Context Free Grammar

A Context Free Grammar(CFG) is a set of rewriting rules that produce patterns of strings.

We will explain this concept via a canonical example which is the language $L = \{0^n 1^n | n \geq 0\}$. As an aside, it is useful to remember that this a language that is not regular (see regular pumping lemma).

Load the file **CFGModuleExample1.jflap** into JFLAP8 to follow along.

1 Mathematical definition

A Context Free Grammar is described by a 4 tuple (V, T, P, S) where V is the set of variables. T is the set of terminals. P is the set of what are called productions and S is the start symbol of the grammar.

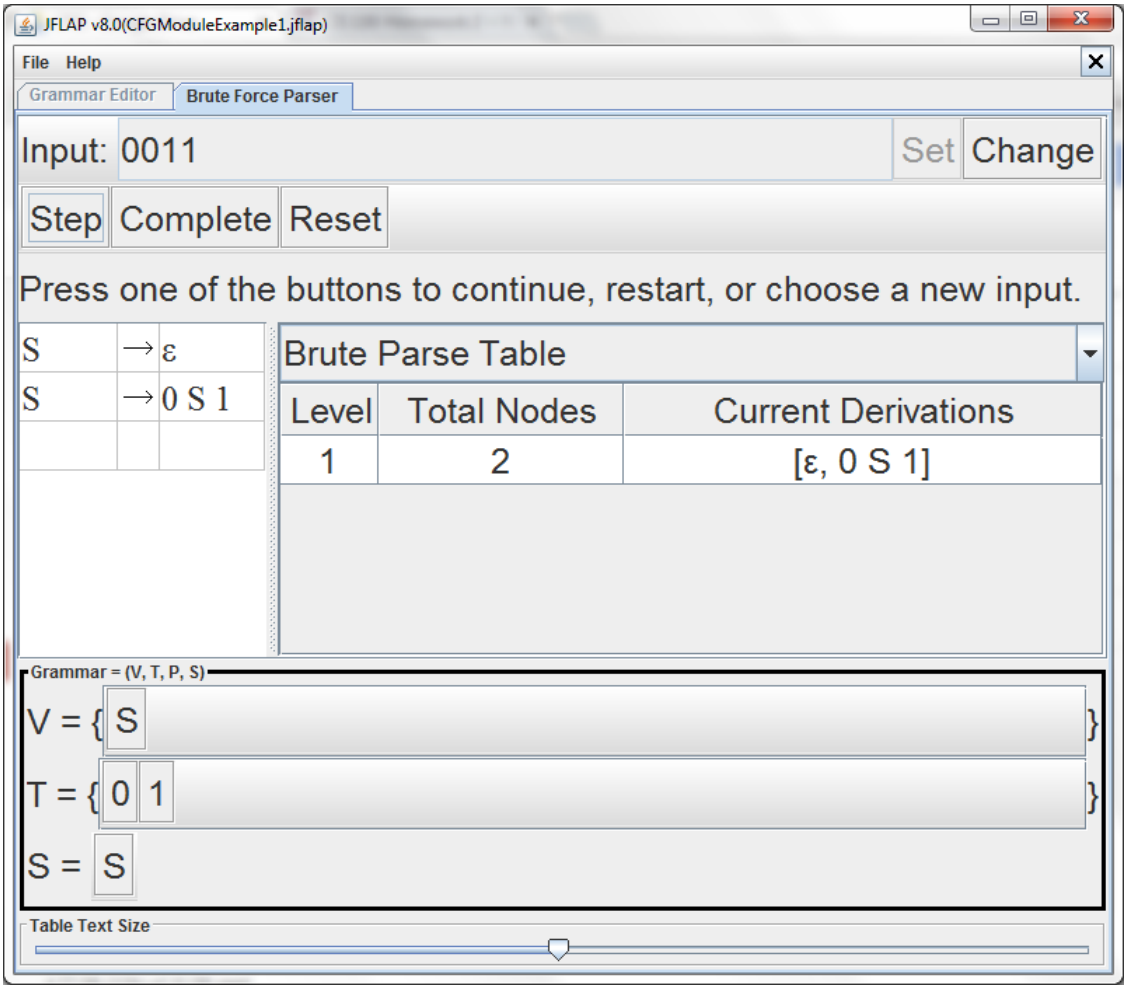
A production is basically a substitution rule. It consists of a symbol and a string separated by an arrow. The symbol is called a variable. The string on the right side of the arrow consists of some variables and some other symbols that are called terminals. A terminal is not allowed to be on the left side of an arrow for any of the rules. The terminals in Context Free Grammar are analogous to the input alphabet in automata.

One variable is called the start variable. Generally, that variable is denoted by S and generally while listing the production rules, a rule with that variable on the left side is listed first.

In the example, you will see the top line has a production rule $S \rightarrow 0S0$. S is a variable. 0 and 1 are supposed to be terminals. Note how 0 and 1 never appear on the left side of the rules.

So in our example $V = \{S\}$, $T = \{0, 1\}$ and the start symbol is S.

This is clearly indicated in JFLAP as well.



2 How a context free grammar produces strings

As mentioned before, a production rule is basically a substitution rule. The variable on the left side can be substituted with the string that is found on the right side.

To produce a string, you always begin with the start symbol and then follow production rules.

In our example, if we use rule 2, it is easy to see how this grammar produces the string ϵ.

For a more interesting string, let us use the other rule. Again, we begin with S. S gets replaced with $0S1$ by using the first rule. If we apply this same rule 3 times we get the string $000S111$. Now to stop this production process, we will use $S \rightarrow \epsilon$ to end up with the string 000111.

The notation used for this production is $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000111$

By trying out a few more examples of such substitutions, it should be clear that this CFG produces string of the form - a certain number of 0s followed by same number of 1s, which

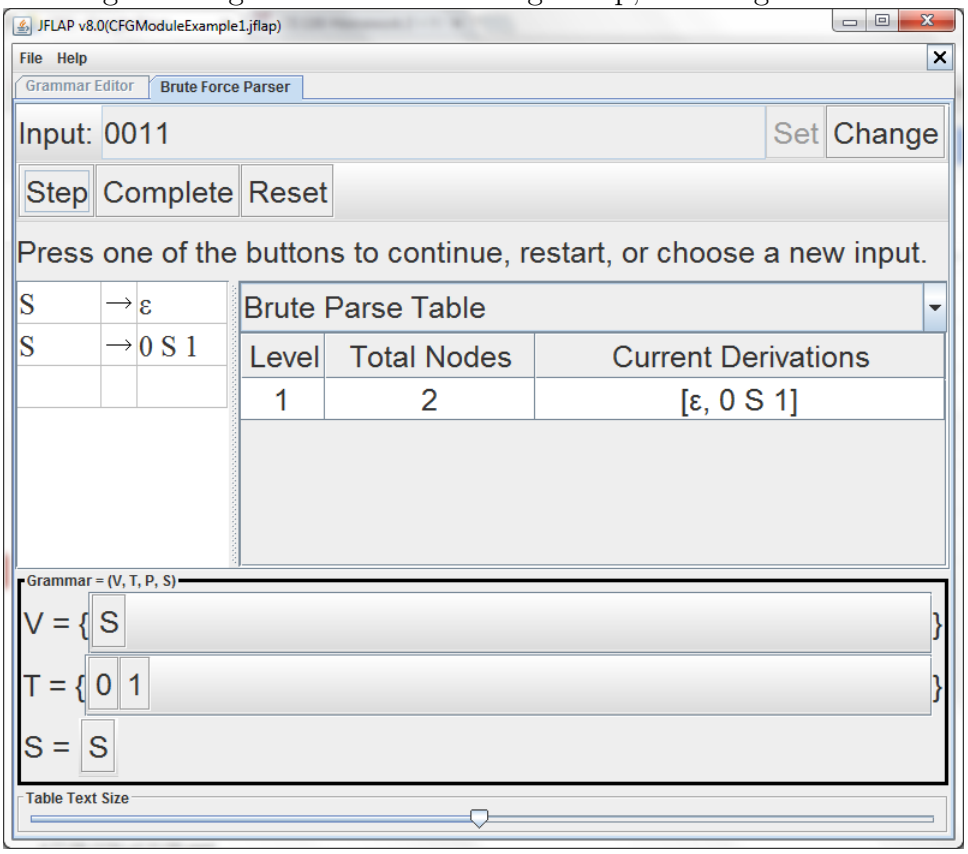
is exactly what the language L_1. We say, the context free grammar generates the language L_1.

3 Using JFLAP to generate strings

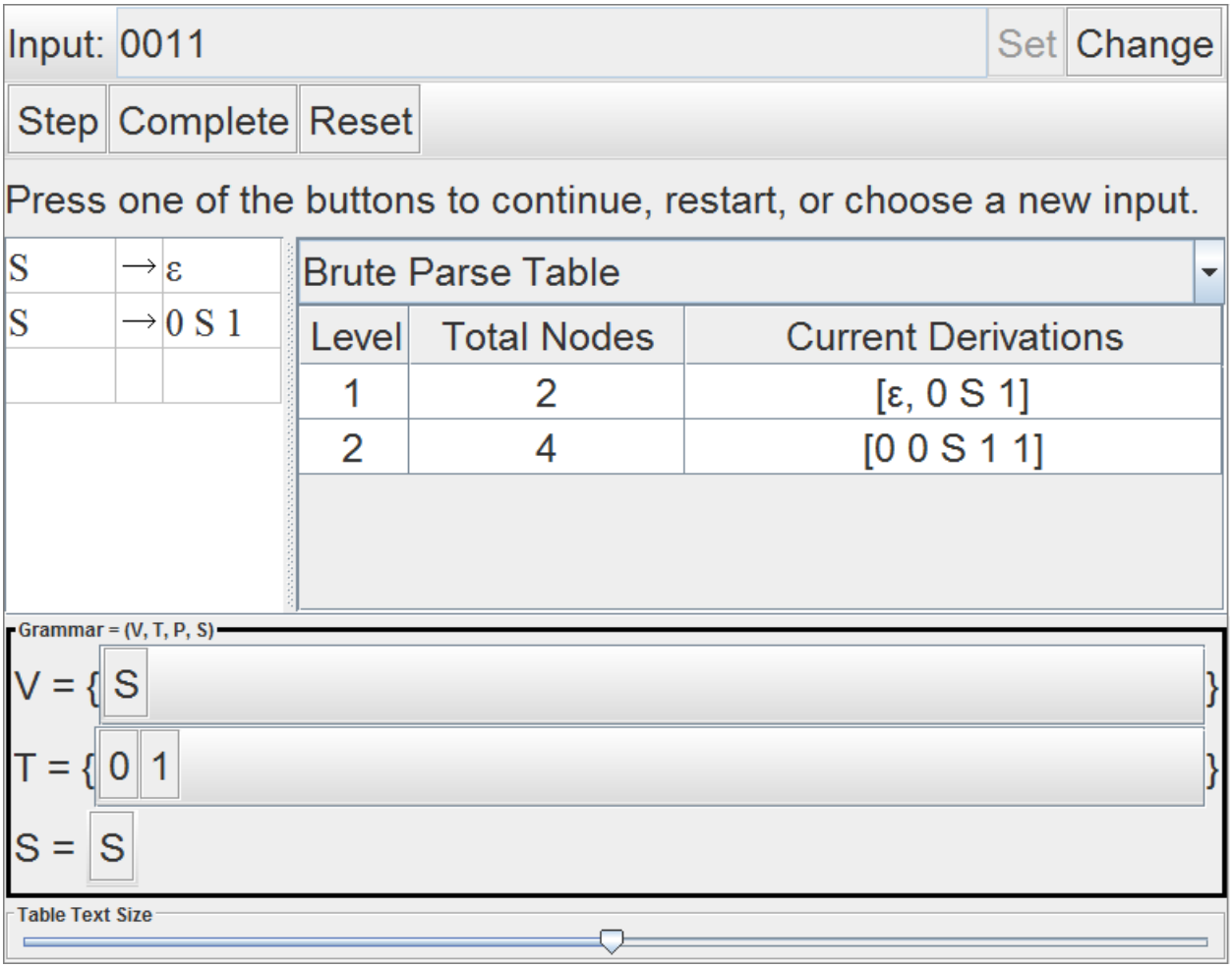
JFLAP allows you to step through the process of using the production rules to generate strings.

Load up the example, CFGModuleExample1.jflap. Click input and then Brute force parse.

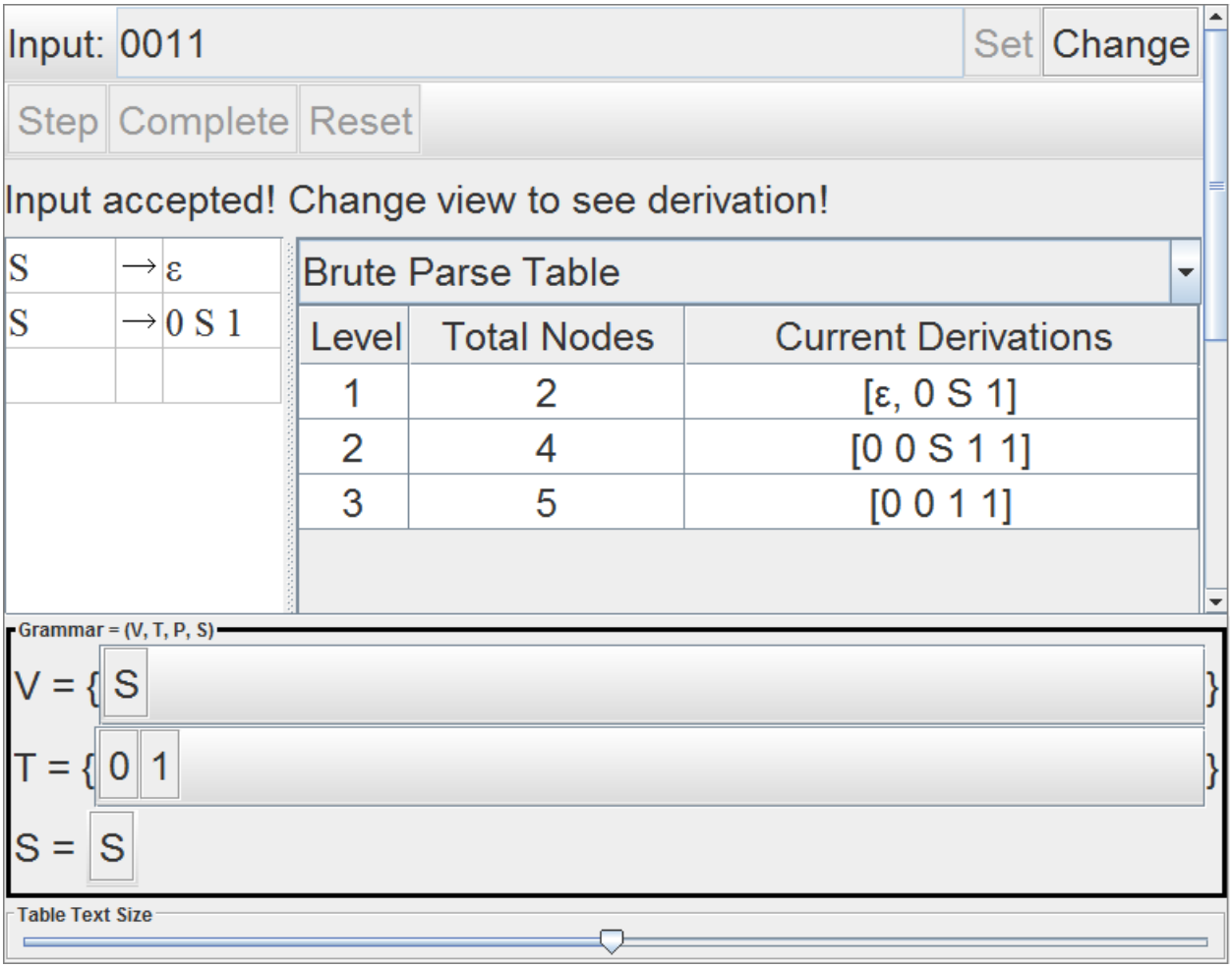
Let us try 2 strings, first 0011 and then 00111. Enter the input and click set. Now clicking the step button will make JFLAP step through the rules and see if the provided input string can be generated. With a single step, we can get ϵ and $0S1$ as shown here



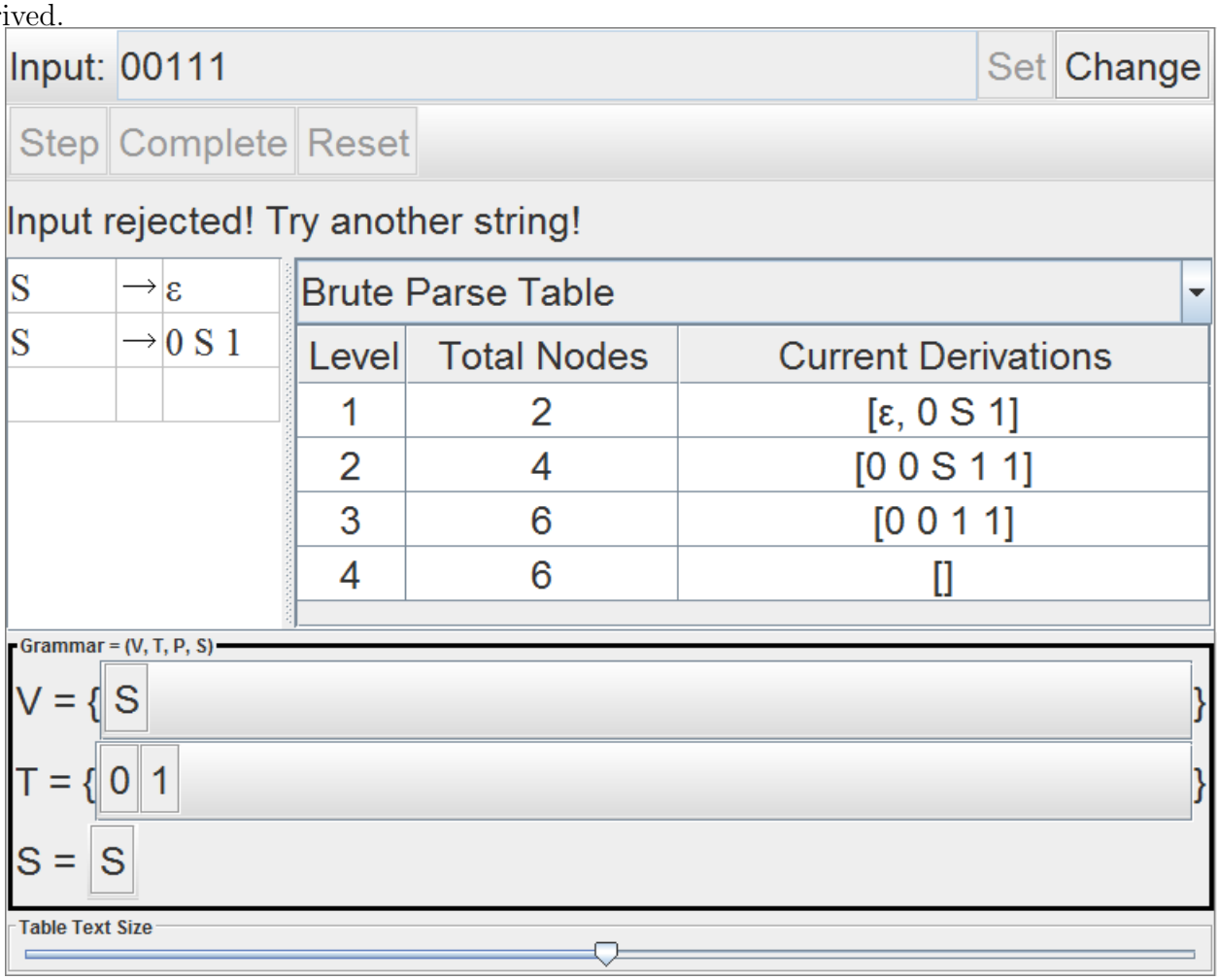
Click on step again and see that we get $00S11$ as a result of using the rules $S \to 0S1$



Finally clicking step once more gives us the string 0011 by using the rules $S \to \epsilon$.


Input: 0011
Set
Change
Step
Complete
Reset
Input accepted! Change view to see derivation!
S → ε
S → 0 S 1
Brute Parse Table
Level
Total Nodes
Current Derivations
1
2
[ε, 0 S 1]
2
4
[0 0 S 1 1]
3
5
[0 0 1 1]
Grammar = (V, T, P, S)
V = { S }
T = { 0 1 }
S = S
Table Text Size

On the other hand if we try a string like 00111 we find at some point the current derivations give you the empty collection [], meaning that the input string cannot possibly be derived.



4 Extension to the previous example

Now that the basic idea of a context free grammar has been described, let us try an extension of the previous example

$L_2 = \{0^n 1^n | n \geq 0\} \cup \{1^n 0^n | n \geq 0\}$

In the previous example we saw how to make a grammar that would give us strings where the 0s are before 1s. The same idea can be used to get strings that have the 1s before the 0s. The only thing remaining is how do we connect them up?. For this, recognize that there is a decision to be made in the beginning of either going down the 0s before the 1s or the 1s before the 0s.

So the start variable will have a substitution rule $S \to A$ and $S \to B$. The A will be responsible for then generating strings of the form $0^n 1^n$. The B will be responsible for

generating strings of the form 1^n0^n.

Try and complete the rules yourself and then enter them into JFLAP.

The complete solution is provided in CFGModuleExample2.jflap.

To convince yourself that this solution actually works, try using the brute force parse to see whether or not the following string are accepted $\{0011, 1100, 10, 101, 010011\}$.

5 Relationship to regular languages

Every regular language can be generated by a CFG. Make a non-terminal for every state and then any transition in a DFA from a state p to state q on a character a can be represented by the transition $p \to aq$. And for every final state f add the rule $f \to \epsilon$.

6 Questions to think about

1. What happens if you accidentally forget the rule $S \to B$ in the second example
2. Does this grammar generate all strings of the form

   $L = \{$strings that have the same number of 0s and 1s$\}$

   Try making a CFG that generates that language.
3. Given two context free grammars generating languages L_1 and L_2, can you always make a context free grammar that generates the union of the two languages.
4. Is there a CFG for the language $a^m b^n$ where $m \neq n$.

7 Answers

1. If you forget $S \to B$, there is no way to generate the strings that begin with a block of 1s followed by 0s. In this case, the rules that have the variable B on the left side are essentially unusable (unreachable).
2. The solution is provided in the file CFGModuleExample3.jff
3. To create a CFG that generates the union of the two languages, create a new start symbol S, and add two rules $S \to A$ and $S \to B$ assuming that A and B are the start symbols of the two languages.

Context Free Grammars are closed under the union operation.