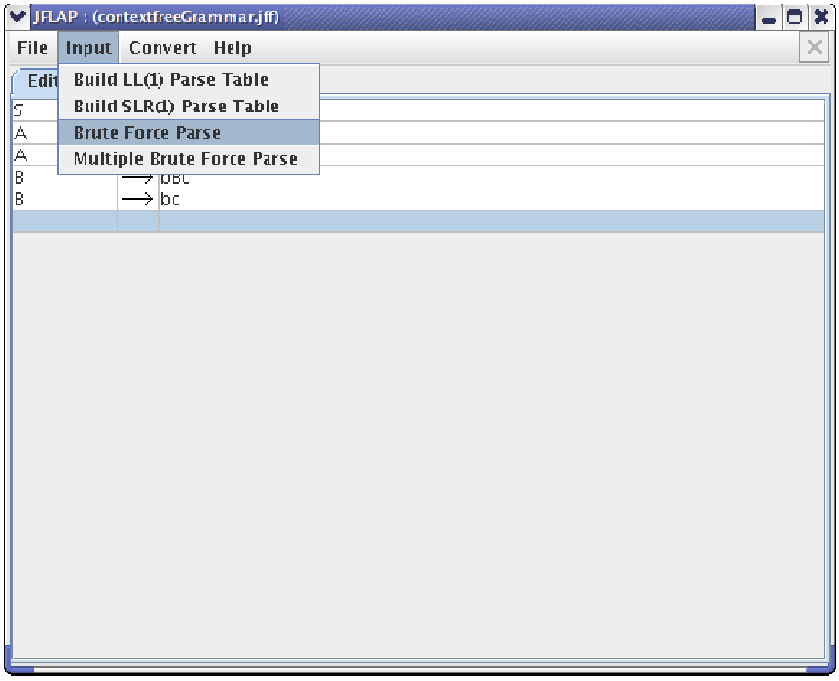
3.1. Regular or Context Free Grammar

JFLAP defines a context free grammar G = (V, T, S, P), where:

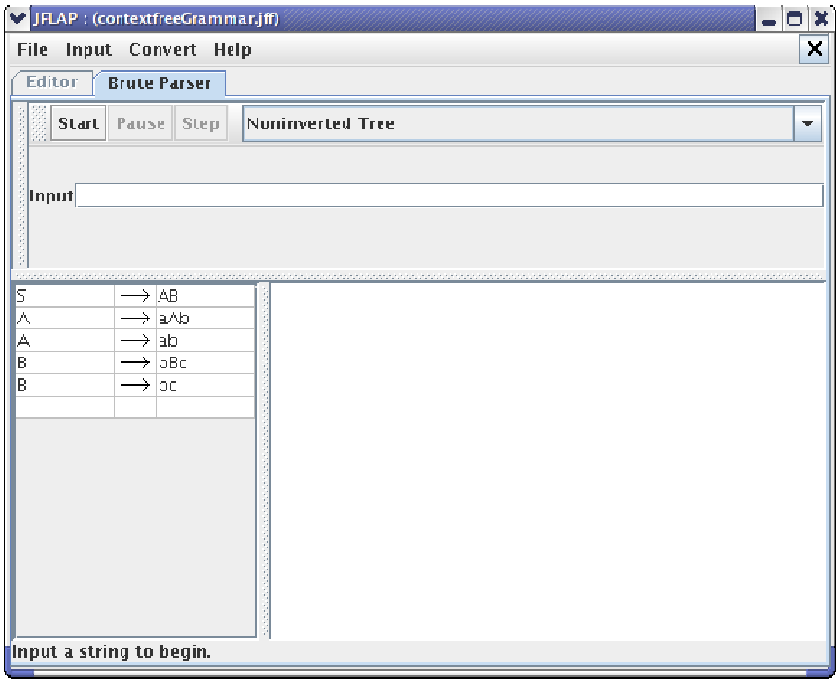
- *V* is a set of variables (S represents the start variable).
- T is a set of terminals.
- P is a set of productions.
- Productions are in the form A→x, where A is a single variable and x is a string of zero or more terminals and variables.

Since JFLAP accepts context-free grammars (CFG), it automatically works with regular grammars (thus, right-linear grammar) as they are also context-free grammars. In a right-linear grammar all productions must have at most one variable in the right-hand side and this variable must be to the right of all the terminals.

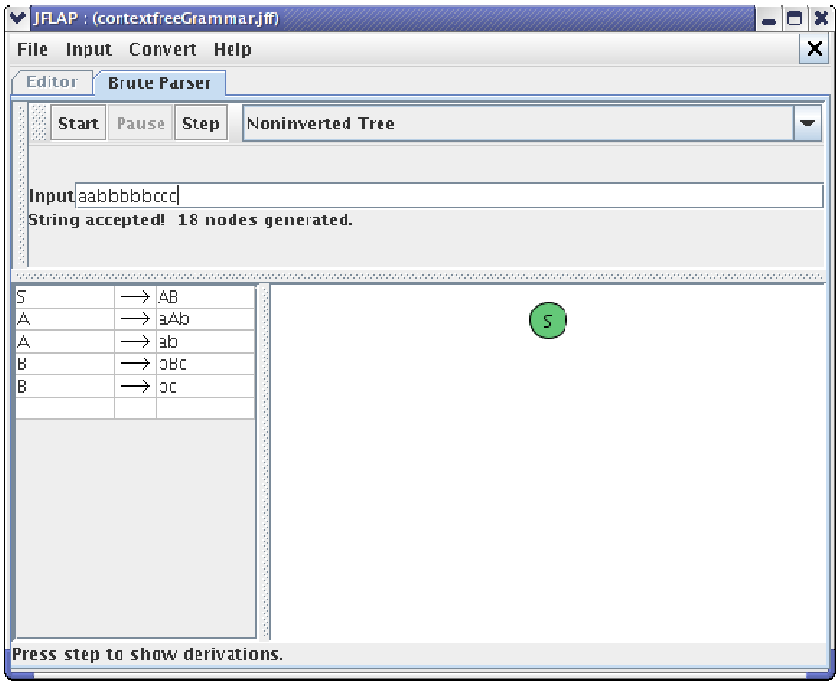
We will begin by loading or entering the grammar that we entered in previous section. Then you can click on ***Input*****,** followed by ***Brute Force Parse***.



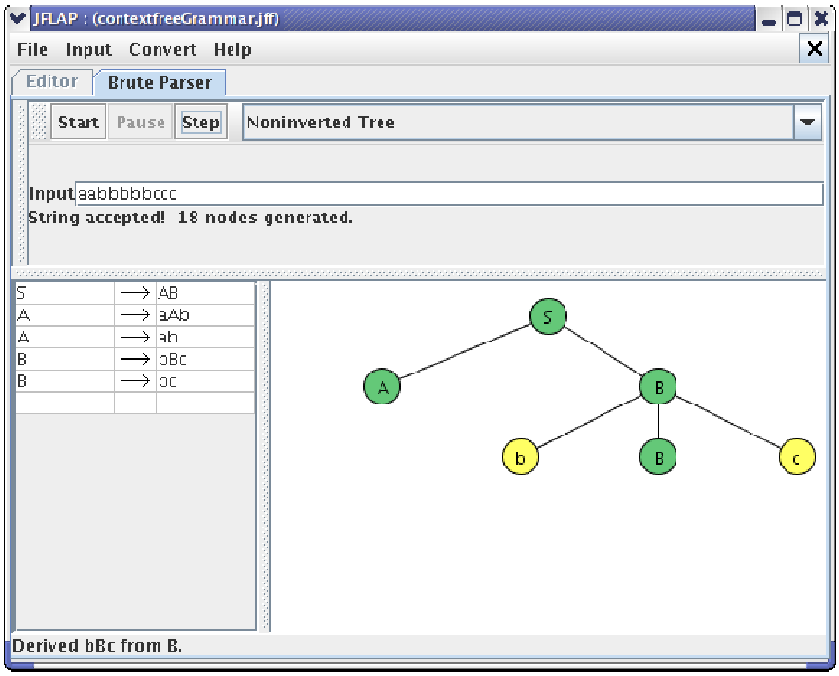
After clicking ***Brute Force Parse***, you should see a window similar to this.



Now you can input a string and check whether it is accepted. Enter the string "aabbbbbccc" ($a^2b^5c^3$) in the input text field. Then either click ***Start*** or press ***Enter key.***

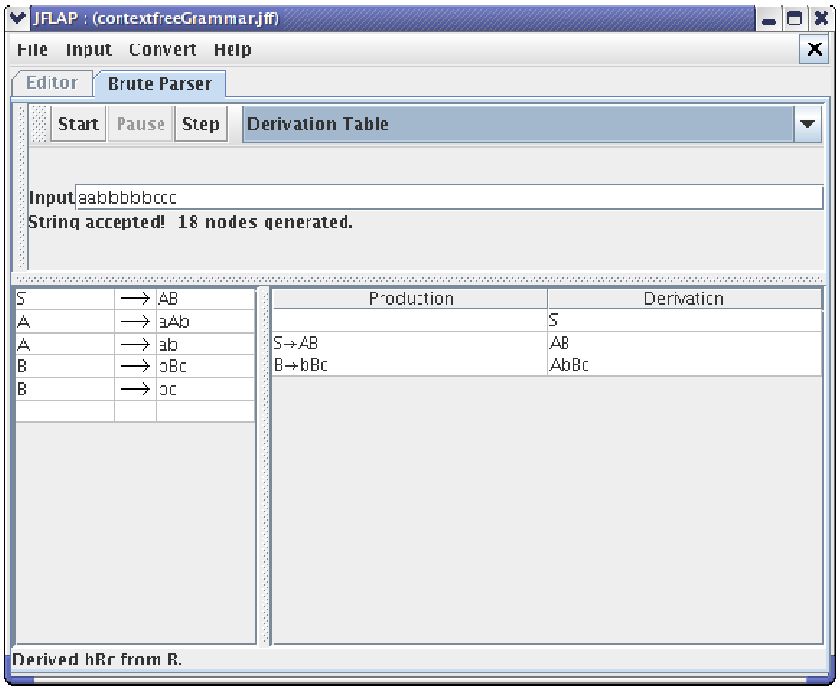


Brute Force Parser notifies you that the string you entered is accepted. Now, we can click on the ***Step*** button to see how we can derive that string using our productions. After clicking the ***Step*** button twice, your window should look like this.

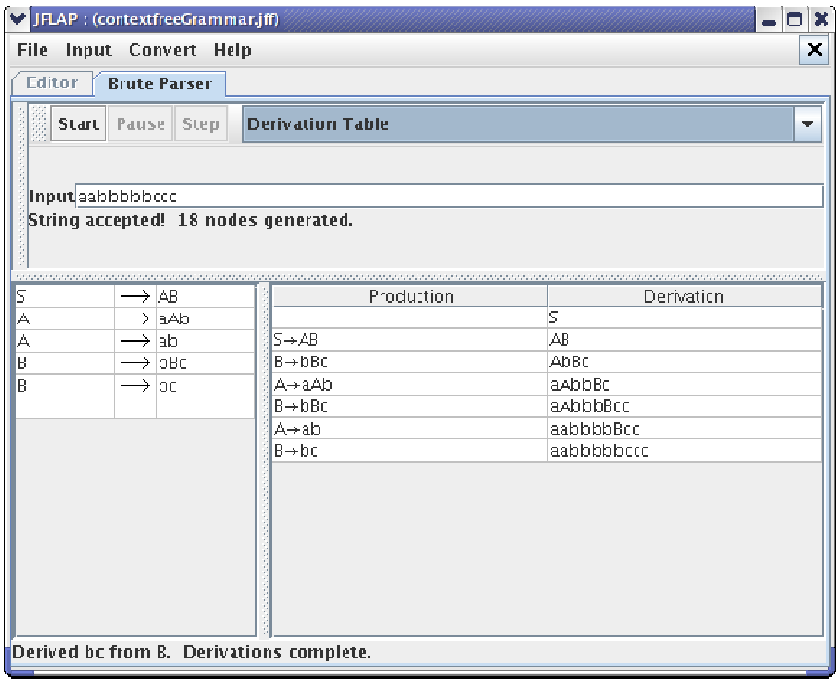


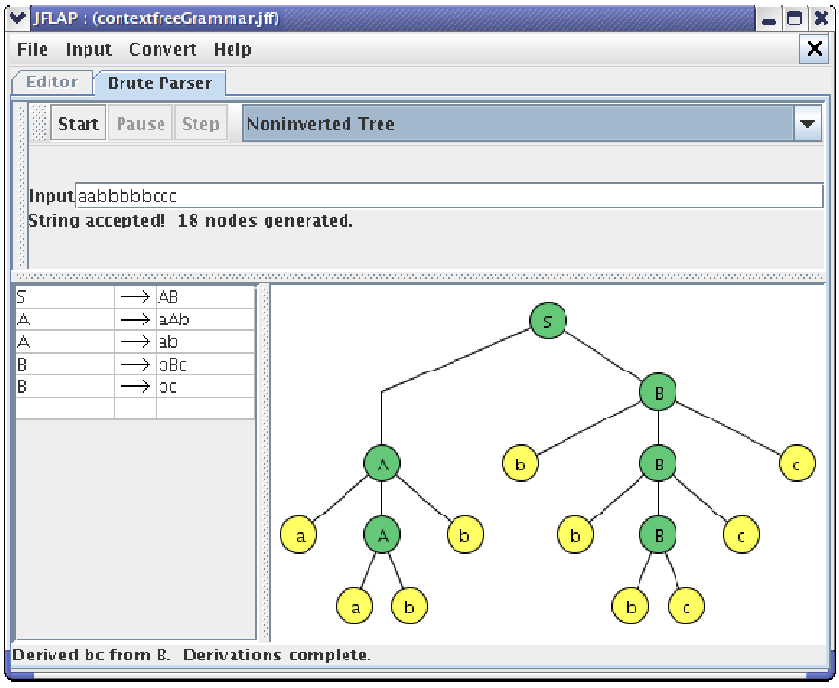
Notice that at the bottom of the window there is a message, “Derived bBc from B”. This message tells us about the last production that was applied. We can also view the ***Derivation Table*** to see how the string is derived by the grammar.

To change to ***Derivation Table*** mode, click on ***Noninverted Tree*** and select ***Derivation Table***. The Brute Force Parse window should change to:



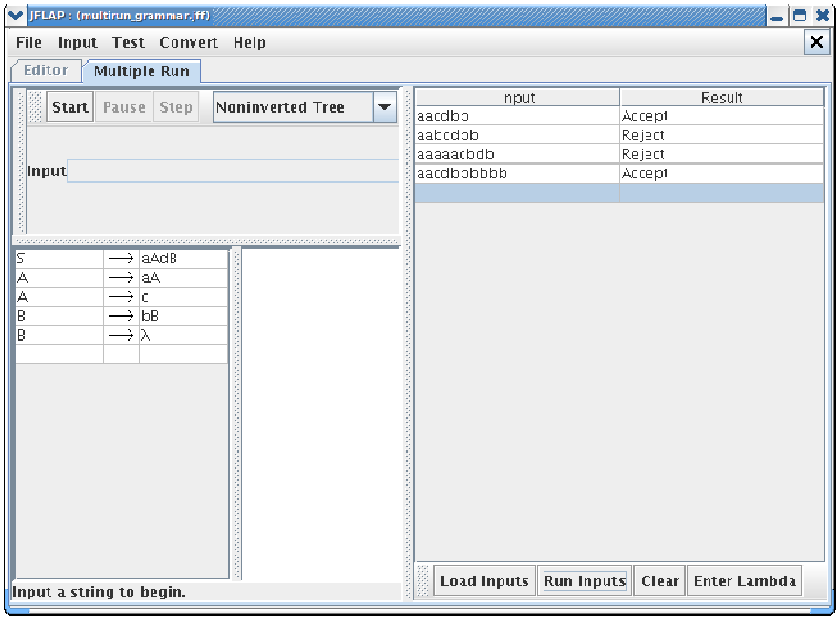
The derivation table shows the productions that are applied (shown under Production) and the result of applying those productions to the start variable (shown under Derivation). It gives a clear picture of how Brute Force Parsing is working. After clicking ***Step*** until we reach our input string, the Brute Force Parse is finished and the ***Derivation Table*** and ***Noninverted Tree*** should look like these:



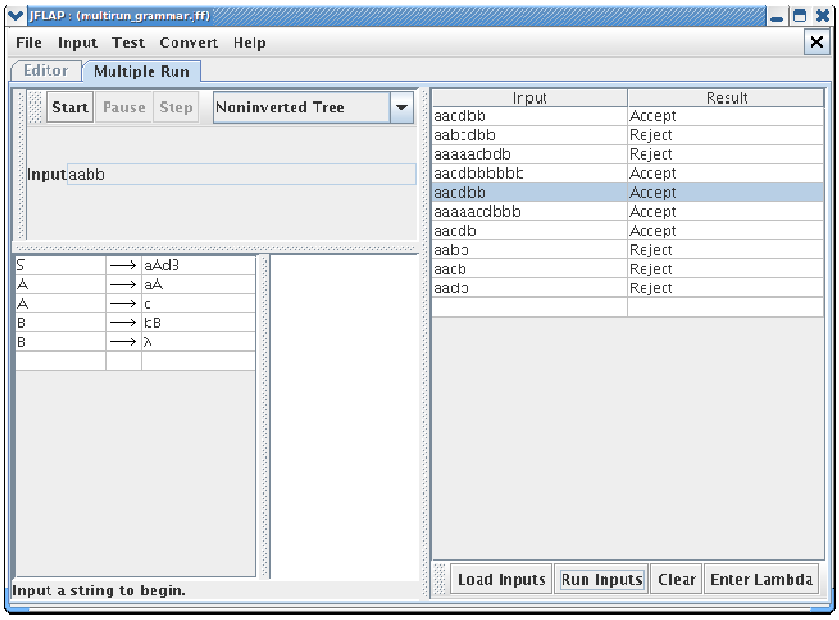


3.2. Multi-Input Run Grammar

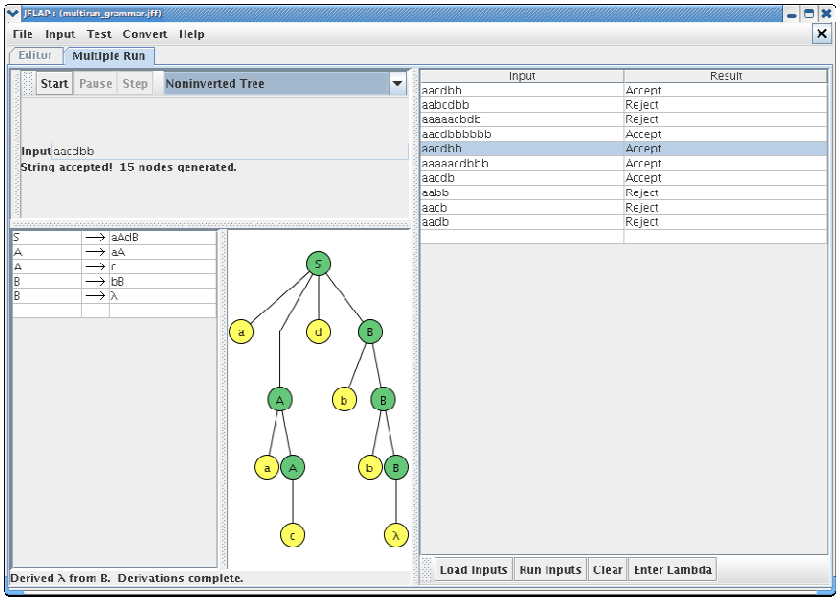
Enter grammar below. To test a grammar on more than one string at once select Multiple Brute Force Parse from the Input menu. On the right is a table with an input column and a result column. To add a test string click in the input column, type the string, and press enter. Selecting run inputs attempts to parse every string in the input column with the grammar.



You can write a list of words (***Menu > Input > Multiple Brute Force Parser***) and then verify (***Run Inputs***) which are accepted or not.



When it is done if you want to see the parse tree for a particular string, select the string by clicking on the row it is in, and click the Start button on the left. If it is accepted you can press Step to see each part of the parse tree.

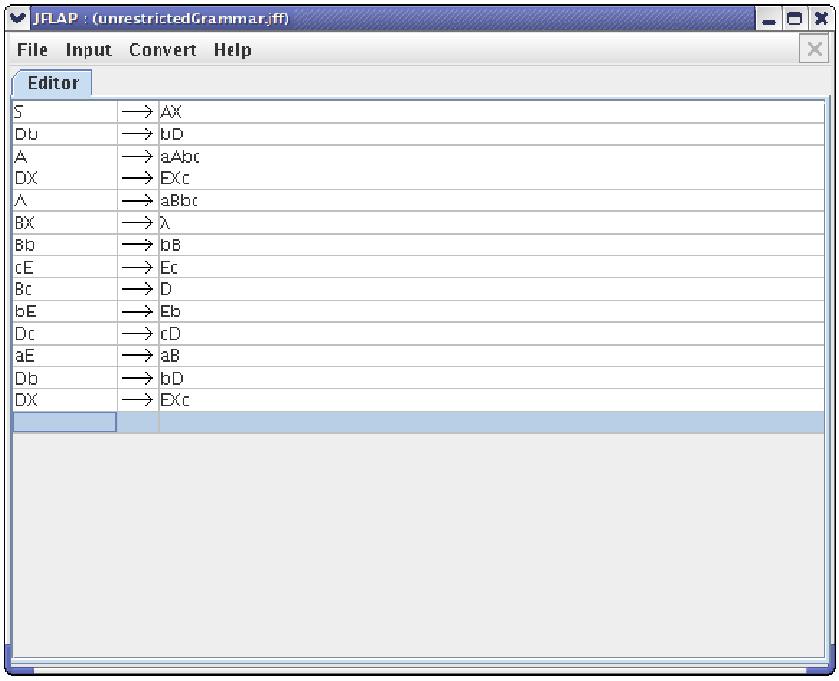


3.3. Unrestricted and Context-Sensitive grammars

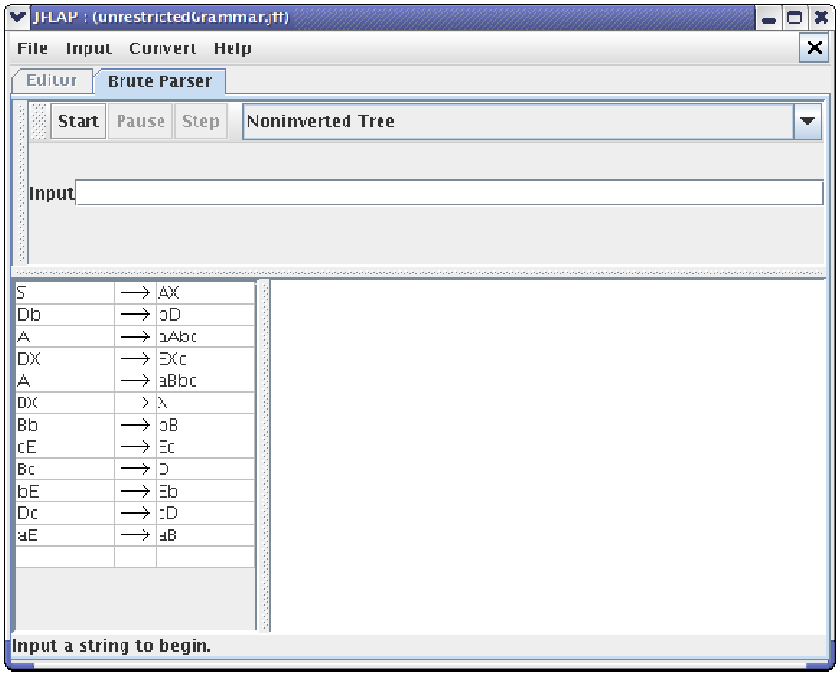
An unrestricted grammar is similar to a context-free grammar (CFG), except that the left side of a production may contain any nonempty string of terminals and variables, rather than just a single variable.

In a CFG, the single variable on the left side of a production could be expanded in a derivation step. In an unrestricted grammar, multiple variables and/or terminals that match the left-side of a production can be expanded by the right-side of that production.

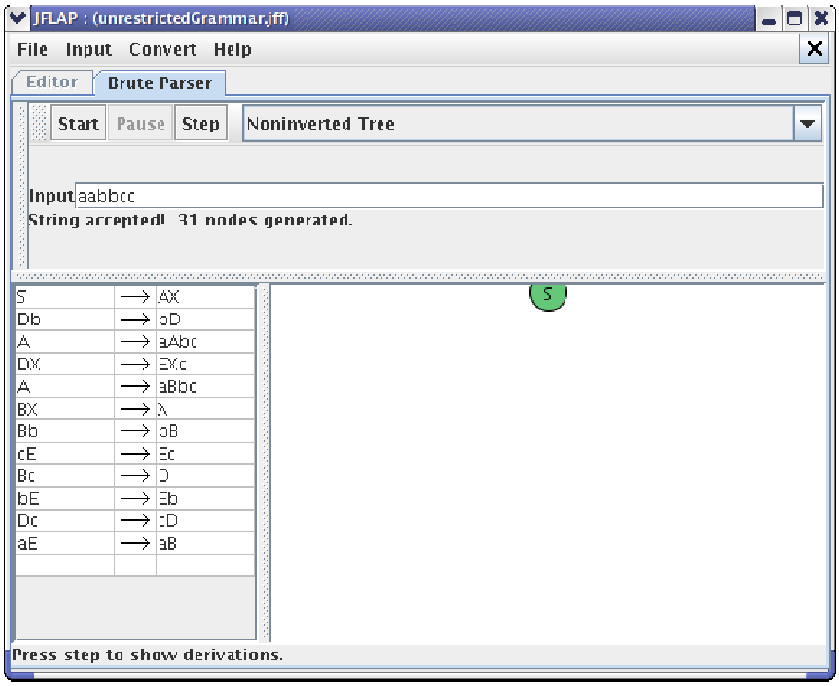
We will begin by introducing an unrestricted grammar for the language $a^n b^n c^n$, n>0. Write the following unrestricted grammar:



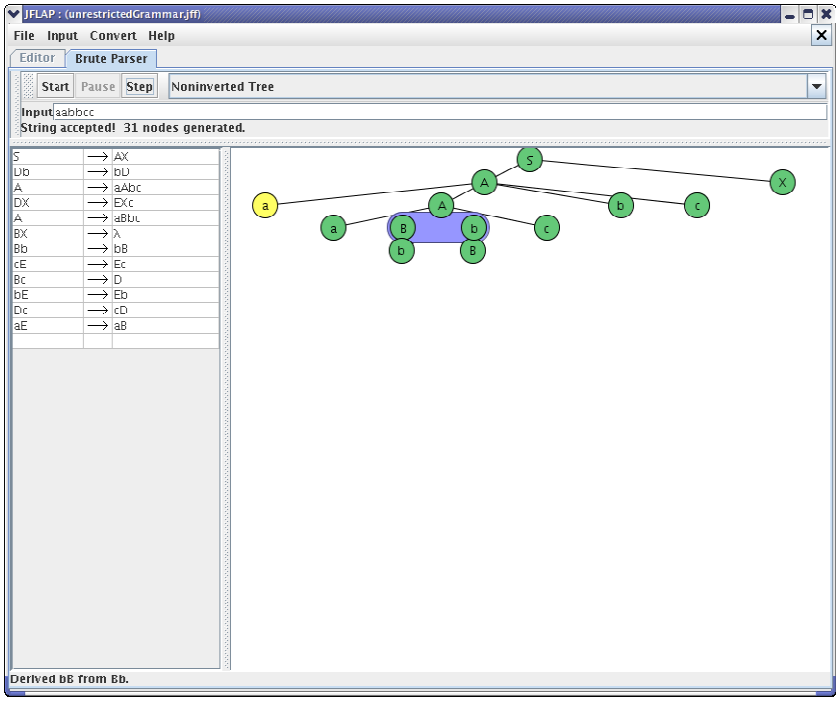
After clicking ***Input***, then ***Brute Force Parse***, you should see a window similar to this.



Now you can input a string and check whether it is accepted. Enter the string "aabbcc" ($a^2b^2c^2$) in the input text field. Then either click ***Start*** or press ***Enter key*.**

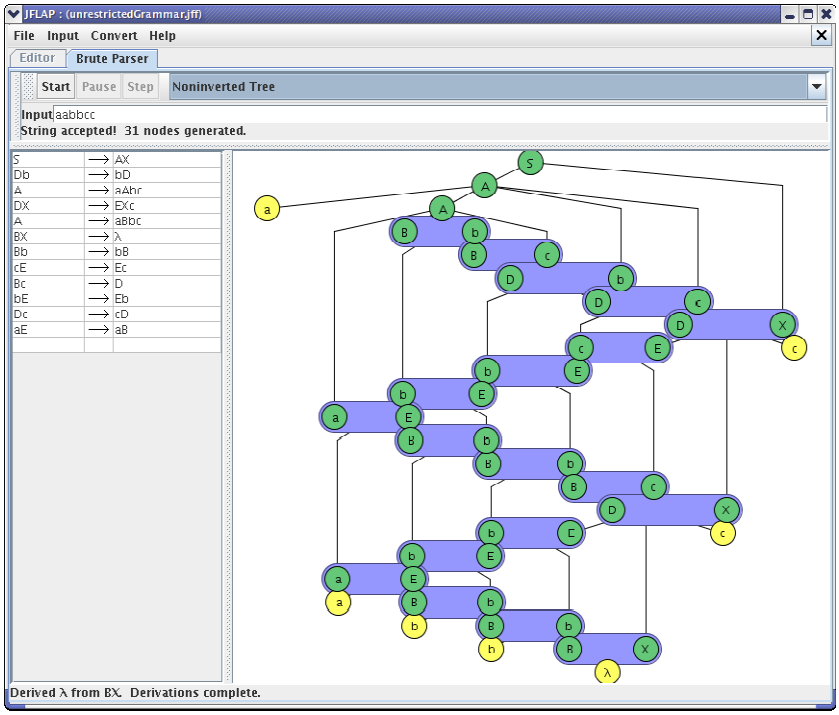


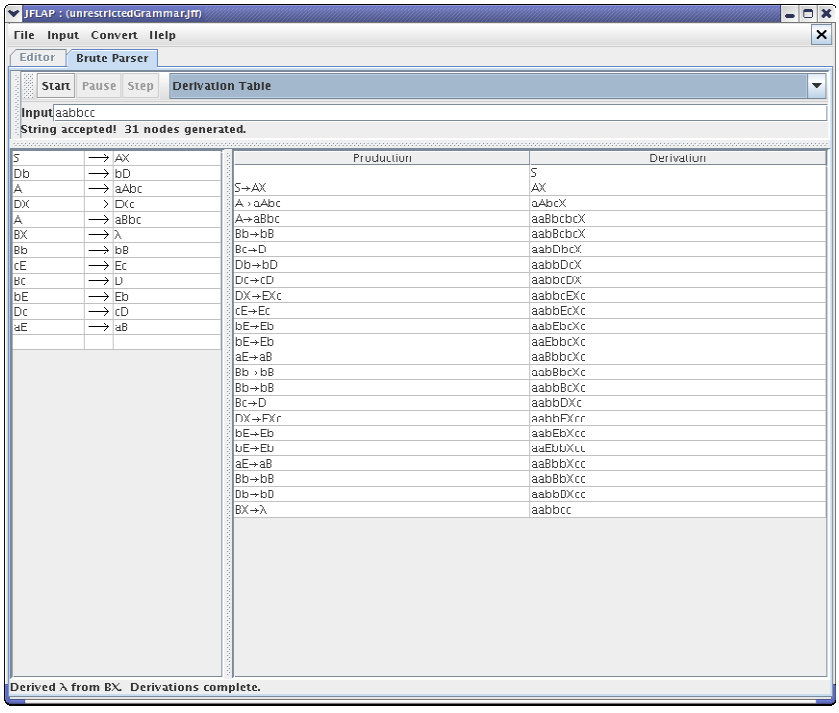
For the first three ***Steps***, Brute Force Parser runs just like it did with context-free grammar. However, after clicking the ***Step*** button the fourth time, you will notice that Brute Force Parser combines variable "B" and terminal "b" in a blue oval and replaces them with the two nodes b and B using the "Bb→bB" production. Your window should look similar to this one:



After clicking the ***Step*** button several times, it becomes clearer that Brute Force Parser is combining two adjacent variables or terminals (representing the left-side of productions) and replacing them with the right-side of their production. Finally, when you reach the last step, you will see that the input string is accepted. Here is the ***Non-inverted Tree*** and ***Derivation Table*** of

the final result. Note in the tree that the leaf nodes from left to right result in the string “aabbcc” we derived.



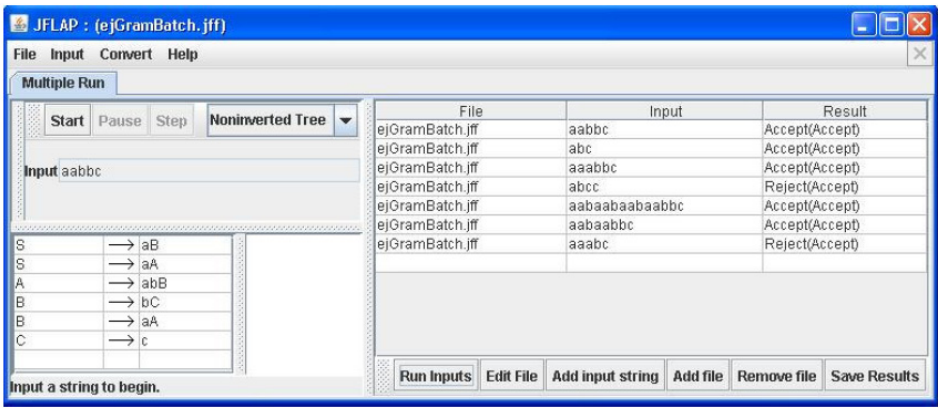


3.4. Batch Mode

Batch mode is a feature included in JFLAP to test files with grammars and words loaded from external files. Thus, you can easily verify the words from a list which are accepted or rejected by a grammar. This feature can be very useful if we want to do a quick test using saved files and without directly using the multiple brute force. As an example, we use the following grammar with a list of words:

S → aB S → aA A → abB B → bC B →aA C → c	aabbc abc aaabbc abcc aabaabaabaabbc aabaabbc aaabc

Enter the grammar in the JFLAP editor and save it. Then create a *.txt file including a column with the list of words. Once you have made these two steps, from initial JFLAP menu click ***Batch > Batch Test***. Introduce the name of the files with the grammar and the words. Then, you will see the following window.



By clicking on the button ***Run Inputs***, JFLAP checks which words in the file are generated by the grammar and which are not. You can also select a specific word and then represent the derivation tree.

From the same window, we can: edit grammar (***Edit > Edit File***); add more words; load another file with another grammar, and testing the same list of words with two different grammars (Add file); save the results (Save Results); clean the grammar, calculate the finite automaton and so on.

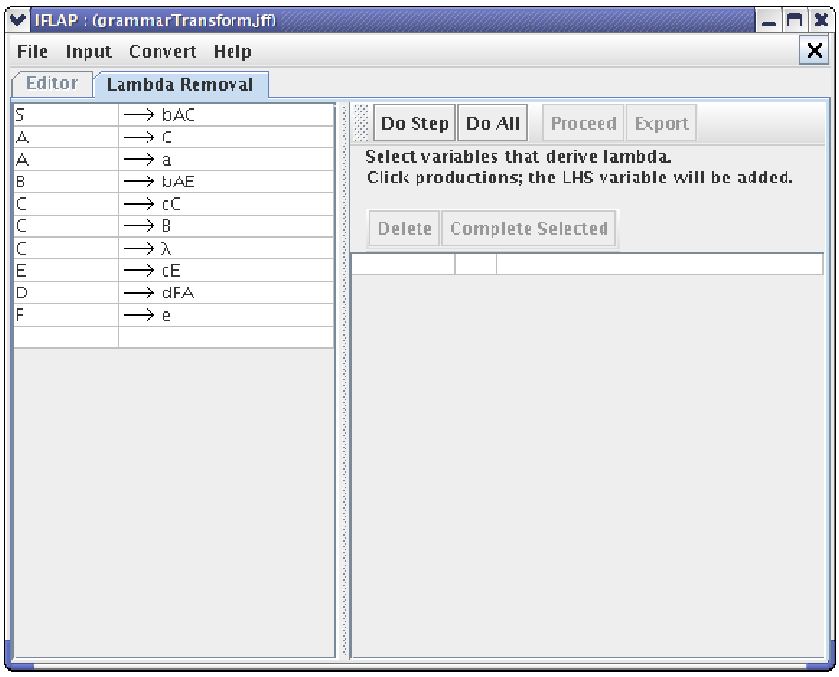
4. Transform Grammar

The Chomsky Normal Form (CNF) for a context-free grammar has a restricted format, in that the right-side of a production has either two variables or one terminal. CNF's restrictions result in many efficient algorithms, such as improving speed in Brute Force parsing.

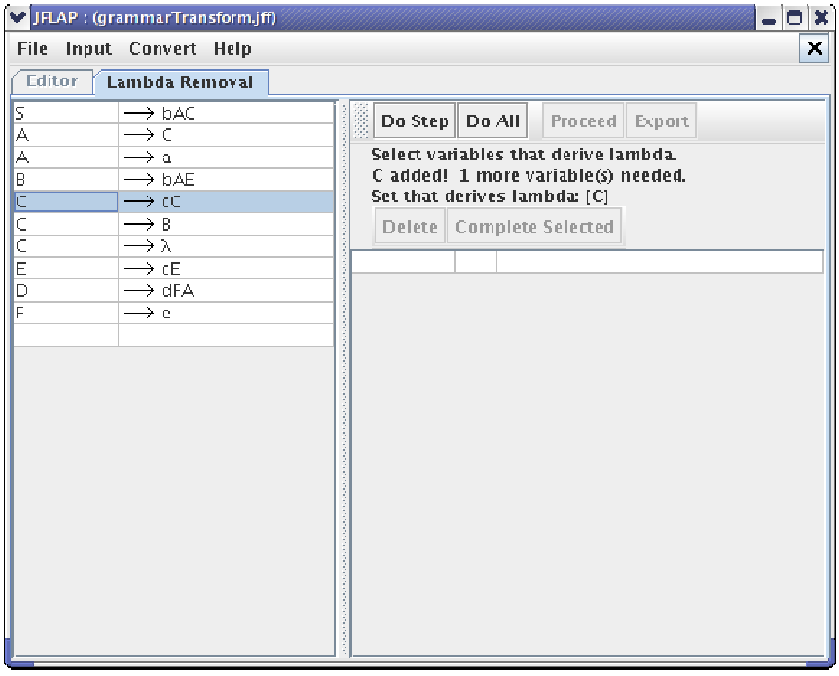
Given a grammar in the file, click on the ***Convert*** menu, then click on ***Transform Grammar***. Now, there are four steps that we have to go through in order to transform the grammar into CNF. They are removing lambda, unit, and useless productions, followed by a final step. Each step is discussed in the following sections.

4.1. Remove Lambda Productions

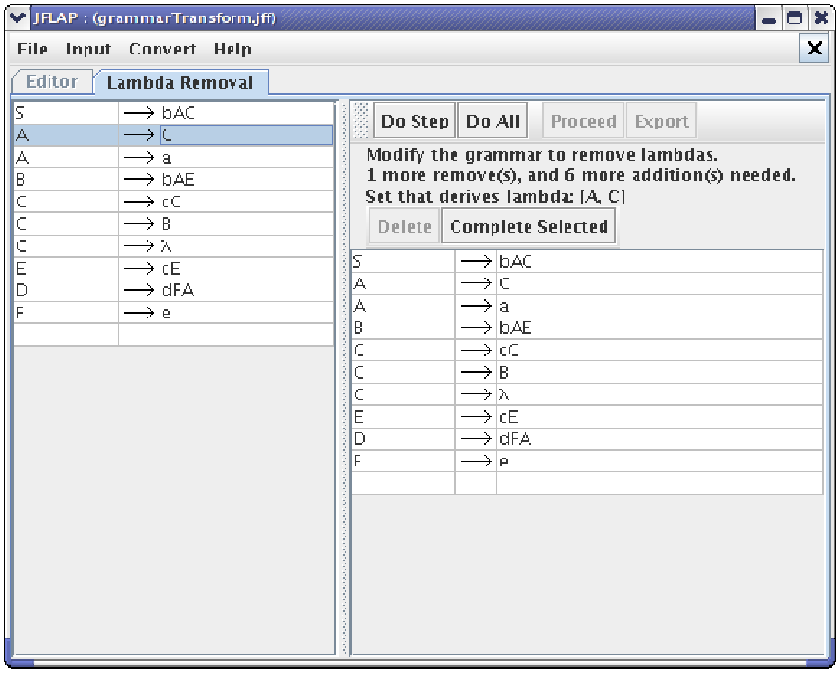
After clicking ***Transform Grammar***, JFLAP will open the tab ***Lambda Removal*** and you should able to see a window similar to this.



The first step is to identify the variables that can derive a lambda production. It is fairly obvious to see that the variable "C" can derive lambda via the production "C→λ" as it is a lambda production. Therefore, we should add the variable "C" to the set of variables that derive lambda by clicking on any "C" production. After clicking, your window should look like this:

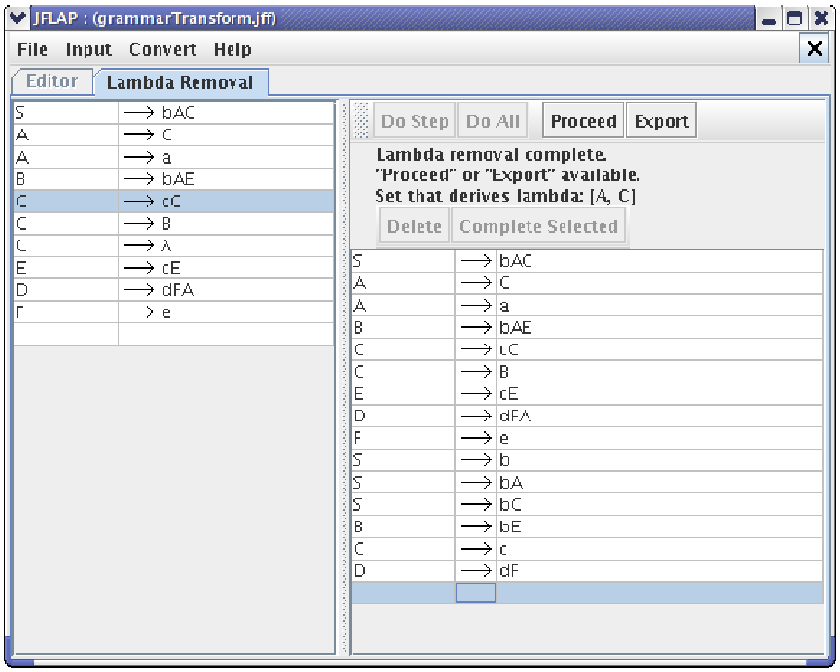


Now JFLAP tells us that there is one more variable that we need to identify. Notice that there is production "A→C". Since variable "C" could derive lambda, it can be concluded that "A→C" production could lead to a lambda production. Thus, we click on this production to add "A" to the set of variables that derive lambda. Your window should now look like this:



We have successfully identified all the variables that can derive λ. Note that a copy of the grammar appears on the right-side of the window for us to modify. Get rid of all λ-productions by clicking on them and clicking ***Delete***. There is one. Now, we have to add new productions to make sure that our grammar still accepts/rejects the same strings as it did before the lambda removal. JFLAP notifies you to add six more productions. A new production(s) must be created for productions with right-hand sides that contain a variable that could derive λ, it could have disappeared before.

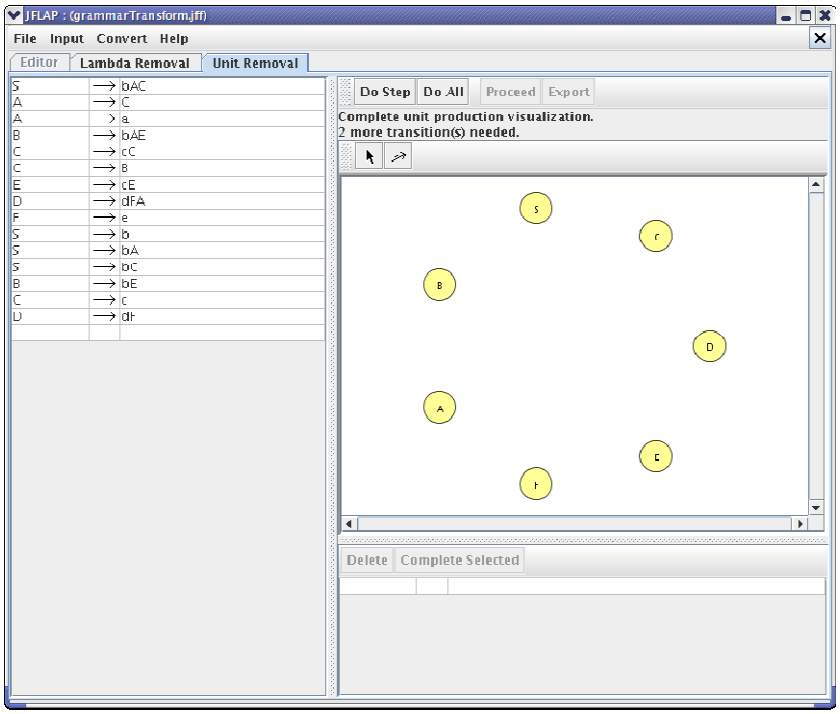
However, do not add any lambda productions. You can click on the production on the left side and click ***Complete Selected*** to let JFLAP add new productions related to that production. You can also manually add all the productions. For instance, since variable "C" could derive lambda, we could have derived terminal "c" from variable "C" ("C→cC" and derived C→λ). However, since we do not have lambda any more we have to add production "C→c" into our grammar. After entering all the productions, your window should look like:



Since lambda removal is complete, we can click ***Proceed*** and move on to the next step.

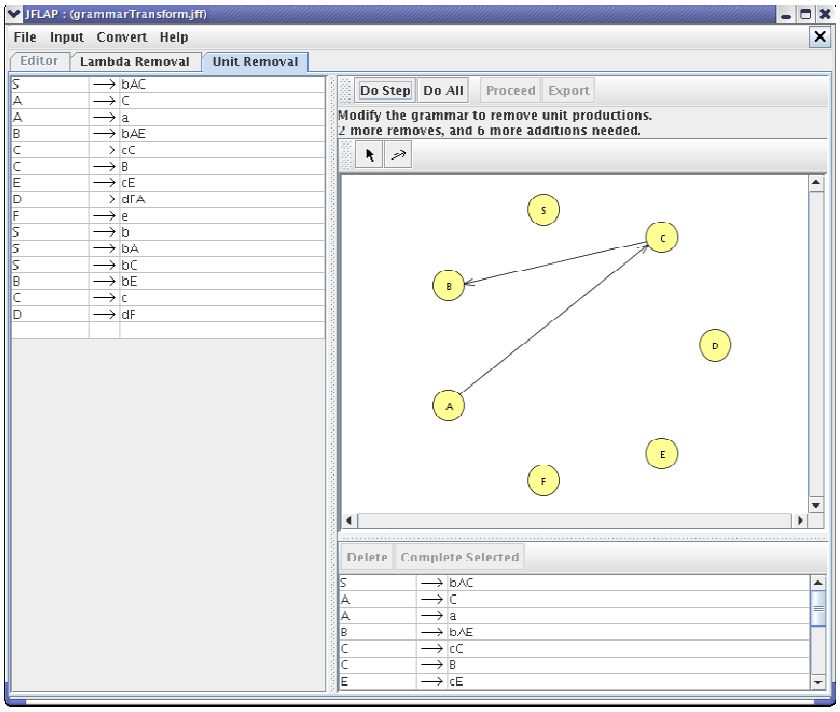
4.2. Remove Unit Productions

Now it's time to remove unit productions. Your unit removal window would look like this:

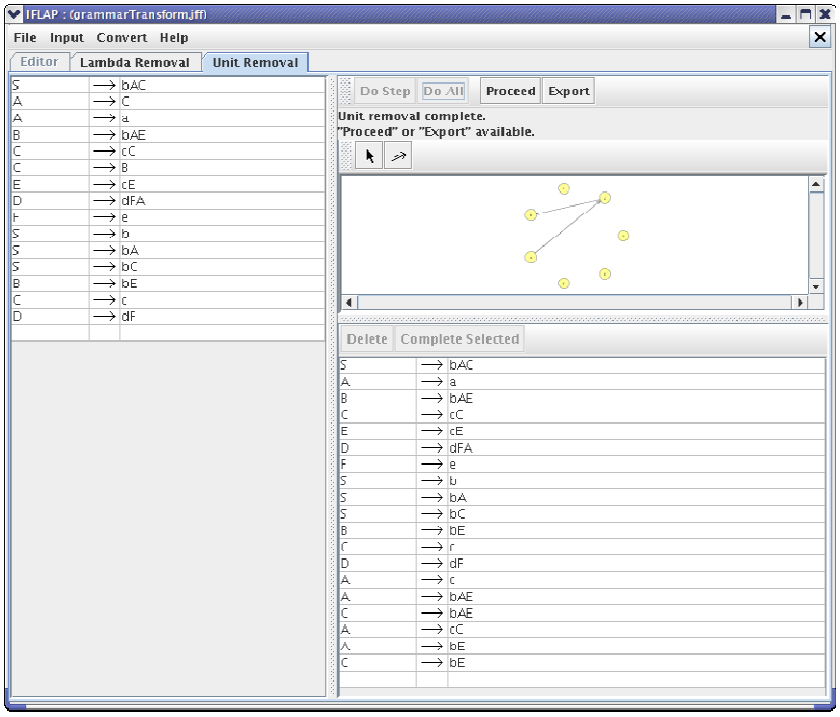


There is a picture with each variable from the grammar shown as a node in a graph with no edges. We will use the graph to see relationships between variables in unit productions. For each unit production "X→Y" add a directed edge(or transition) from node X to Node Y. JFLAP

notifies you that you need to add two more transitions to the unit production visualization. The variable "A" is directly connected to variable "C" through the production "A→C". For the similar reason, we can connect variable "C" to "B". After we finish adding these two transitions, our window should look like:



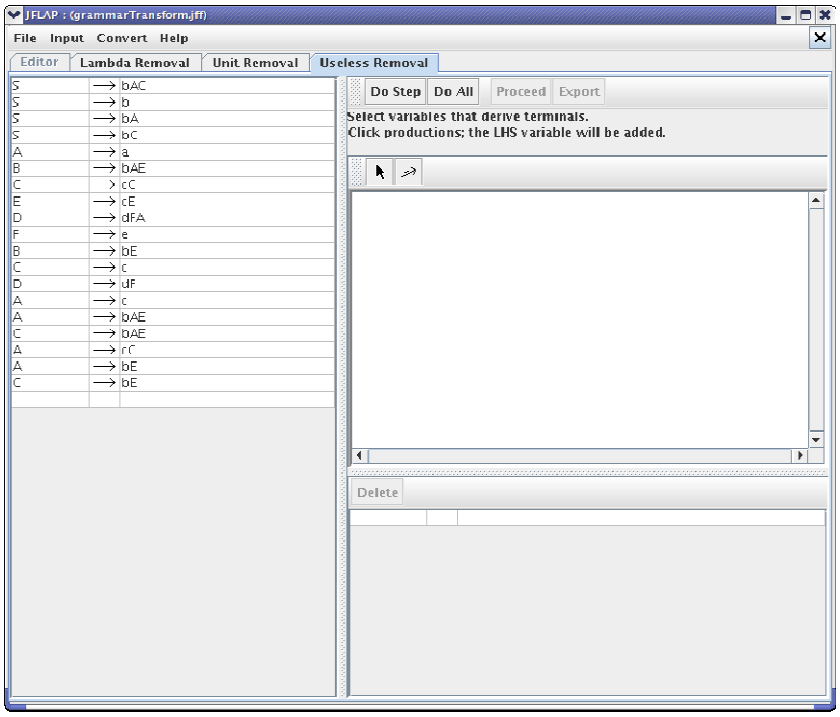
A copy of the grammar you can modify is now shown in the lower right window. First remove the unit productions by selecting them in the bottom right panel. Select the production and click ***Delete***. After the deletion is complete, we need to add new productions. For unit productions, both implicit and explicit (such as "A→B" from "A→C" and "C→B" as shown from the visualization), additional rules must be added to handle the missing unit productions. We have to make sure this changed grammar still accepts/rejects the same strings as it did before. For example, since we removed the production from variable "A" to variable "C", we cannot derive terminal "c" from variable "A". So, we have to add the production "A→c" into our grammar. JFLAP kindly notifies us how many more productions we have to add. After we finish adding all the productions, we are finished with unit removal and the window should look like this:



Now, we click on **Proceed** button and move on to the next step.

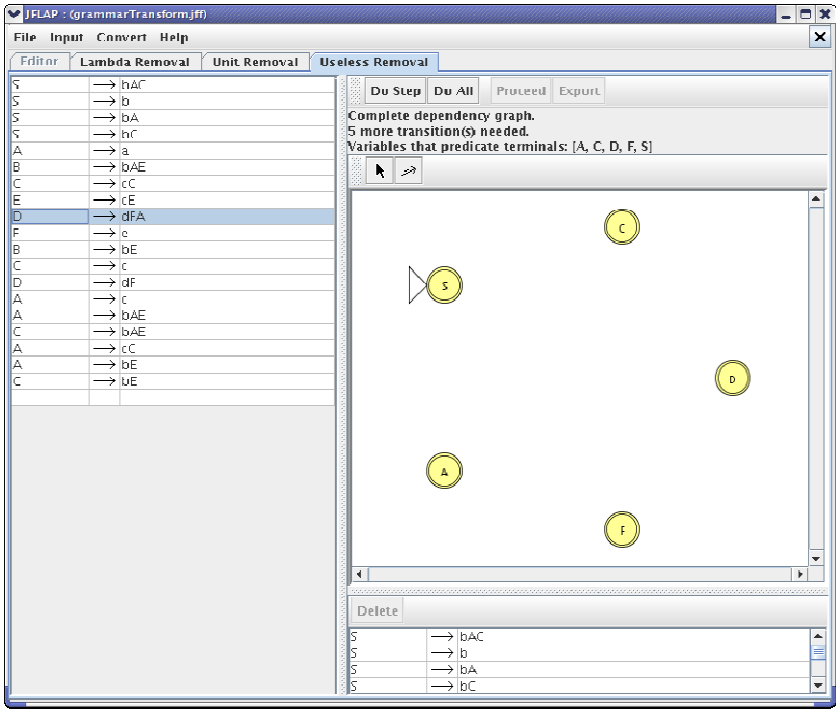
4.3. Remove Useless Productions

After successfully removing both λ and unit productions, your grammar window would look like:



A production is useless if no derivation can use it. Removing useless productions is a two-step process. First, find useless variables that cannot derive any string of terminals and remove productions with those variables. Second, find variables that are unreachable from the start symbol and remove productions with those variables.

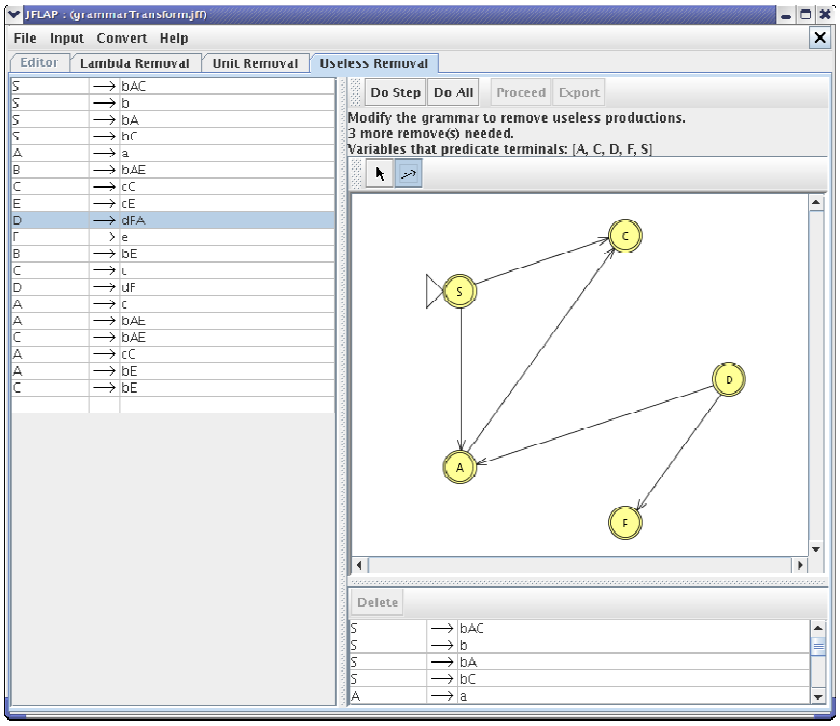
As JFLAP indicates, first we click on variables that derive terminals directly. They are variables "S", "A", "C", and "F". Also, if there are productions with only these variables on the right-side, then the variable on the left-side can also derive a string of terminals, and should be added such as "D" from "D→dF". Click on variable "D" to add it. After clicking these variables, your window should look like:



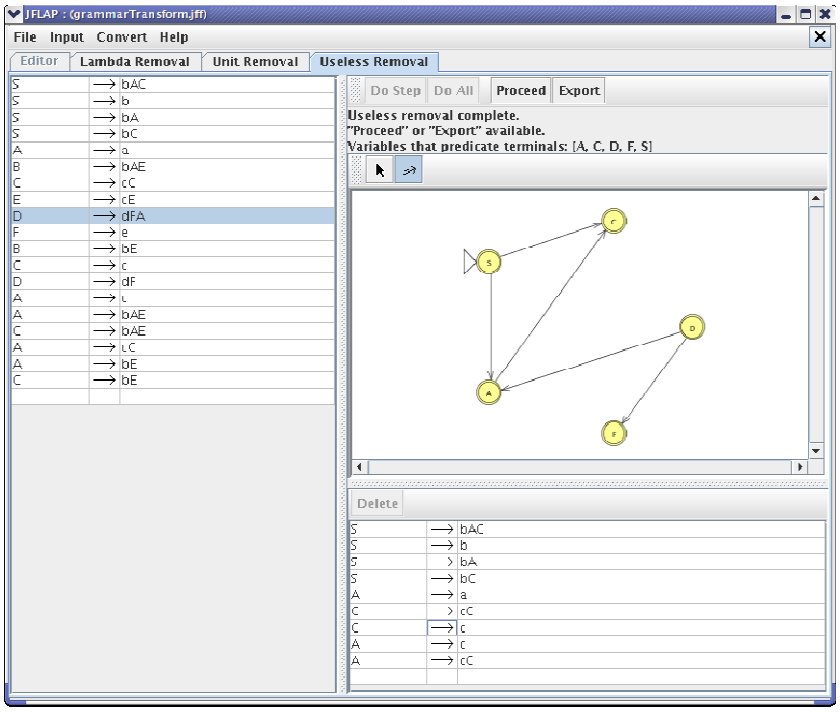
There are two new items shown. A visualization graph and below it, a copy of the grammar is shown, with rules missing that have variables that could not derive a string of terminals, productions with the variables "B" and "E".

The visualization has a node for each variable in the remaining grammar. This graph is different than the graph for removing unit-productions. For this graph, add a transition from variable "X" to variable "Y" if there is a production with "X" on the left-side and "Y" anywhere on the right-side.

Now, we have to add five transitions among these variables. From the production "S→bAC", we know that variable "S" depends on both "A" and "C". We also know that "A" depends on "C" based on the production "A→cC". We will add these transitions to our state diagram, by clicking the arrow in the panel and creating transitions. Add the remaining transitions. Now your window would look like:



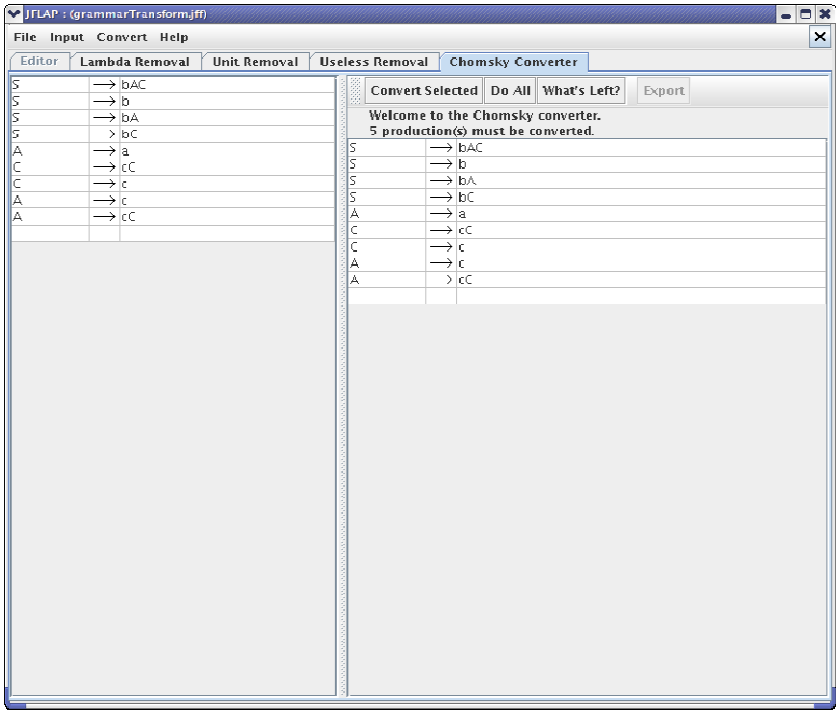
We now need to remove rules with those variables that are not reachable from “S”. That would be rules with “D” and “F” as those nodes are not reachable from “S” in the graph. Click on them and then click ***Delete***. After eliminating all of the useless productions, your window should look like:



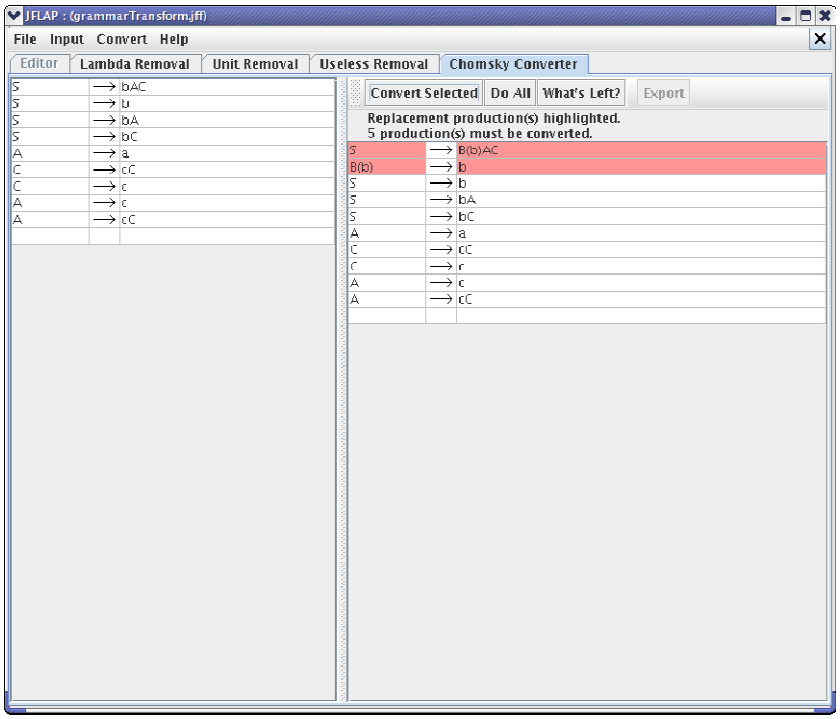
Now we have successfully removed useless productions. By clicking on the ***Proceed*** button, we will move on to our final step.

4.4. Convert to CNF

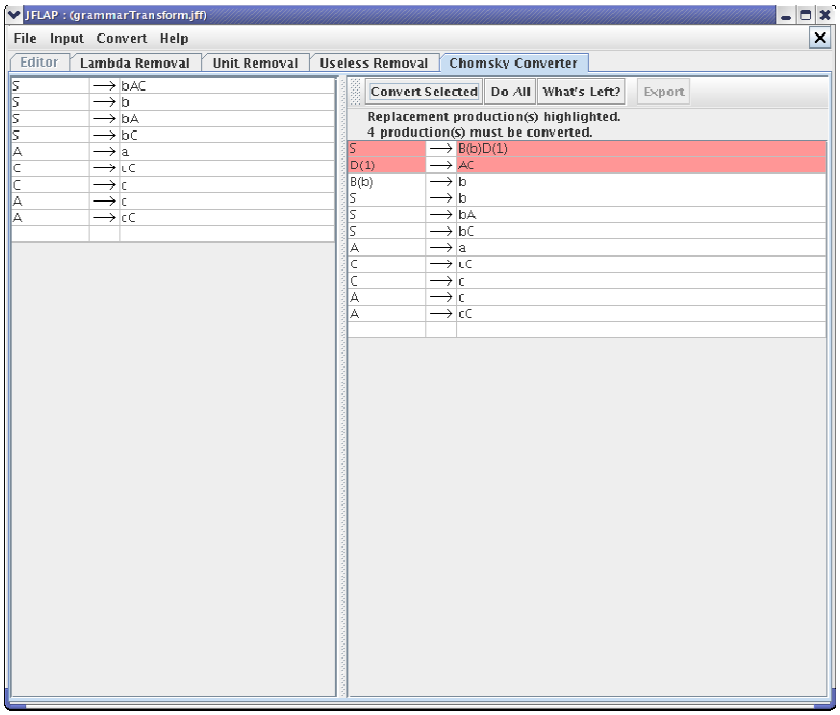
Now, we are at the final step of converting our grammar to CNF. In CNF, the right side of a production is either one terminal or two variables. If you have done everything correctly in previous steps, your window should look like this now:



Notice that the production "S→bAC" does not follow our CNF restriction and it must be converted. You can convert this production by clicking on the production on the right panel and click ***Convert Selected***. Now your JFLAP window will look like:



We have replaced the terminal “b” in “S→bAC” with a new variable “B(b)” and a production “B(b)→b”. The production “S→B(b)AC” now has all variables on the right-side, but has too many. Clicking ***Convert Selected*** again expands this production into two productions “S→B(b)D(1)” and “D(1)→AC” and another new variable “D(1)”. After following this step, your window should look like:



As JFLAP indicates, we have to convert four more productions. Try to select those productions, and create the missing productions. Alternatively, click on ***Do All*** to finish the conversion. After we are finished with the conversion, the grammar will be in CNF. Our final window should look like below and we can export this grammar to utilize in fast parsing.