CFG Example

Example Context Free Grammar

Load the file CFGExercise.jflap or alternately click on the Grammar option and enter the following rules

$$S \to TT$$
$$S \to U$$
$$T \to 0T$$
$$T \to T0$$
$$T \to \#$$
$$U \to 0U00$$
$$U \to \#$$

A context free grammar basically consists of a system of symbols, some of which can be replaced by other symbols via rules that are called production rules. For instance, in this particular grammar, the S can be replaced at any time by two Ts . The symbols that cannot be replaced are called the non-terminals. In the case of this particular grammar, the 0 and the # are terminals.

Grammars are used to provide rules to produce strings of terminals. The process of producing these strings always begins with one special symbol, the start symbol. The start symbol is usually the symbol that is on the left side of the top rule.

For instance, in this particular example, here is one way of producing 0##0.

1. $S \to TT$
2. replace the left T with $0T$. So we have $0TT$
3. replace the rightmost T with $T0$. We have $0TT0$.
4. finally replace both T with # to get the desired string

Brute Force Parsing

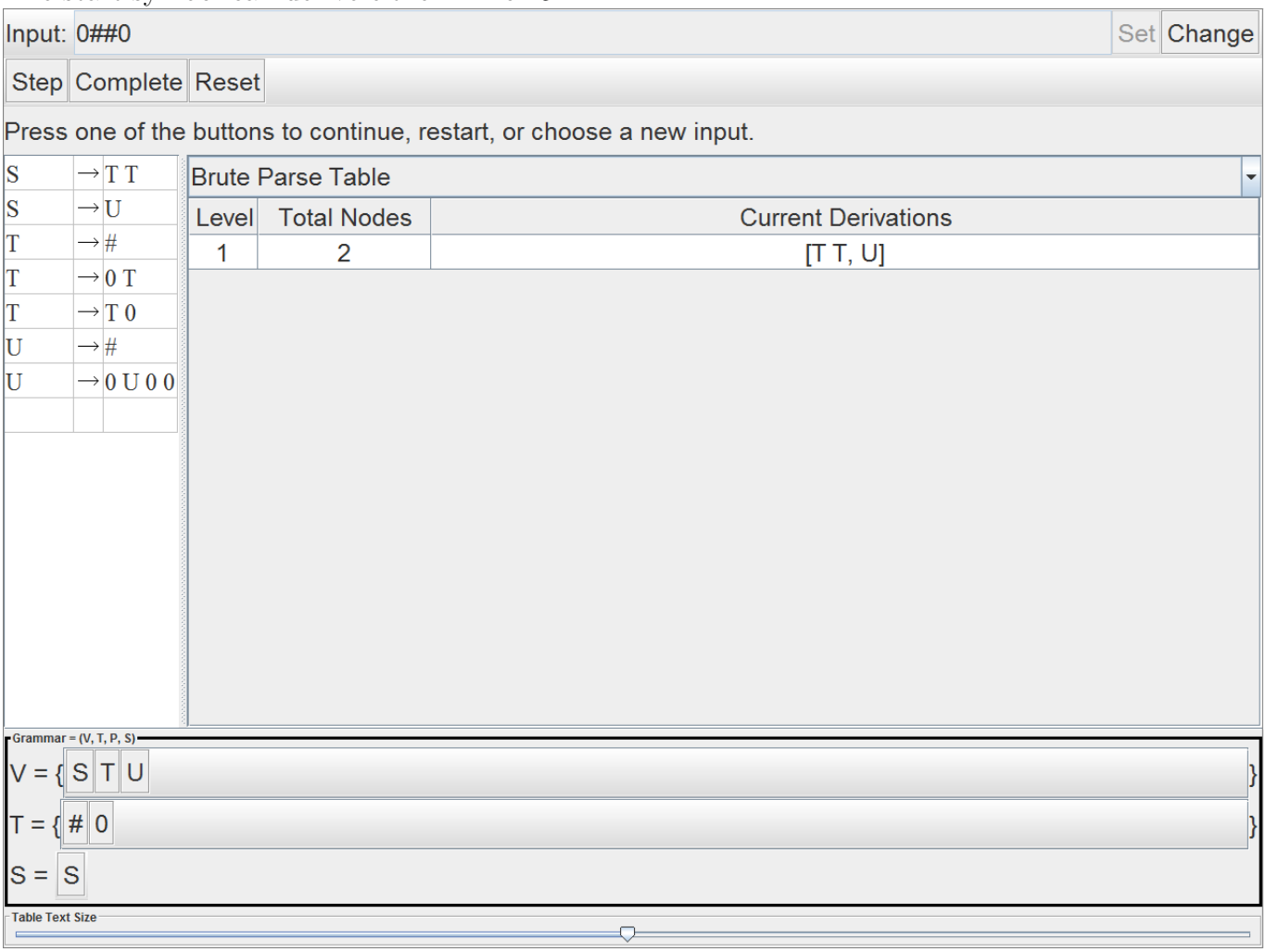
The process of trying to see whether a given string can be produced via the rules of a grammar is called parsing. There are several methods of parsing, some more efficient than others. Depending upon the properties satisfied by the grammar, certain efficient algorithms may or may not apply.

Regardless, one method that can always be applied is that of brute force parsing. This basically means we explore every possible path at each stage.

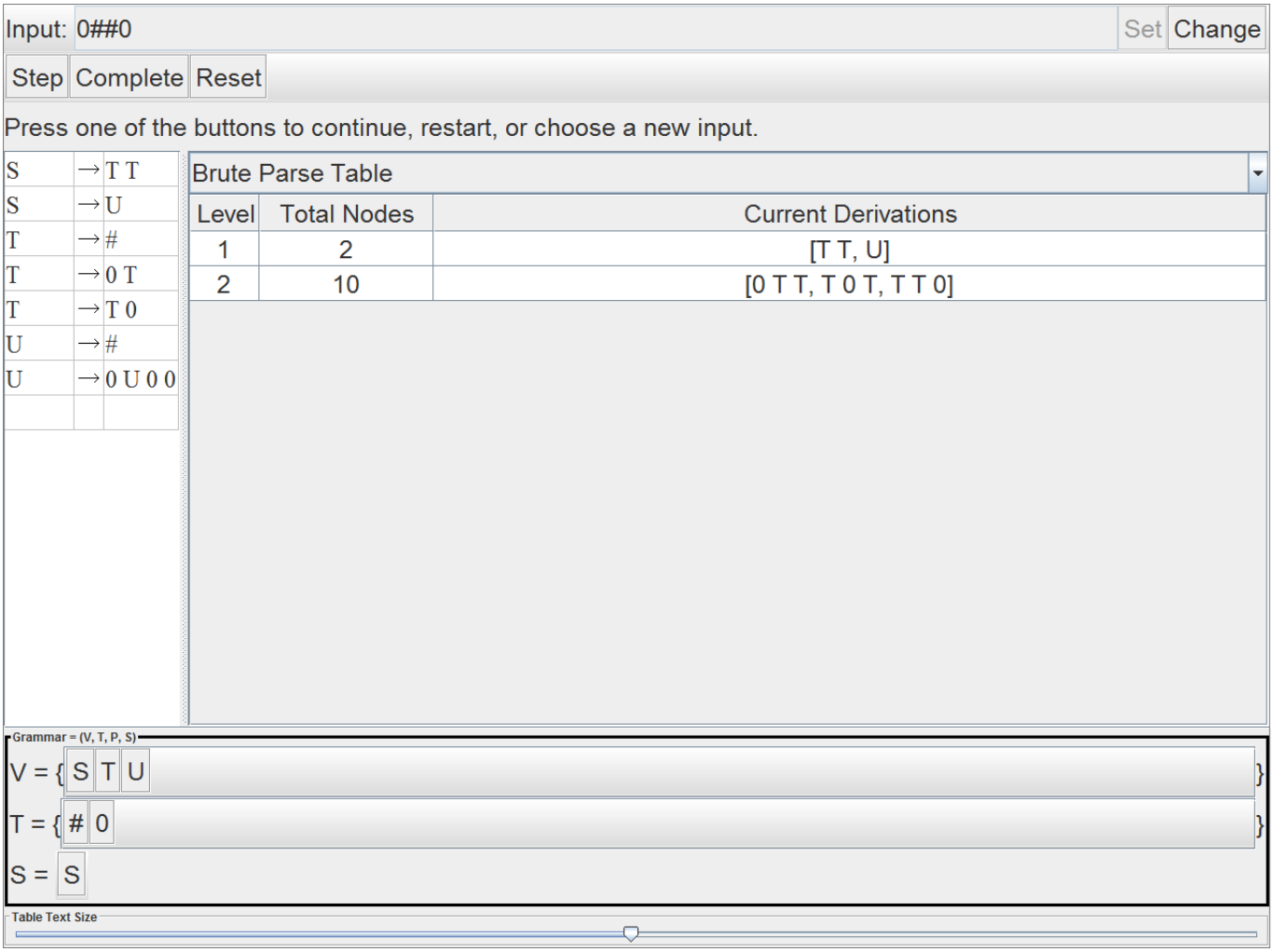
Let us try using the brute force parser in JFLAP to see how to produce 0##0

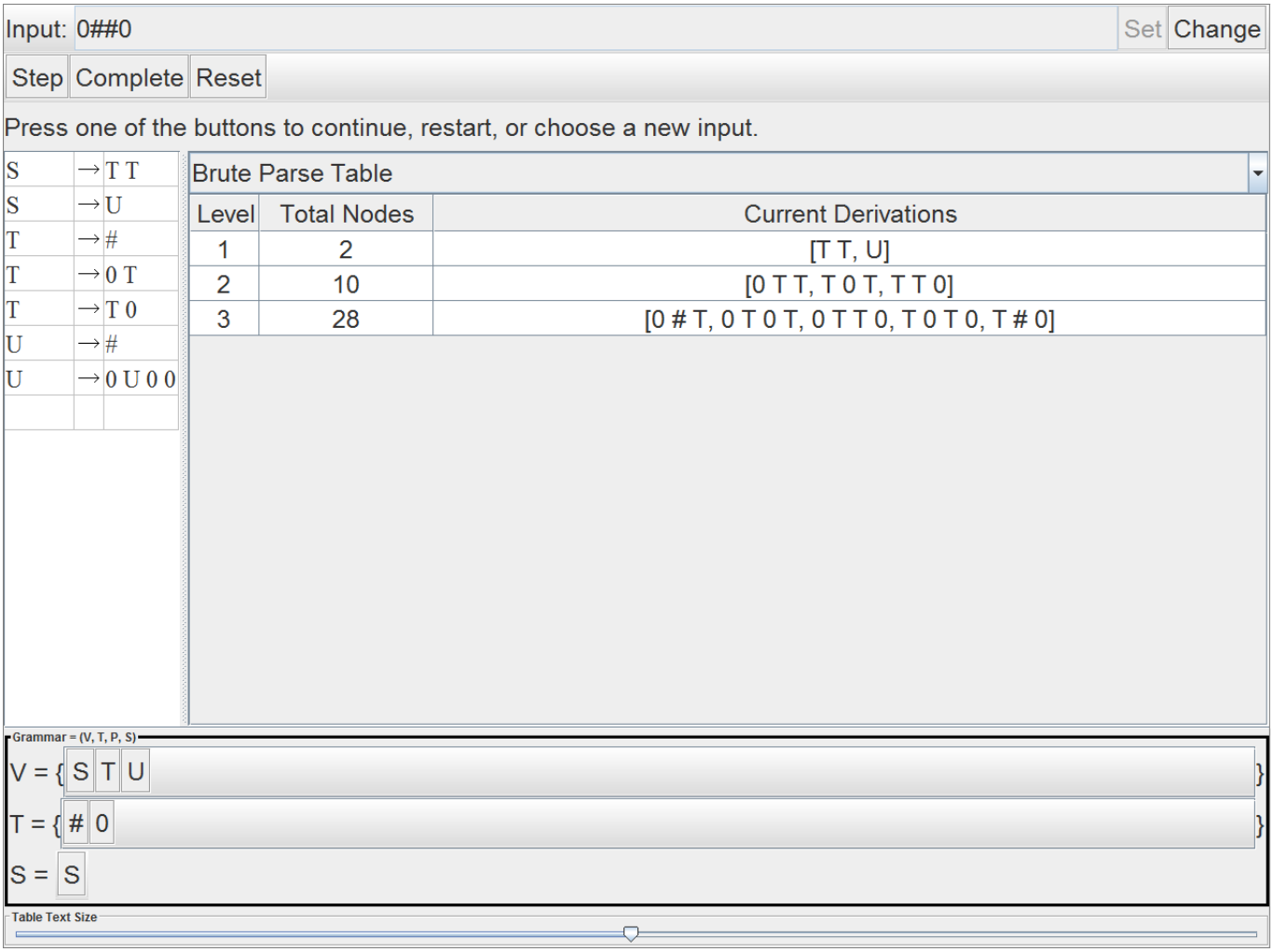
By clicking on Brute Force parse (in the input sub menu) you get the option of entering a string to be parsed. Clicking set and stepping through the parser, the following steps are executed

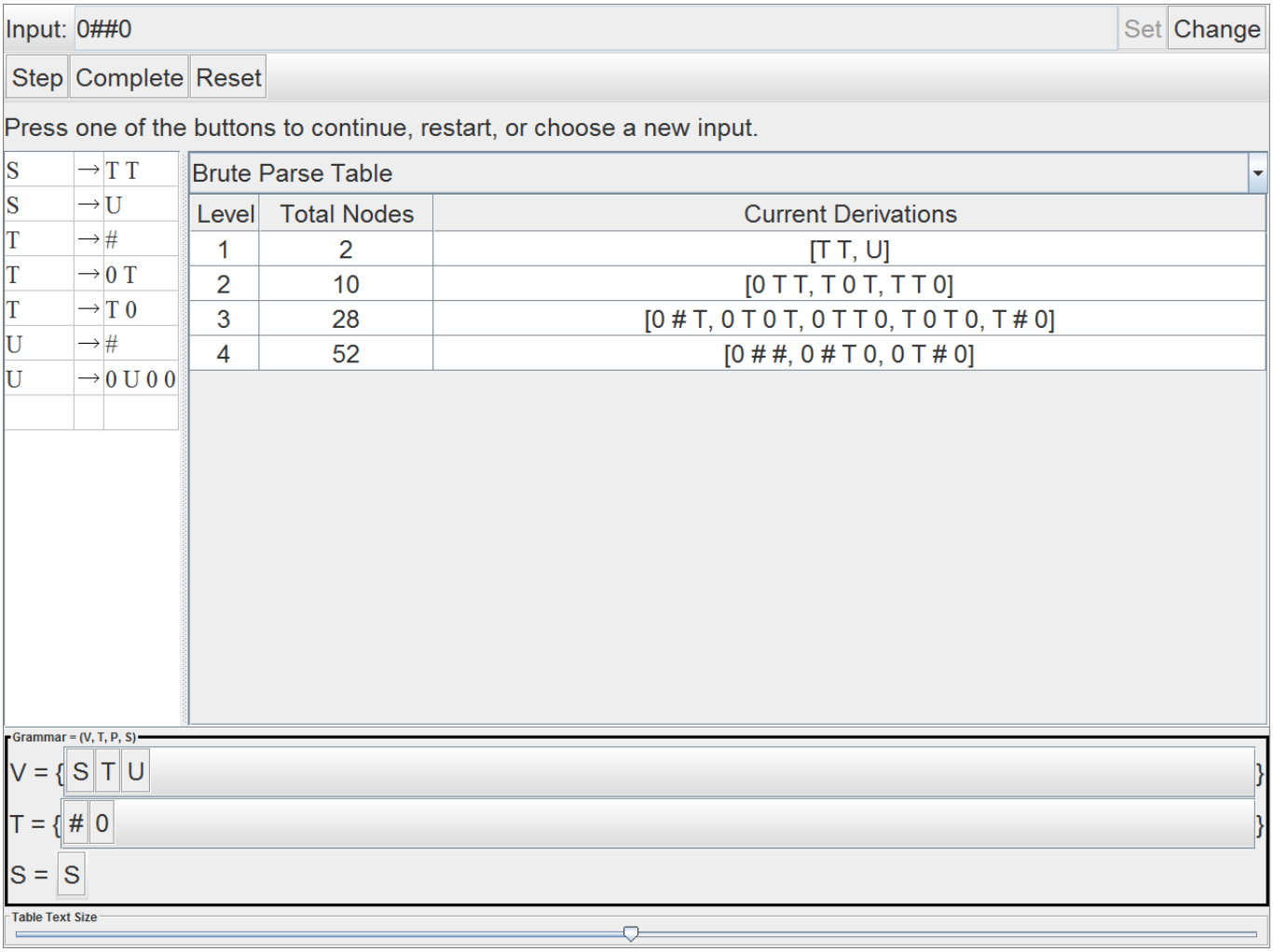
The start symbol can derive either TT or U.



At each step we use a production rule but eliminate the ones that cannot possibly produce the strings in question. For instance in this case, the rules for U can be discarded since they either produce a single # or a string with 3 0s. So the current derivations correspond to using the different rules for T. The current nodes column gives you all the nodes that are being produced, but the current derivations column does the pruning (otherwise this is very tough to maintain).

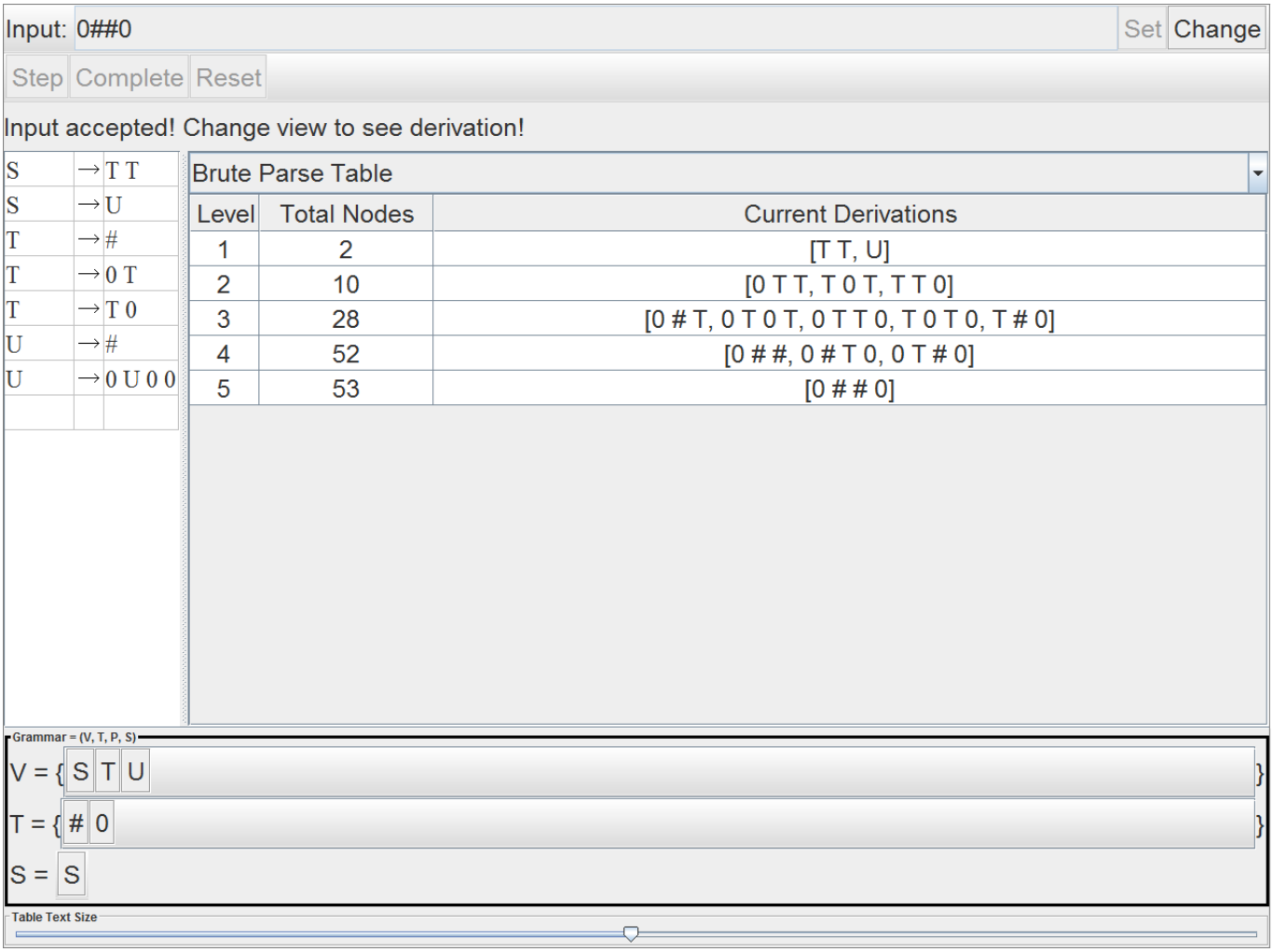

Input: 0##0
Set
Change
Step
Complete
Reset
Press one of the buttons to continue, restart, or choose a new input.
S → T T
S → U
T → #
T → 0 T
T → T 0
U → #
U → 0 U 0 0
Brute Parse Table
Level
Total Nodes
Current Derivations
1
2
[T T, U]
2
10
[0 T T, T 0 T, T T 0]
Grammar = (V, T, P, S)
V = { S T U }
T = { # 0 }
S = S
Table Text Size


Input: 0##0
Set
Change
Step
Complete
Reset
Press one of the buttons to continue, restart, or choose a new input.
S → T T
S → U
T → #
T → 0 T
T → T 0
U → #
U → 0 U 0 0
Brute Parse Table
Level
Total Nodes
Current Derivations
1
2
[T T, U]
2
10
[0 T T, T 0 T, T T 0]
3
28
[0 # T, 0 T 0 T, 0 T T 0, T 0 T 0, T # 0]
Grammar = (V, T, P, S)
V = { S T U }
T = { # 0 }
S = S
Table Text Size

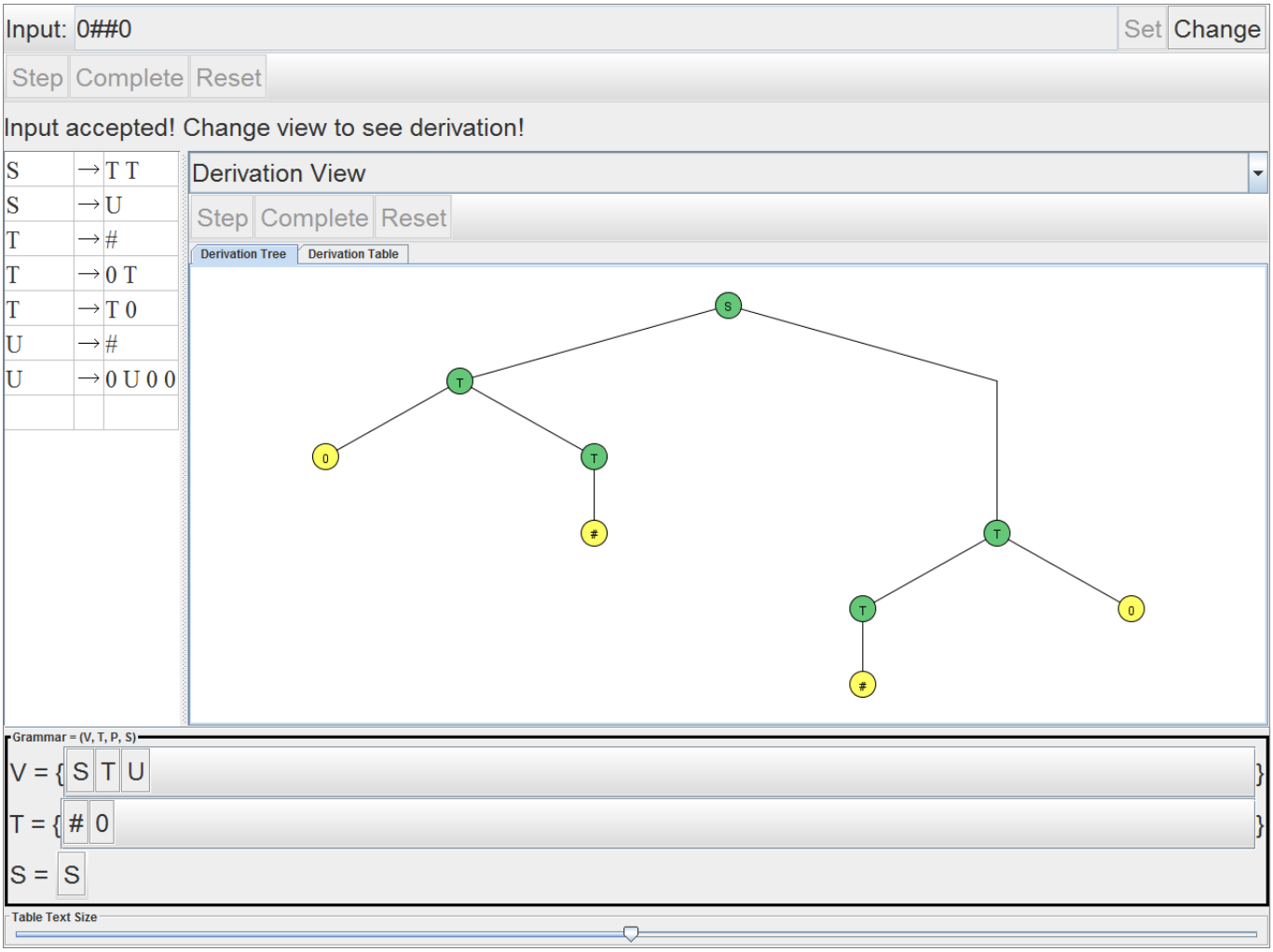


At this stage, there are only 3 possible derivations that are being retained and it is clear that the final step will have to involve the $T \to \#$ rule.

This is the complete brute parse table



The parsing can also be viewed in the format of a tree by changing the view(use the selector above the parse table) to a derivation view.

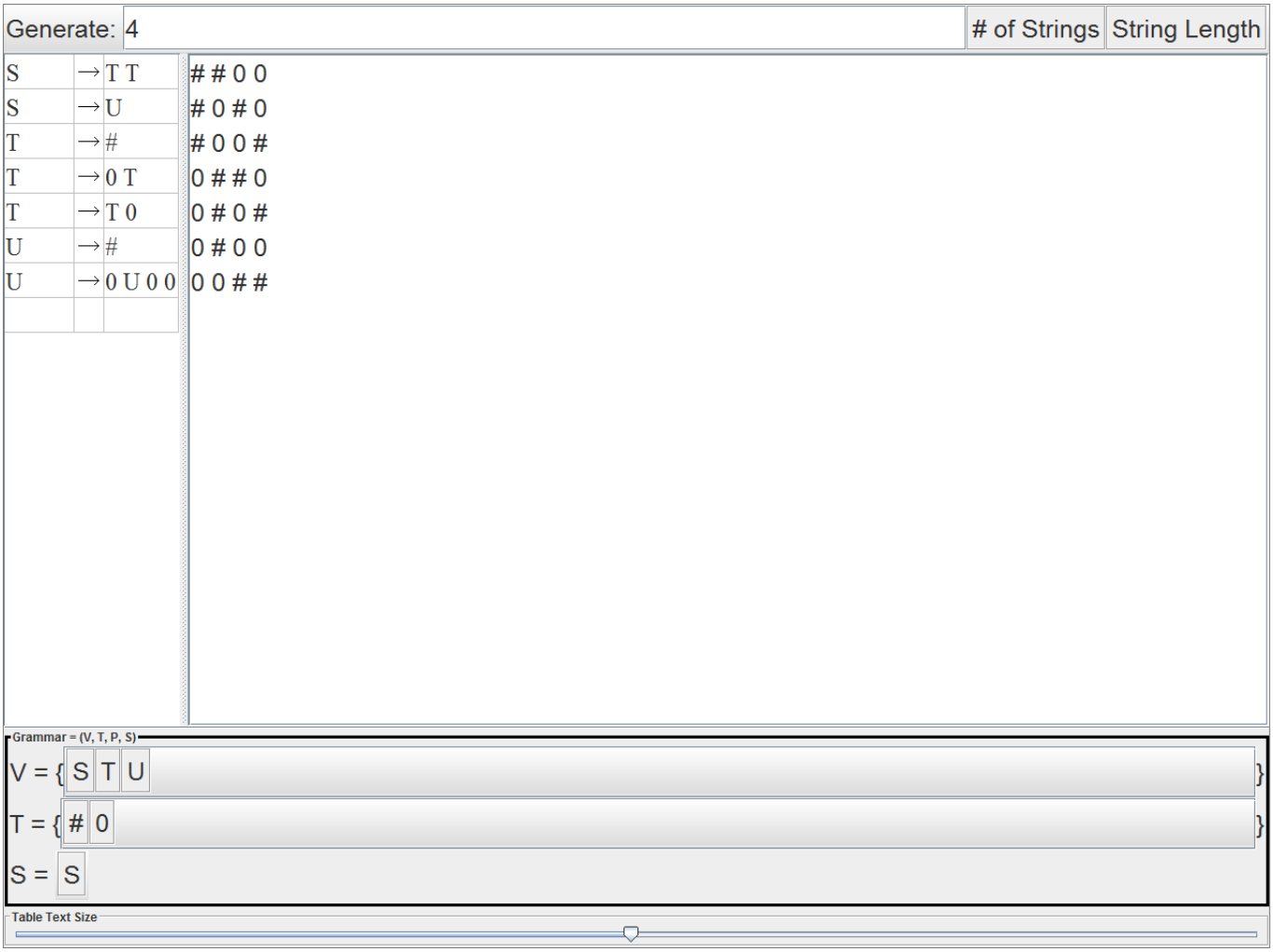


Language Generation

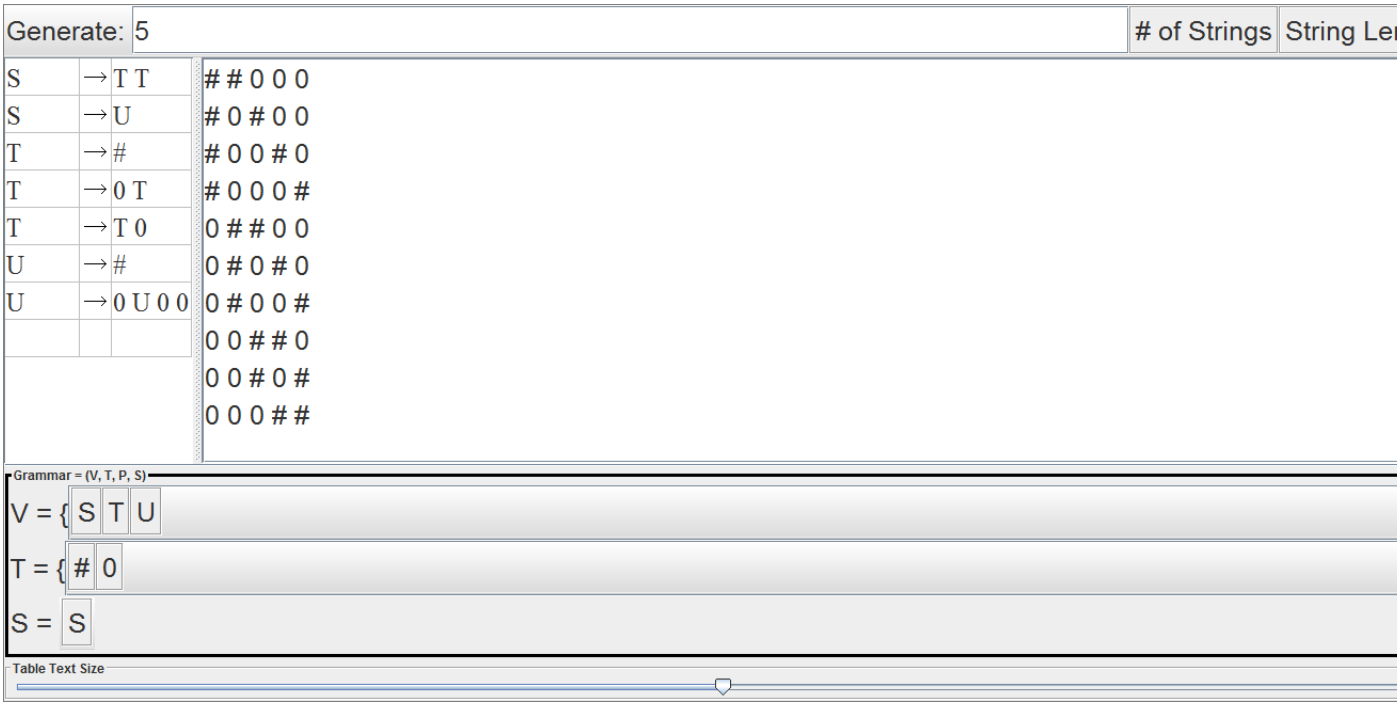
JFLAP can also provide some insight into the complete language that is generated by the grammar. By using the generate language feature you can generate strings up to a certain length.

Before using the feature, what do you think the language being generated is? What property do string being generated by these rules satisfy?

Here for instance are all strings of length 4.



And here are strings of length 5


Generate: 5
# of Strings
String Le
S → T T
S → U
T → #
T → 0 T
T → T 0
U → #
U → 0 U 0 0
0 0 0
0 # 0 0
0 0 # 0
0 0 0
0 # # 0 0
0 # 0 # 0
0 # 0 0 #
0 0 # # 0
0 0 # 0 #
0 0 0 # #
Grammar = (V, T, P, S)
V = { S T U
T = { # 0
S = S
Table Text Size