CYK parsing

1 Parsing in CFG

Given a grammar, one of the most common applications is to see whether or not a given string can be generated by the grammar. This task is called Parsing.

It should be easy to see that unless a grammar is trivial, the naive method of trying to track down every single production path from the start symbol is going to take too much time. Hence we need to look for more efficient algorithms.

2 Chomsky Normal Form

Every context free grammar can be converted to Chomsky Normal form where the only rules are of the form

$A \to a$ - that is a single variable producing a single terminal

$A \to AB$ - that is a single variable producing two variables.

If an empty string is part of the grammar, the only rule that is allowed is that of the start symbol producing the empty string. $S \to \epsilon$.

3 CYK parsing main idea

Given a string w of length n, CYK parsing works by filling up a triangular table with n rows and n columns.

The first row will have n columns, the second $n-1$ and so on until the n^{th} row has just 1 column.

The idea is that in each cell of the table, the algorithm will find the variables in the grammar that can generate a substring of w beginning from a certain character in w . For instance for the sentence 'I love automata theory' the table of substrings looks like the following

I	love	automata	theory
I love	love automata	automata theory	
I love automata	love automata theory		
I love automata theory			

Now the idea of the algorithm is fill the entries of the table with variables that produce these substrings. So for instance, the $(1,1)^{\text{th}}$ entry is filled with any variable that produces 'I'. The $(3,2)^{\text{th}}$ entry is filled with any variable that produces the string 'love automata theory'.

Finally note that the bottom corner gives you all the variables that produce the entire input string. If the start symbol of the grammar DOES produce the entire string then the

start symbol will be found in this bottom corner. That would then be an indication that the input string is in the grammar.

The table is filled up using a dynamic programming technique. The solution is being built from the bottom up where first we find variables that produce substrings of length 1, then use that information to find variables that produce substrings of length 2 and so on and so forth until we eventually use the information in the table to figure out which variables can produce the entire string.

These steps will be illustrated in the next two sections using examples. First an example with a grammar that uses single character terminals and then a grammar whose terminals correspond to english language words.

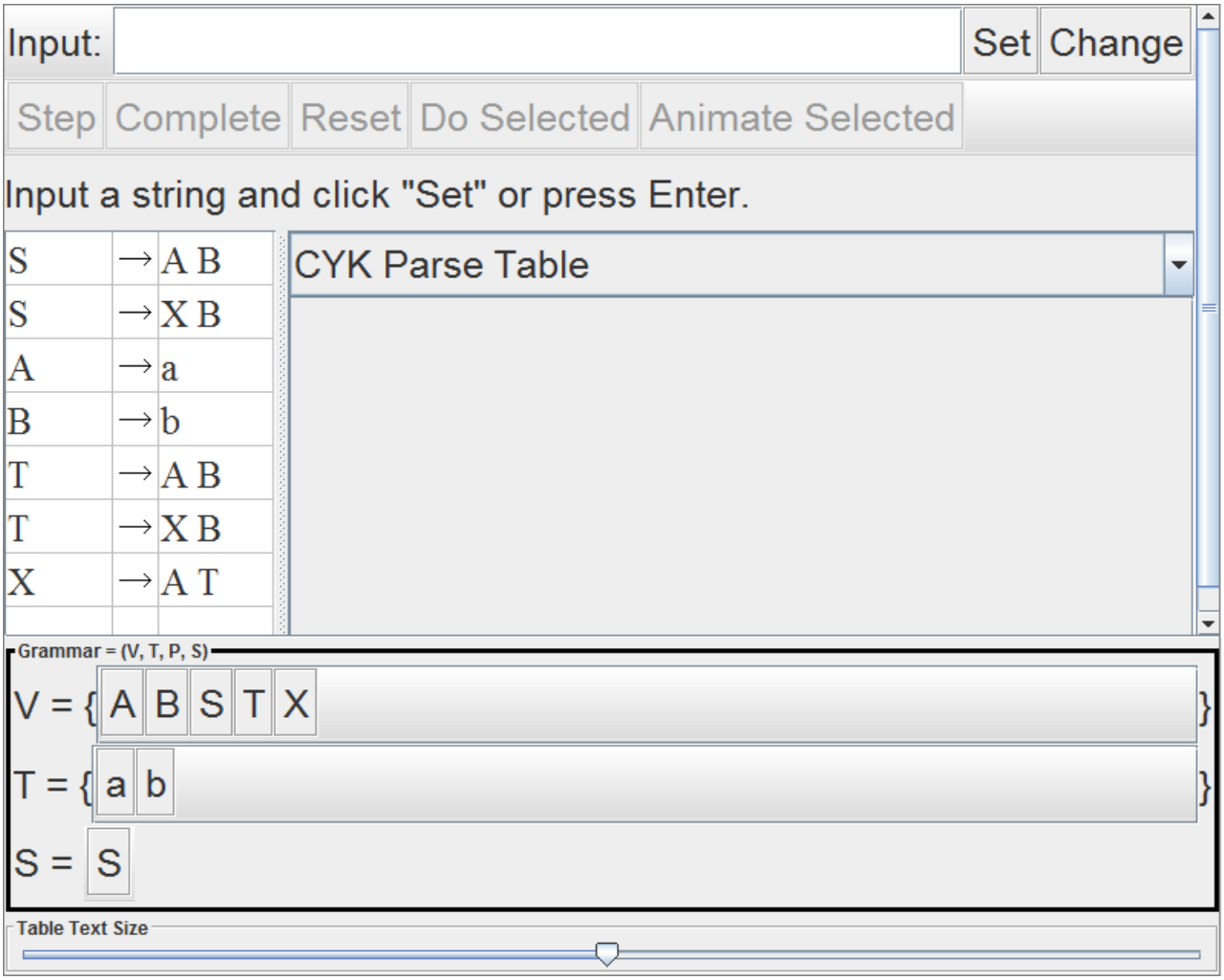
4 CYK parsing example (using JFLAP)

We will work with the following example

$$
\begin{aligned}
S &\to AB \\
S &\to XB \\
T &\to AB \\
T &\to XB \\
X &\to AT \\
A &\to a \\
B &\to b
\end{aligned}
$$

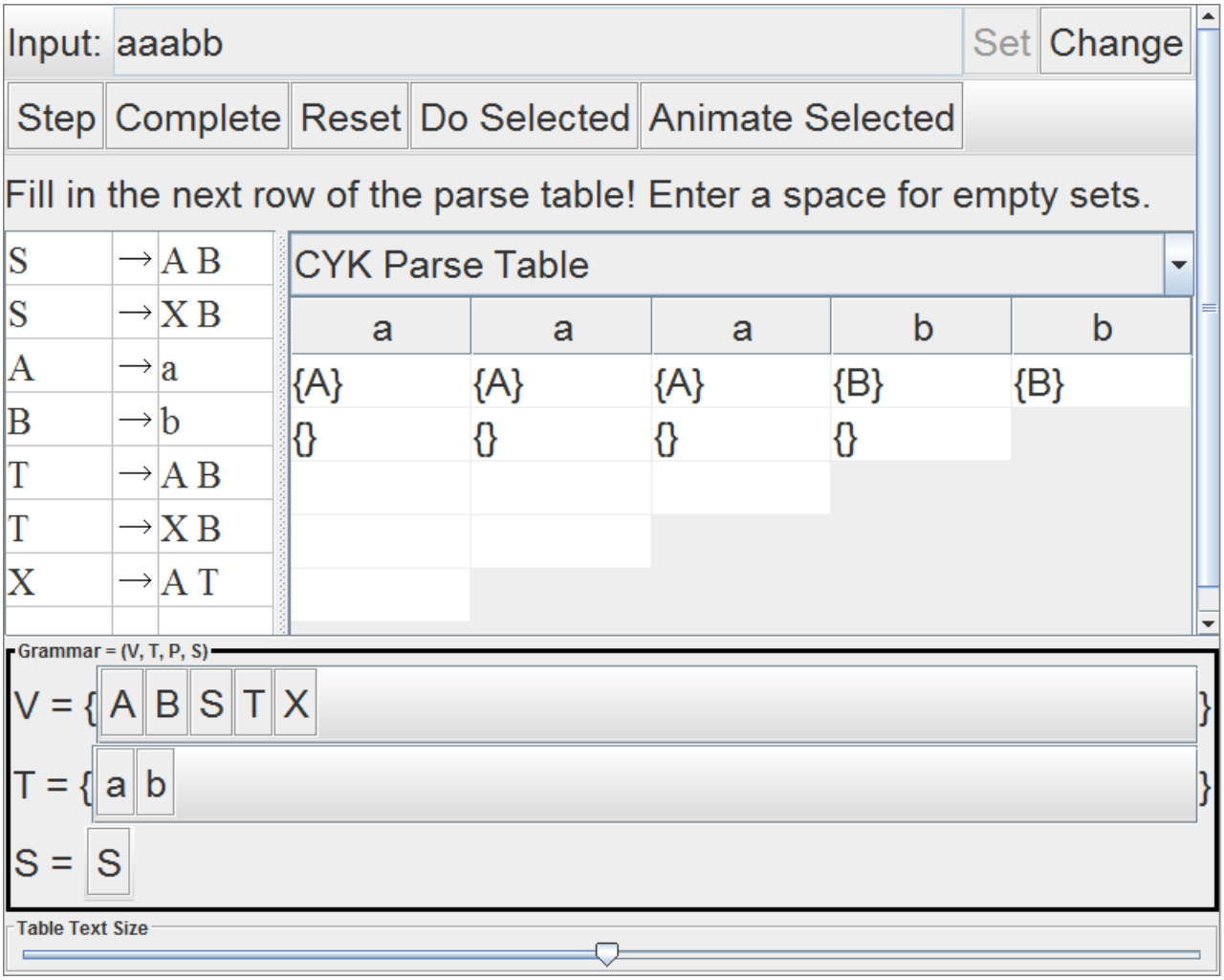
Enter the grammar in JFLAP and save the file. Alternatively load up the file CYKModule1.jflap.

To begin the parsing, click Input and CYK Parse as the option. This will bring up a display that looks like the following.

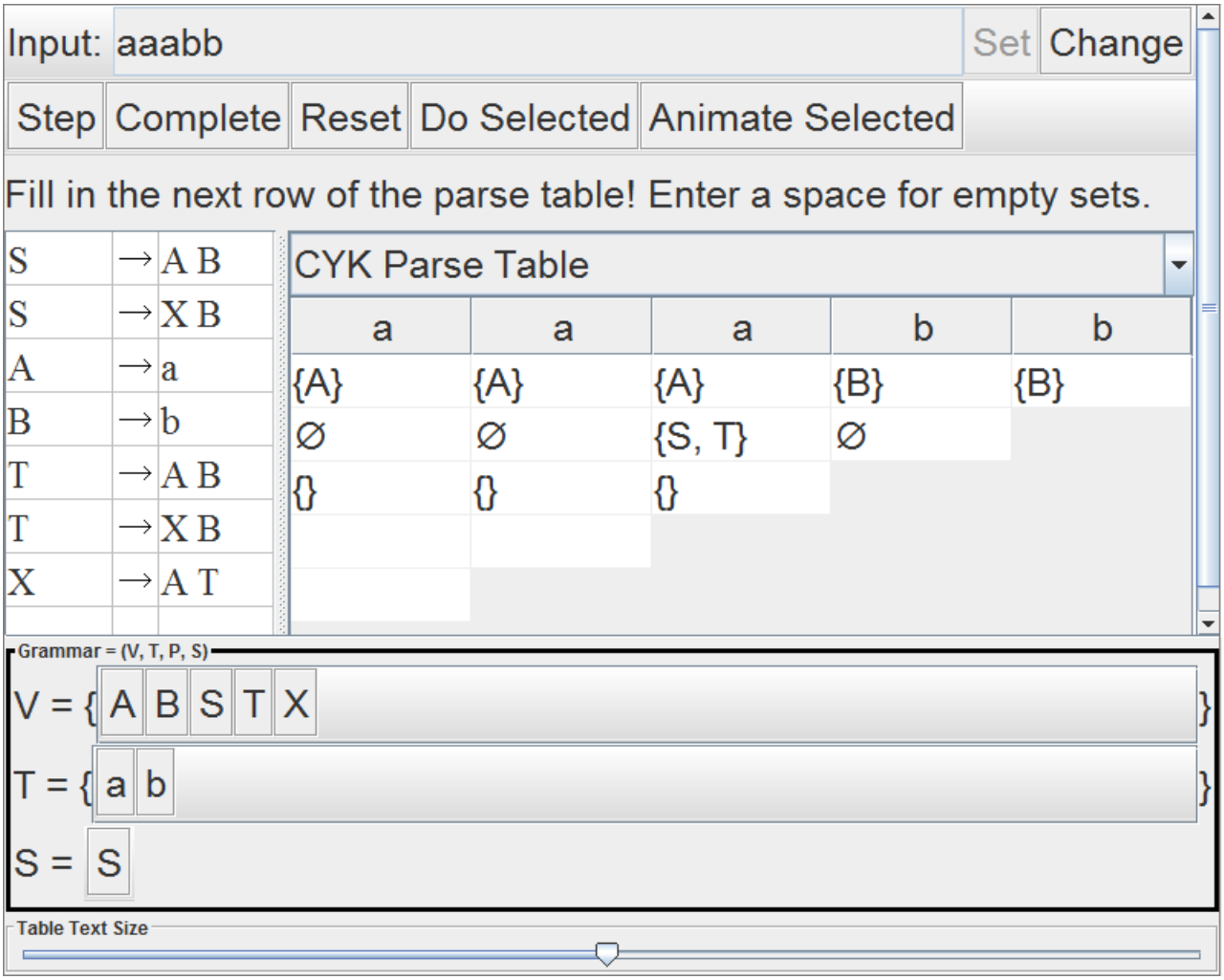


Enter the string *aaabb* as the input string and click Set.

The algorithm works on the principle of dynamic programming. The idea is to build the string bit by bit and using a table to keep track of information. The first step is to see which variables can produce substrings of length 1. The substrings of length 1 are just the individual characters. $A \to a$ and $B \to b$ are the rules to be used and the variables that can produce the respective substring are placed in the first row.



The next step is to see which variables can generate the different substrings of length 2. The first row already tells us the variables that generate strings of length 1. We can use this information. For the leftmost spot (the $(2,1)^{\text{th}}$) in the second row, we are looking for a variable that generates the combination AA. There is no such variable so that remains empty. The only two variable combination (from the first row) that can possibly be generated is the one that is AB. By going through the rules we see that there is one rule $S \to AB$ and $T \to AB$. So the set of variables that can generate the two variable sequence AB is $\{S, T\}$. That is placed in the table as shown below

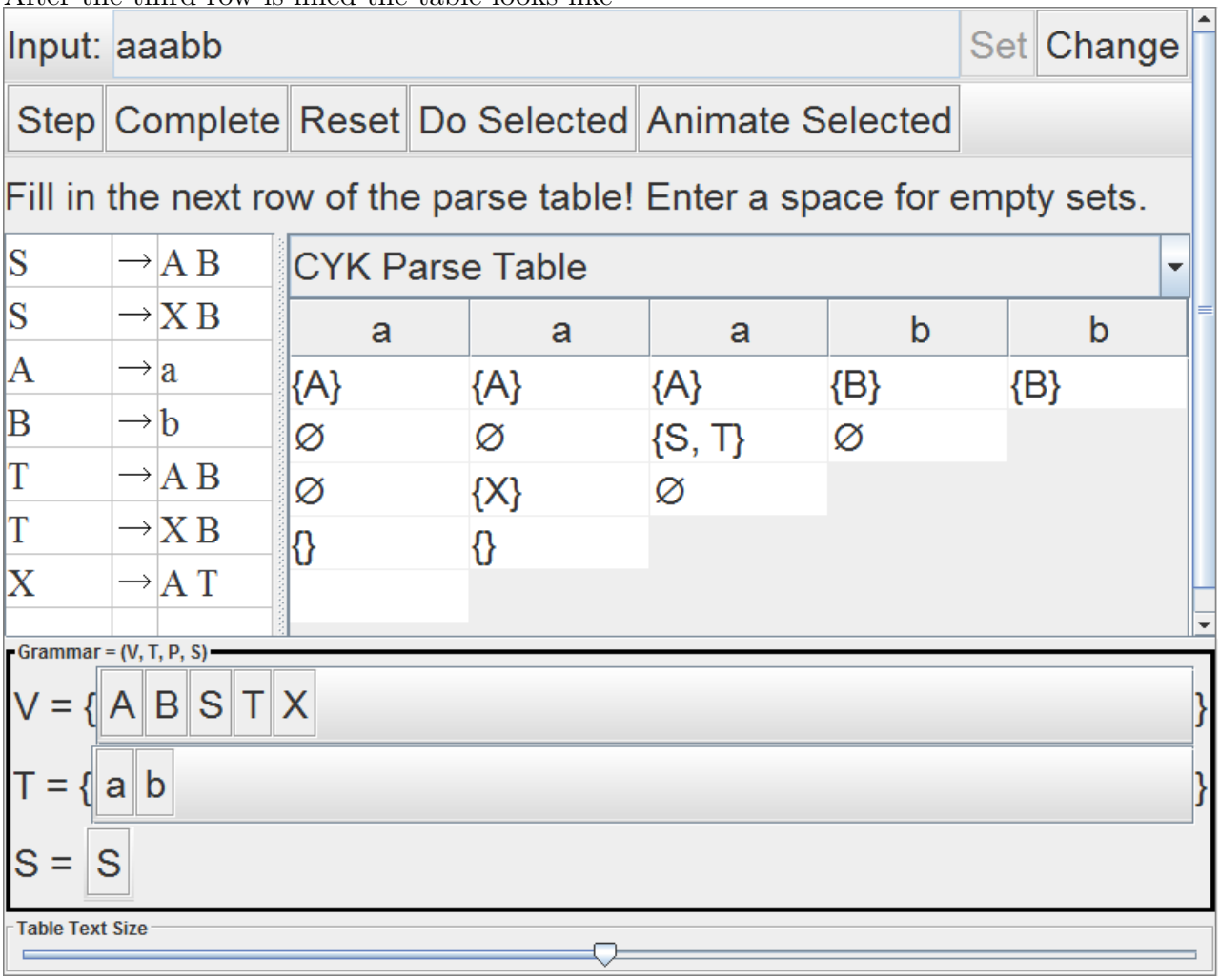


For the next step, we are looking for variables that can produce substrings of length 3. Since the rules are in Chomsky Normal Form, this can only happen if the variable can produce two variables. These two variables then satisfy the property that either one of them gives the first character of the substring and the other gives can produce the remaining 2 characters or the first two characters of the substring are produced by the first variable and the last character can be produced by the second variable.

To get the information required to populate the third row therefore, all we need is to check the rules and the information that we have stored in the first and second rows. The $(3,1)^{\text{th}}$ is going to be empty because there is no rule that produces AA. The $(3,2))^{\text{th}}$ entry is filled by any variable that produces either AS or AT. While there is nothing that produces AS, there is $X \to AT$. This means X is the only variable that is put into the table for that entry.

Finally for the last entry in this row, we are looking for rules where either SB or TB appears on the right side. There are no such rules in our grammar, so that entry remains empty as well.

After the third row is filled the table looks like



In the next step we find variables that generate substrings of the original string that are of length 4. A substring of length 4 can be generated by

- having one variable produce the first character (substring of length 1), and the second variable producing the next 3 characters. This is the 1 +3 combination. In the case of the $(4, 1)$ entry it would mean looking for a rule that generates AX
- a variable producing a substring of length 2, and the second variable producing the remaining 2 characters. This is the 2 + 2 combination. In that first position, there is no variable that generates the first 2 characters of the substring aa, so this is not possible.
- a variable producing the initial substring of length 3, followed by a variable that can produce the last character1. This is the 3 + 1 combination. Again, in row 3 we see an empty entry for the first column. So again, there is no possible variable that we can put down in the $(4, 1)^{\text{th}}$ spot.

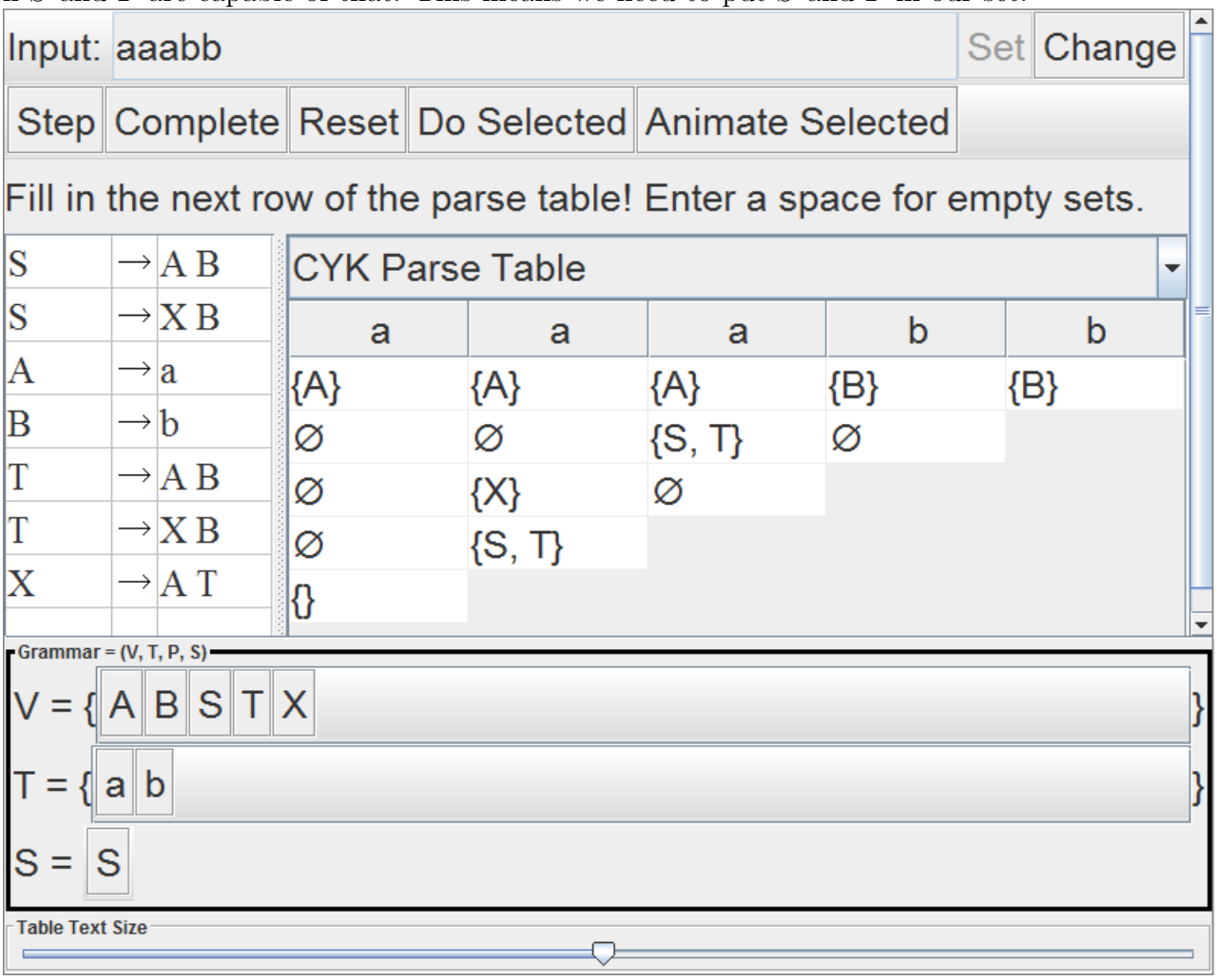
This therefore means we get an empty set in the first entry of the fourth row.

Now for the second entry in this row, by applying the same reasoning.

The 1 + 3 combination does not exist

The 2 + 2 combination also does not exist

The 3 + 1 combination is asking for all those variables that generate XB. Note that both S and T are capable of that. This means we need to put S and T in our set.



The final step is to see which variables can generate a substring of length 5. There is only 1 substring of length 5 - the entire string. Again, by looking at the various splits we see that we have to examine the following

The 1 + 4 combination which is a rule that generates AS or AT. Clearly X is a variable that does this.

The 2 + 3 combination. The empty set in the first entry of row 2 rules out this option.

The 3 + 2 combination. The empty set in the first entry of row 3 rules out this option.

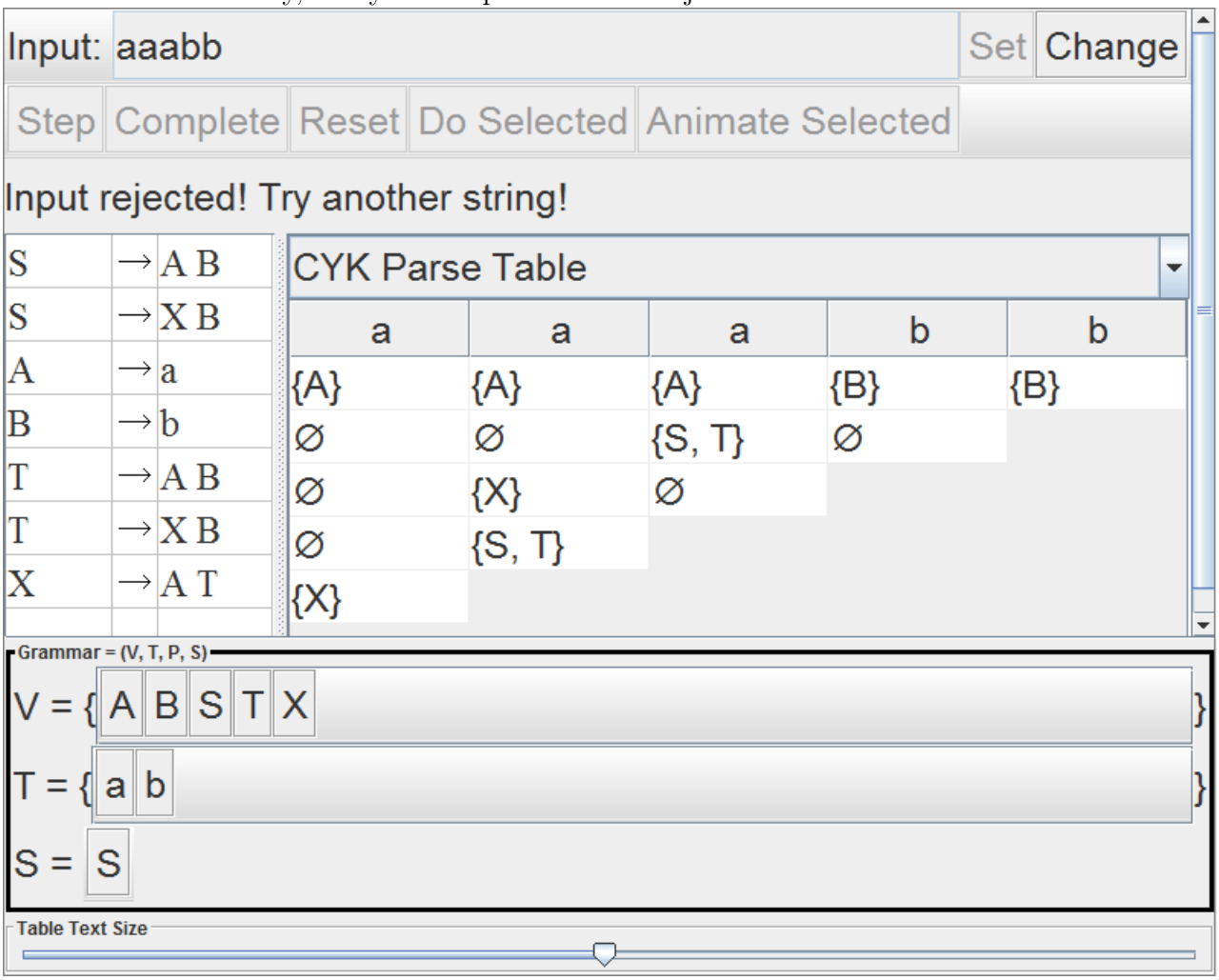
The 4 + 1 combination. Again, the empty set in row 4 rules out this option.

That means that the only variable in row 5 is X.

Once the entire table is filled in this manner, the question still remains, does the grammar generate the string that was provided as input? It should be clear that as long as the final set has the starting varible S as an element, the input string can be generated. Else it cannot.

In this particular case, the only variable that can generate the string is X. But X is not the start variable. So the string *aaabb* cannot be generated by this grammar.

As you can see in the figure, JFLAP fills that final entry in the table and realizing that S is not in this final entry, it says the input has been rejected



5 An example with an English sentence

As you have probably learnt, grammars can actually be used for analyzing languages. Here for instance is an example taken from wikipedia - CYK parsing in wikipedia.

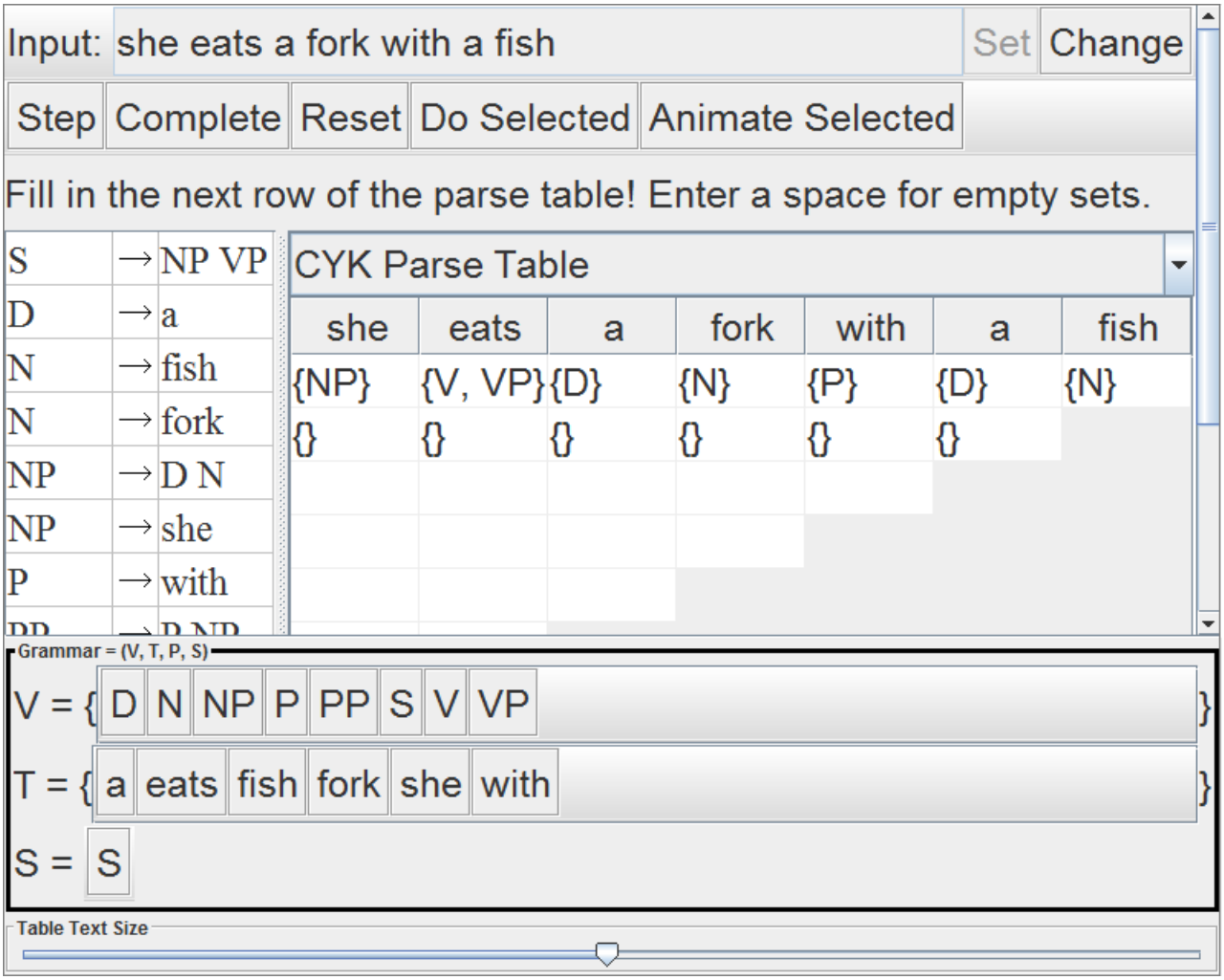
To enter this grammar note that you have to first set your preferences to allow multiple characters. This can be done via the help menu. Select preferences and select the default multiple character input.

Then you should be able to enter the grammar

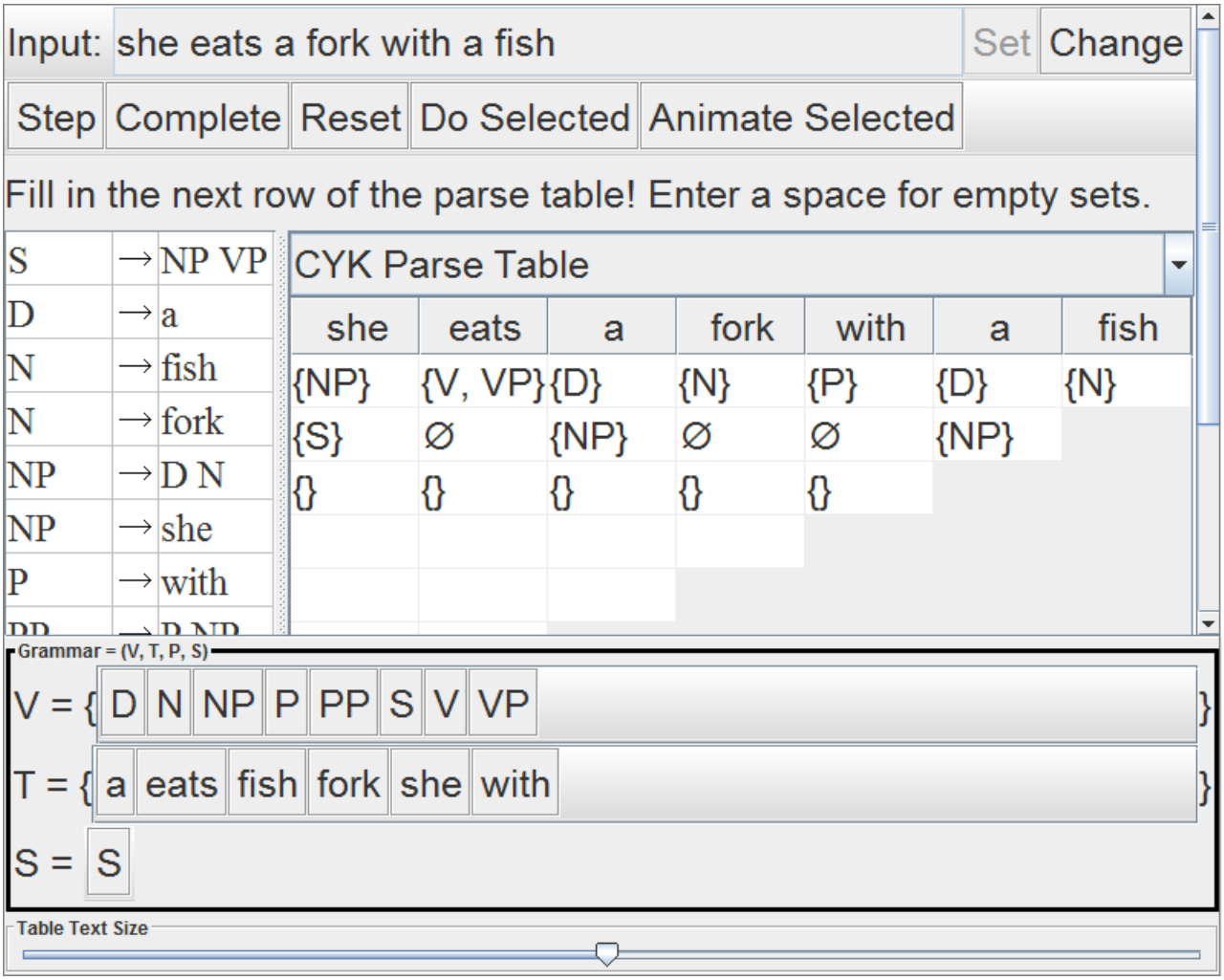
$$
\begin{aligned}
S &\to NP\,VP \\
VP &\to VP\,PP \\
VP &\to \text{eats} \\
PP &\to P\,NP \\
NP &\to D\,N \\
NP &\to \text{she} \\
V &\to \text{eats} \\
P &\to \text{with} \\
N &\to \text{fish} \\
N &\to \text{fork} \\
D &\to \text{a}
\end{aligned}
$$

Now just like the wikipedia example (except we will use JFLAP) to parse the sentence 'she eats a fork with a fish'.

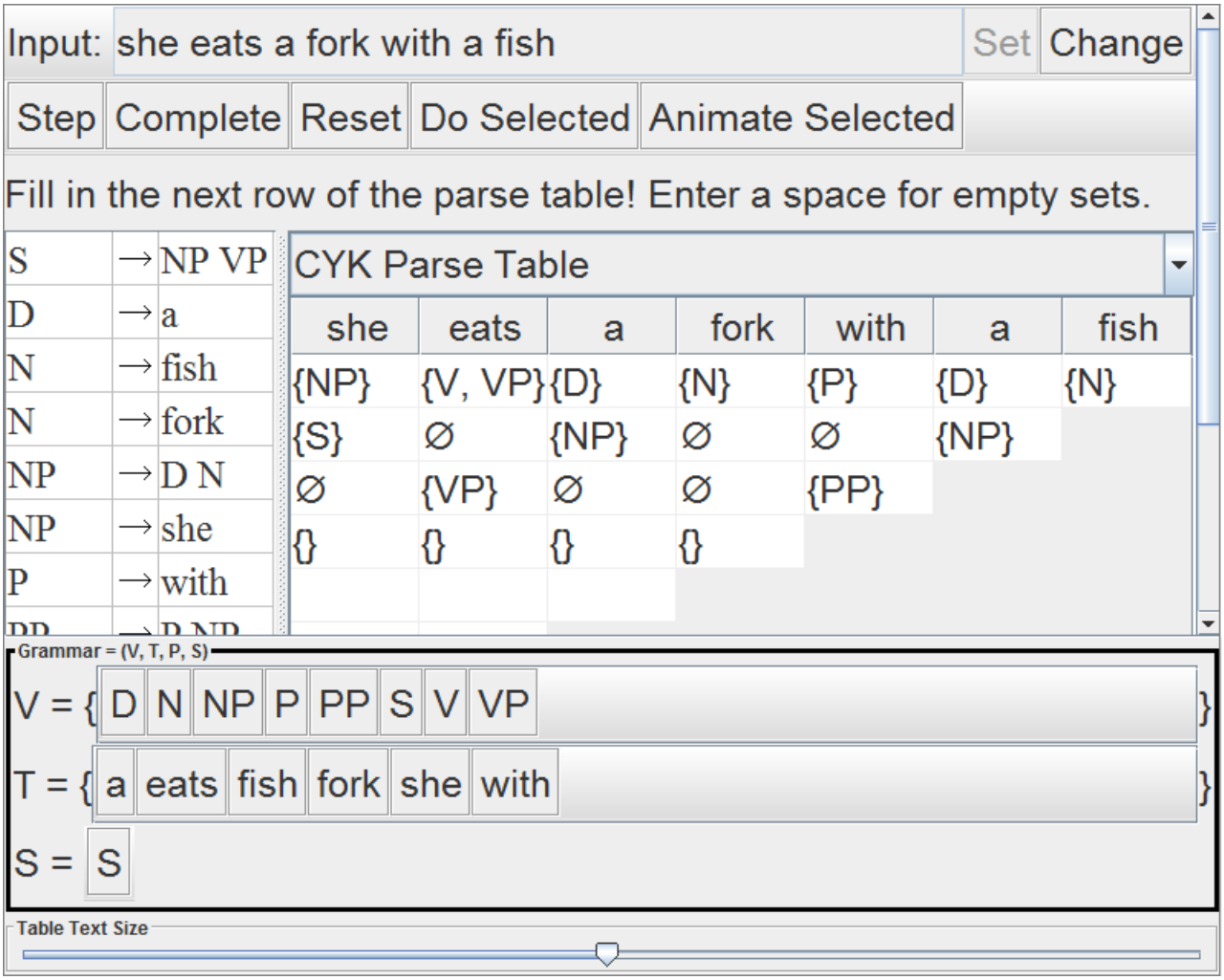
Again the first step is to see which of the variables produce the individual terminals, which in this case are the words in the sentence.



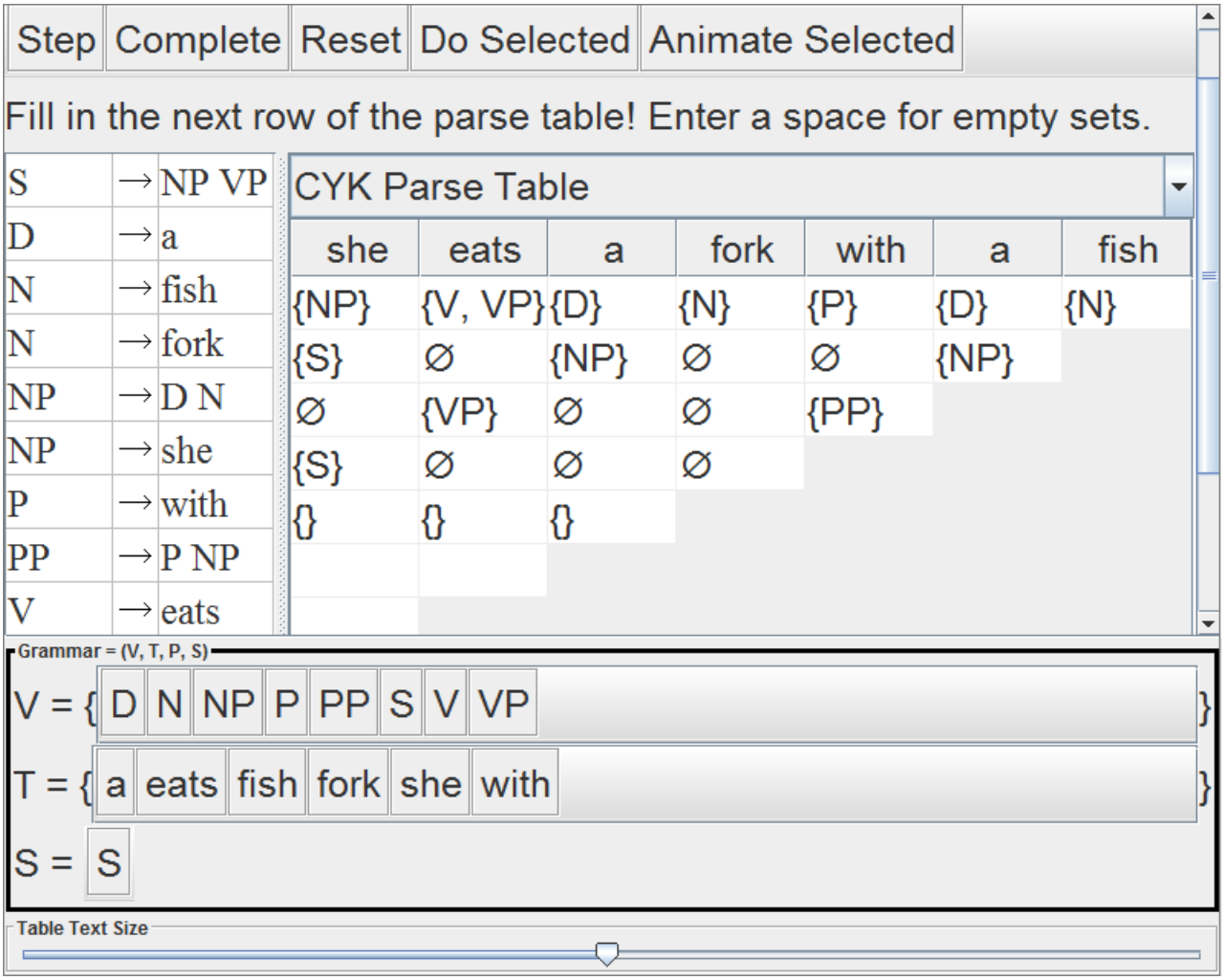
The next step is to see which variables can give you a combination of two consecutive entries that you find in row number 1. For instance, for the first entry in this row, we are looking for rules which have either $NPVP$ on the right side or NPV. The very first rule of the grammar gives you the information that S is one such variable. Then by looking through all the other rules we find this is the only such variable. Continuing this process for all entries in this row we get the following



Next to find variables that can produce 3 consecutive words. This would mean looking for variables that have rules that produce a variable in the first row followed by the next variable in the second row or the other way round. For instance, to fill the $(3,2)^{\text{th}}$ entry over here we should be looking for rules that have $V\ NP$ or $VP\ NP$ or $NP\ P$. We see the rule that says $VP \to V\ NP$ and that is the only variable that gives us one of these rules. Hence the entry is just $\{VP\}$.



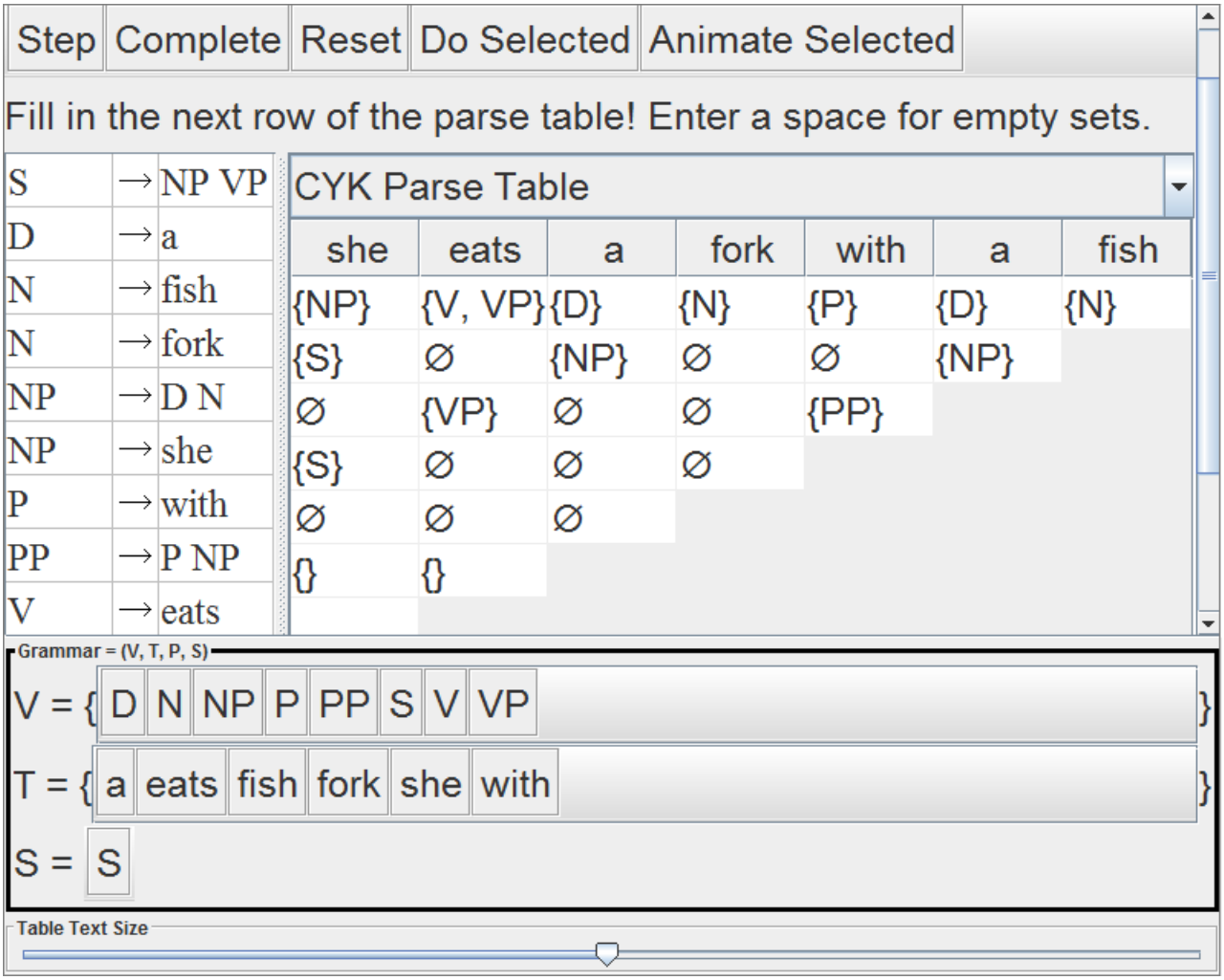
In the next step we have more possible combinations again. $4 = 3 + 1$ or $2 + 2$ or $1 + 3$. So for the first entry we are looking for rules of the form $S\ NP$ (corresponding to the $2 + 2$) or $NP\ VP$ (corresponds to the $1 + 3$). We find only S.



String of length 5 can be broken down into 2 pieces in the following manners - as 4 + 1 combination or 3 + 2 combination or a 2 + 3 combination and finally a 1 + 4 combination.

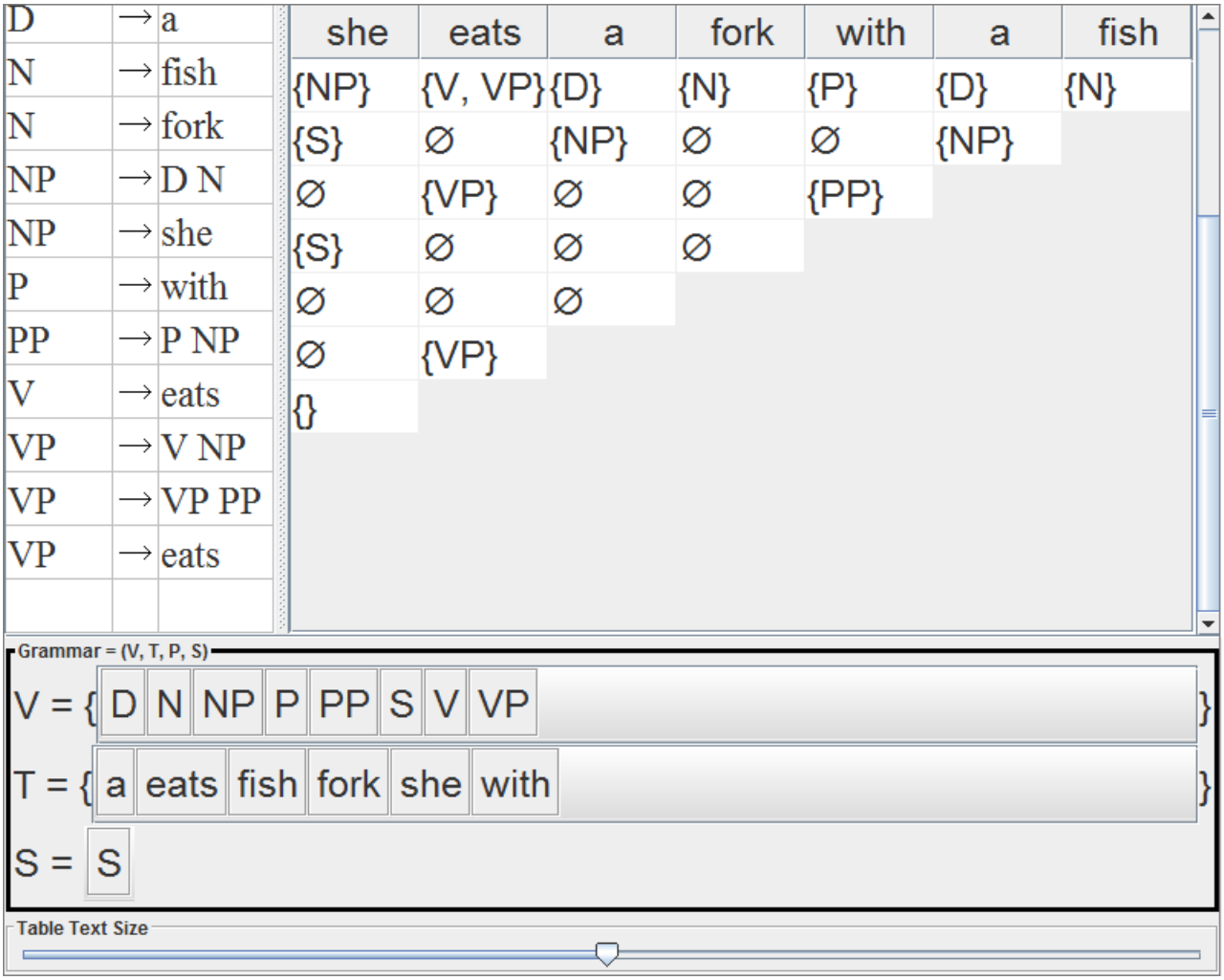
In the case of the first entry this would mean looking for $S\,P$ (the 4 and 1 combination), the 3 and 2 combination is ruled out because of the empty set in the first entry in the 3rd row, the 2 and 3 combination gets ruled out because while S can produce the first two words there is nothing can produce the 3 words right after that (the empty set in the 3,3 position), finally the 1 and 4 combination can also be ruled out because of an empty set in the 4,2 position.

Try it - Work out the other two entries in this row and convince yourself that the entire row is empty.



For the next row, strings of length 6 can be broken into a 5 + 1 combination, 4 + 2, 3 + 3 , 2 + 4 and finally 1 + 5. Since the previous row was completely empty in this case we can rule out the 5 + 1 combination and the 1 + 5.

For the $(6, 2)$ entry we will look for $VP\ PP$ (from the 3 and 3 combination). Note that none of the other possibilities exist in this case. We see that VP is a variable that produces $VP\ PP$. Hence this is table at the end of this step



Finally to produce the entire string, we just have one entry to fill. The string has length 7 so the following two way splits need to be checked.

6 + 1 - not possible because of empty entry in the 6,1 position

5 + 2 - entire row 5 is empty

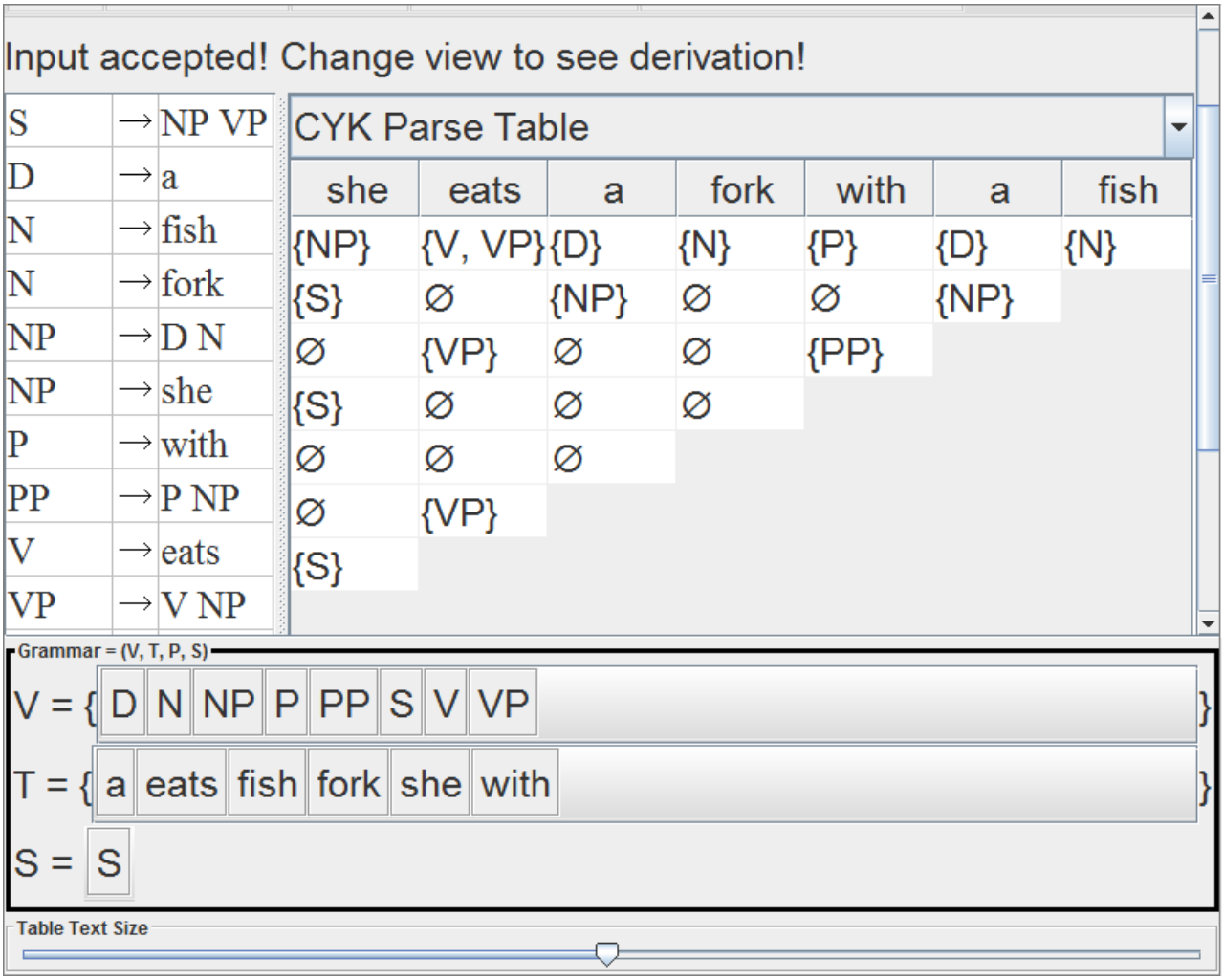
4 + 3 - we are looking for a rule that says SPP on the right side. does not exist

3 + 4 - not possible because of the empty entry in the 3,1 position

2 + 5 - entire row 5 empty. Rule out this possibility

1 + 6 - looking for $NPVP$ and we find a rule! $S \rightarrow NPVP$.

So the final table looks like this



Since the start symbol can be found in the last entry, this sentence can be generated by this grammar.

6 Further questions

1. Work out the first example with *aabb*. First convince yourself that this actually can be generated by the grammar and then use JFLAP to step you through that process.

2. In the wikipedia example, check if the sentence 'a fork eats a fish' can be produced by the grammar. Remember, a grammar is not responsible for the meaning of a sentence.

3. What happens if the grammar is not in Chomsky Normal form? Can JFLAP detect this? What happens if you try and use CYK parsing without first converting over to CNF.

4. JFLAP allows you to fill in the cells yourself. What happens when you click a cell?

What happens if you enter an incorrect variable in a cell?