JFLAP 8 Brute Force Parsing and Generate Language Walkthrough
Martha Kosa

By now, you have explored the topic of context-free grammars. Now you can practice testing a grammar with JFLAP.

Remember that a context-free grammar generates a language. It should generate exactly that set of strings specified, no more and no less. We cannot manually test every string over our alphabet because every non-regular context-free languages is infinite, but we can use testing to improve our confidence that the grammar is correct. Formal proofs can be used to establish correctness of grammars, but these may be long for complex grammars. Remember that in software, testing is used to detect the presence of bugs, not their absence.

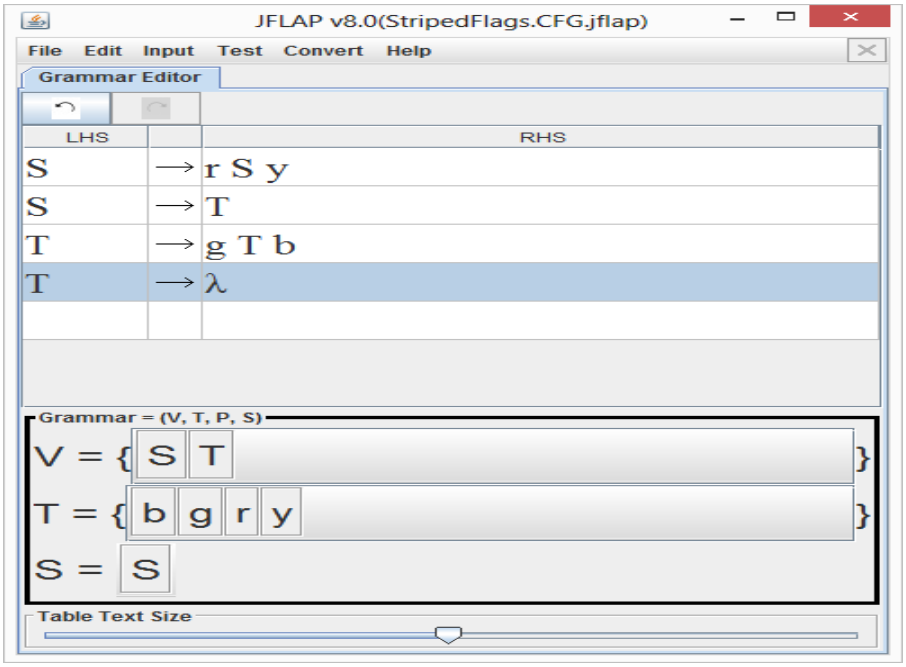
Let use consider a grammar to generate patterns consisting of stripes that are red, green, yellow, or blue. Our desired patterns will have a sequence of red stripes, then a sequence of green stripes, then a sequence of blue stripes, and finally a sequence of yellow stripes. The numbers of red and yellow stripes will be the same, and the numbers of green and blue stripes will be the same.

Questions to Think About:

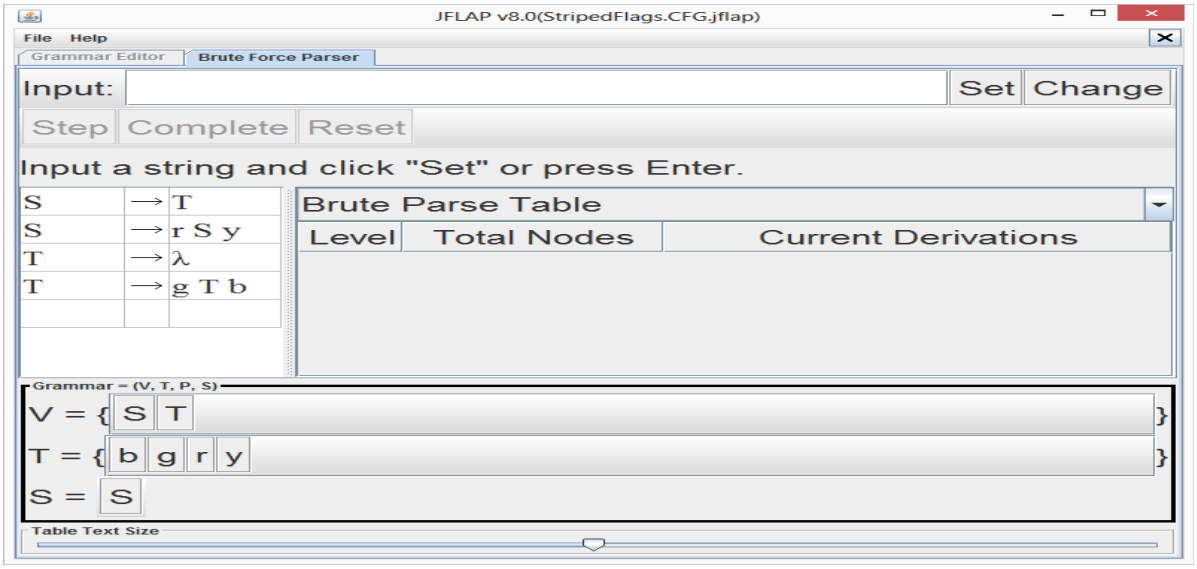
1. What is a possible alphabet for this language?
2. Based on this alphabet, write a formal definition for the language in set notation. You can use variables such as m and n to denote nonnegative integers.

Try It!

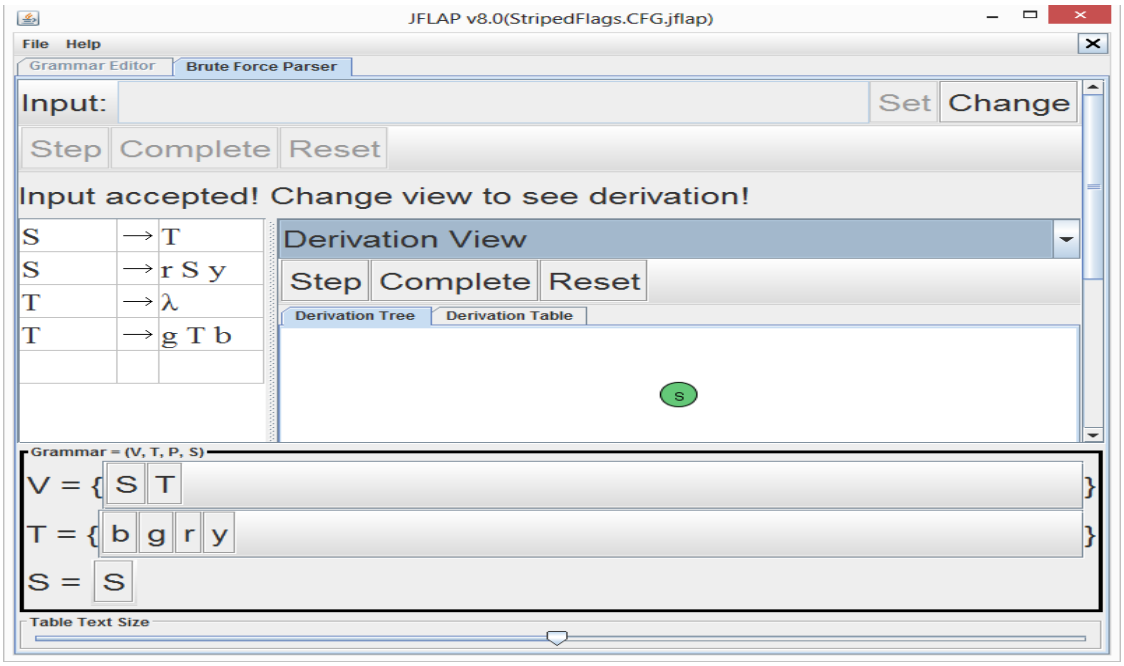
1. Start JFLAP and select ***File > Open*** to load the file **StripedFlags.CFG.jflap**. Your view should look like the following.



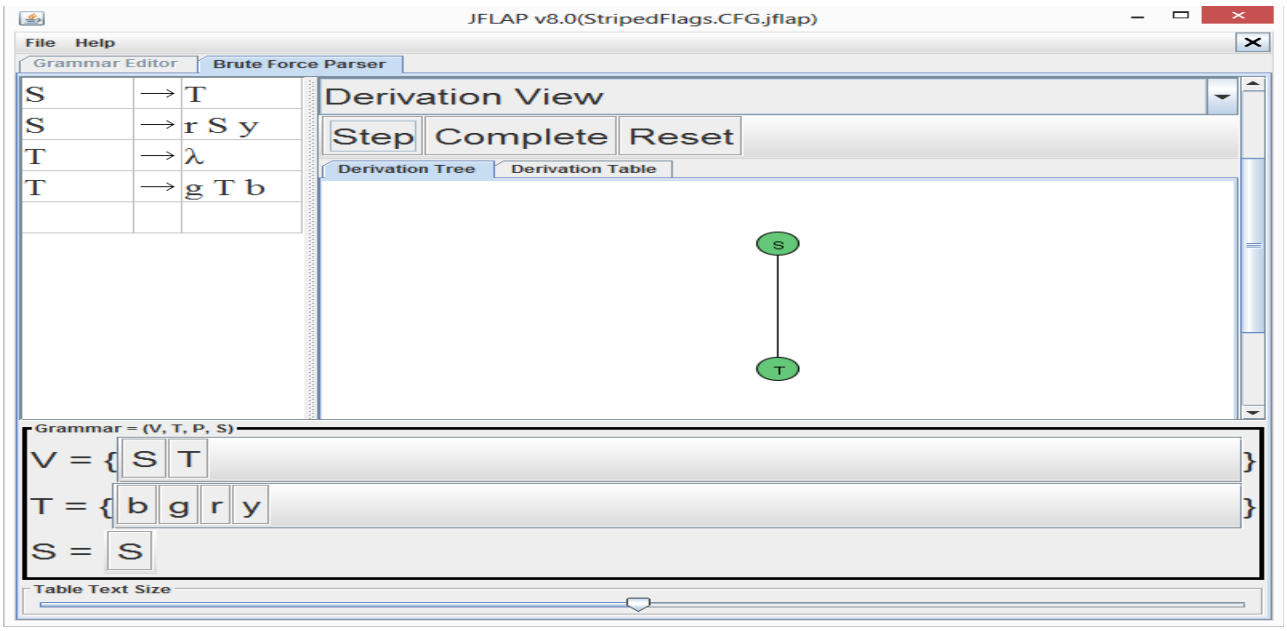
2. Select ***Input > Brute Force Parse***. Your view should look like the following.



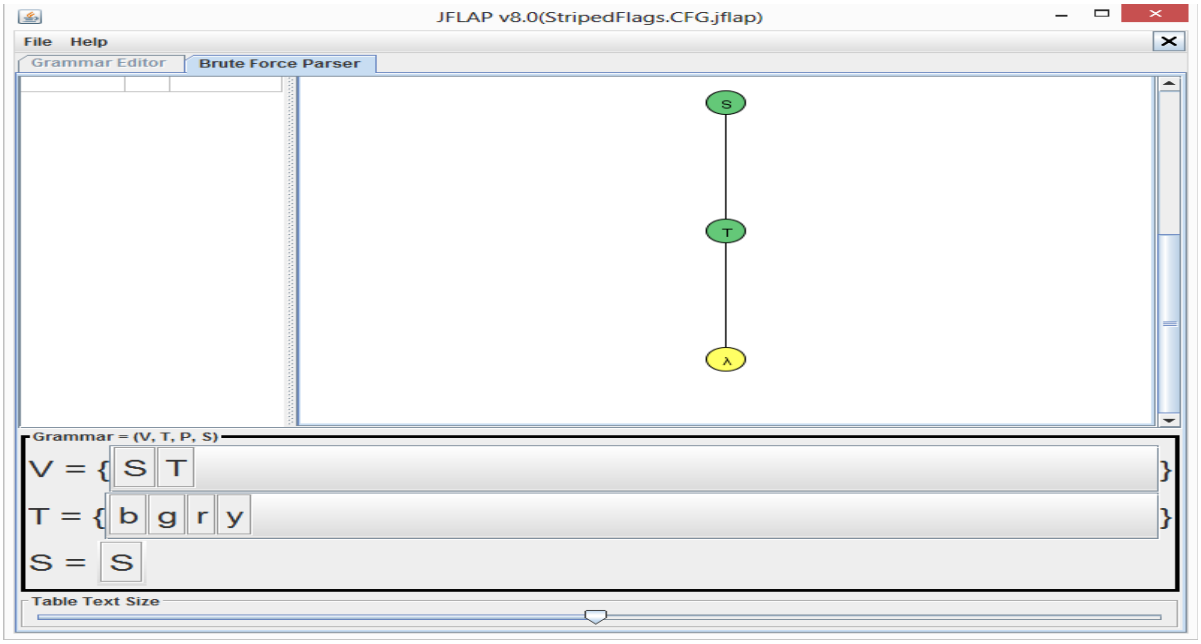
3. Click the **Set** button. What string are we parsing? Is it a valid string? We will soon see.
4. Click the **Step** button. What has been done? All right-hand sides of rules with S on the left-hand side have been expanded.
5. Click the **Step** button again. What has been done? The left-most nonterminal in each active derivation has been expanded, subject to a prefix match with the parsed string. The current derivation list contains our parsed string. Our string has been accepted.
6. Click the down arrow next to **Brute Parse Table** to select **Derivation View**. Your view should be similar to the following.



7. Click the **Step** button. Your view should be similar to the following when you scroll.



8. Click the **Step** button again. Your view should look similar to the following when you scroll. This is a parse tree for our string. The leaves of the parse tree contain the symbols of the string in order from left to right.

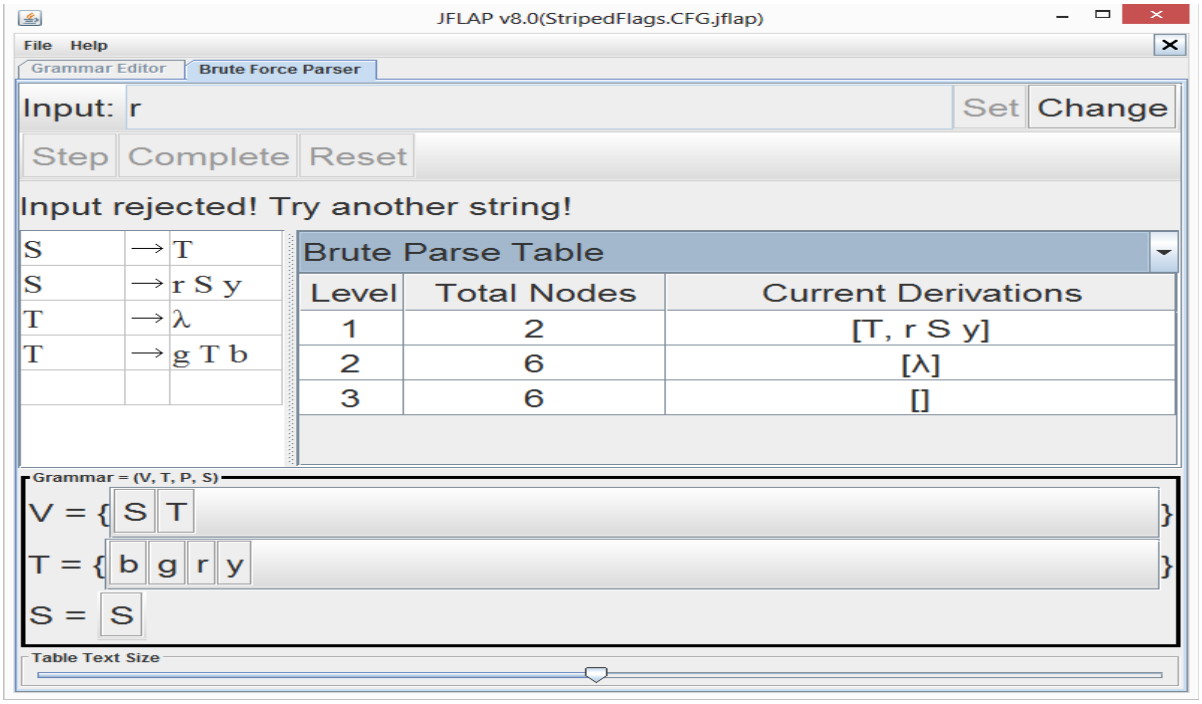


9. Use the techniques we just used to parse several non-empty valid strings with no g's and no b's. What happens as the lengths of your strings increase?
10. Use the techniques we just used to parse several non-empty valid strings with no r's and no y's. What happens as the lengths of your strings increase?
11. Use the techniques we just used to parse several non-empty valid strings with all four symbols. What happens as the lengths of your strings increase?

By now, we have confidence that any valid string can be generated. Let's now test to make sure invalid strings cannot be generate. How can a string over our alphabet {b, g, r, y} be invalid? It could have only one type of character, or the characters could be in an invalid order, or the numbers of b's and y's could be different, or the numbers of r's and g's could be different, to name a few possibilities.

Try It!

1. If JFLAP is still active, click the **Change** button. Otherwise, restart JFLAP and open the file **StripedFlags.CFG.jflap**.
2. Enter the string **r** and click the **Set** button.
3. Click the **Step** button until it is disabled. Observe what happens each time the button is clicked.



4. Perform the same tasks as above on a string containing only g's, a string containing only b's, and a string containing only y's. Don't forget to click the **Change** button!
5. Perform the same tasks as above on a string containing all four characters in an invalid order. Don't forget to click the **Change** button!
6. Perform the same tasks as above on strings containing a group of r's followed by a group of y's of a different size. What are the two general ways that the sizes can be different?
7. Perform the same tasks as above on strings containing a group of g's followed by a group of b's of a different size. What are the two general ways that the sizes can be different?

By now, you have confidence that no invalid strings can be generated.

To know for sure that the grammar generates the correct language, a mathematical proof based on derivations from the start symbol, the grammar rules, and the number of applications of the rules is necesssary. This proof would use mathematical induction as a technique.

This grammar generates an infinite language. We can let JFLAP generate some of the strings for us,

and then we can use the **Brute Force Parse** feature to produce the derivations and parse trees.

Try It!

1. If JFLAP is still active, select ***File > Dismiss Tab***. Otherwise, restart JFLAP and open the file **StripedFlags.CFG.jflap**.
2. Select ***Input > Generate Language***.
3. We first generate a certain number of valid strings. Enter **1** next to the **Generate:** label and click the **# of Strings** button. What is the first string generated?
4. Enter several increasing numbers, one at a time and clicking the **# of Strings** button as before. What happens as the entered numbers get larger?
5. The **Language Generator** feature also allows strings to be generated by their lengths. What is the length of the smallest valid string? Enter that number next to the **Generate:** label and click the **String Length** button.
6. What is true about the length of any valid string in this language?
7. Enter several increasing numbers, one at a time and clicking the **String Length** button as before. What happens as the entered numbers get larger?

Congratulations! You have successfully explored the **Brute Force Parser** and **Language Generator** features for testing context-free grammars in JFLAP. You can use these features now when testing your own grammars.