CFG Pumping Lemma

We use one of the provided examples in JFLAP to explain the context free pumping lemma. Remember that to show that a language is not context free using the contrapositive argument of the pumping lemma, you have to show the following.

Regardless of the value of m chosen, there exists some string w in the provided language of length greater than m such that there is no way that the string w can be decomposed into five parts $w = uvxyz$ and satisfy the following 3 conditions

1. $|vy| > 0$
2. $|vxy| \leq m$
3. uv^ixy^iz is in the language for all $i \geq 0$.

JFLAP treats the showing of a language to not be regular in a manner similar to adversarial arguments. That is, the user is given the chance to pick a pumping length and the computer will show why that will not work by first producing a string and then showing the user how regardless of which way they go about making their partition into u, v and x, y and z.

The chosen example is $L = \{a^ib^jc^k : i > j, i > k\}$

Solution

As with any proof involving showing a language to not be regular using the pumping lemma, assume the language is regular and has a pumping length m.

The next step is to come up with a string that cannot be decomposed in accordance with the requirements of the pumping lemma.

In this particular case, since we have to have the block of as be greater than the block of bs and cs, a natural choice is to make the number of as be just one greater than the block of bs and cs. Also, let the bs and cs have the same length.

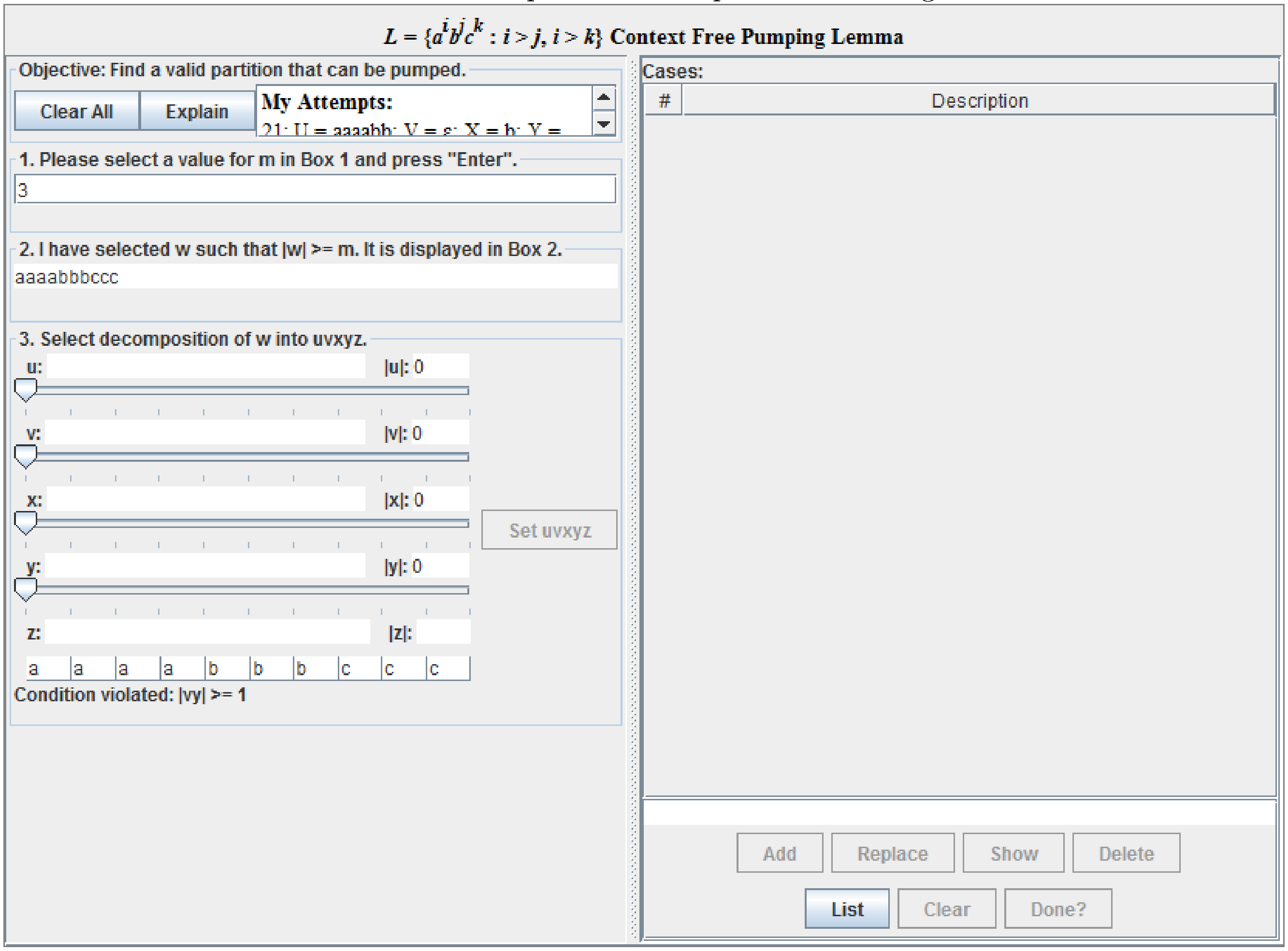
That is, pick the string to be $a^{m+1}b^mc^m$.

If you click 'Explain' in JFLAP you get an explanation of the solution, which we present here, with some slight edits.

Unfortunately no valid partition of w exists. For any m value, a possible value for w is "$a^{m+1}b^mc^m$". The v and y values together thus would have a maximum of two unique letters. Any possible v or y values would then be problematic if i = 0, i = 2, or perhaps both. Thus, this language is not context-free.

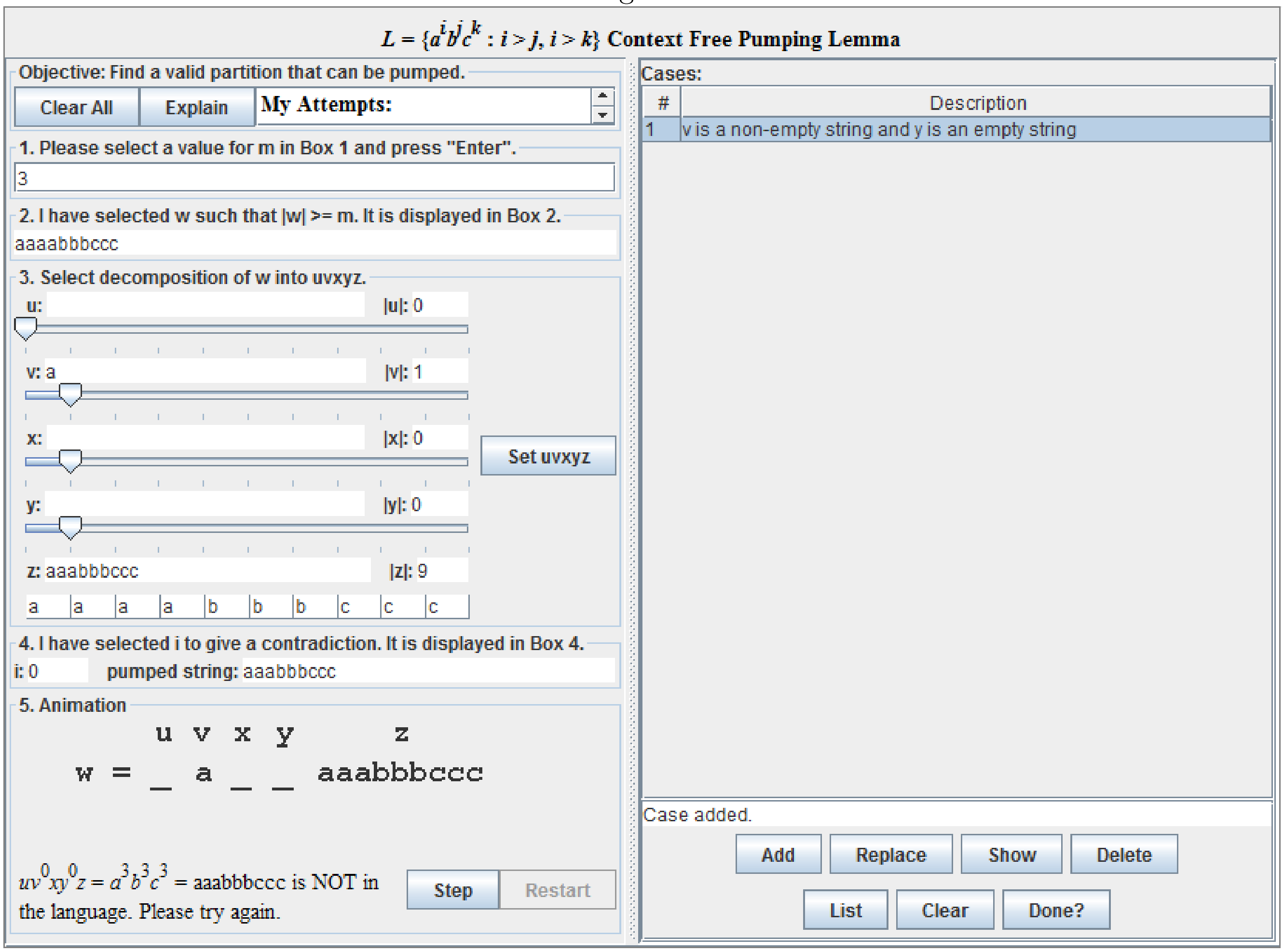
To help with the understanding of this proccess, JFLAP allows you to pick whether you or the computer makes the first move.

If you pick 'you go first' in this case, you are allowed to enter a value of m. In this case, say we enter a value of 3 and then the computer comes up with the string *aaaabbbccc*.

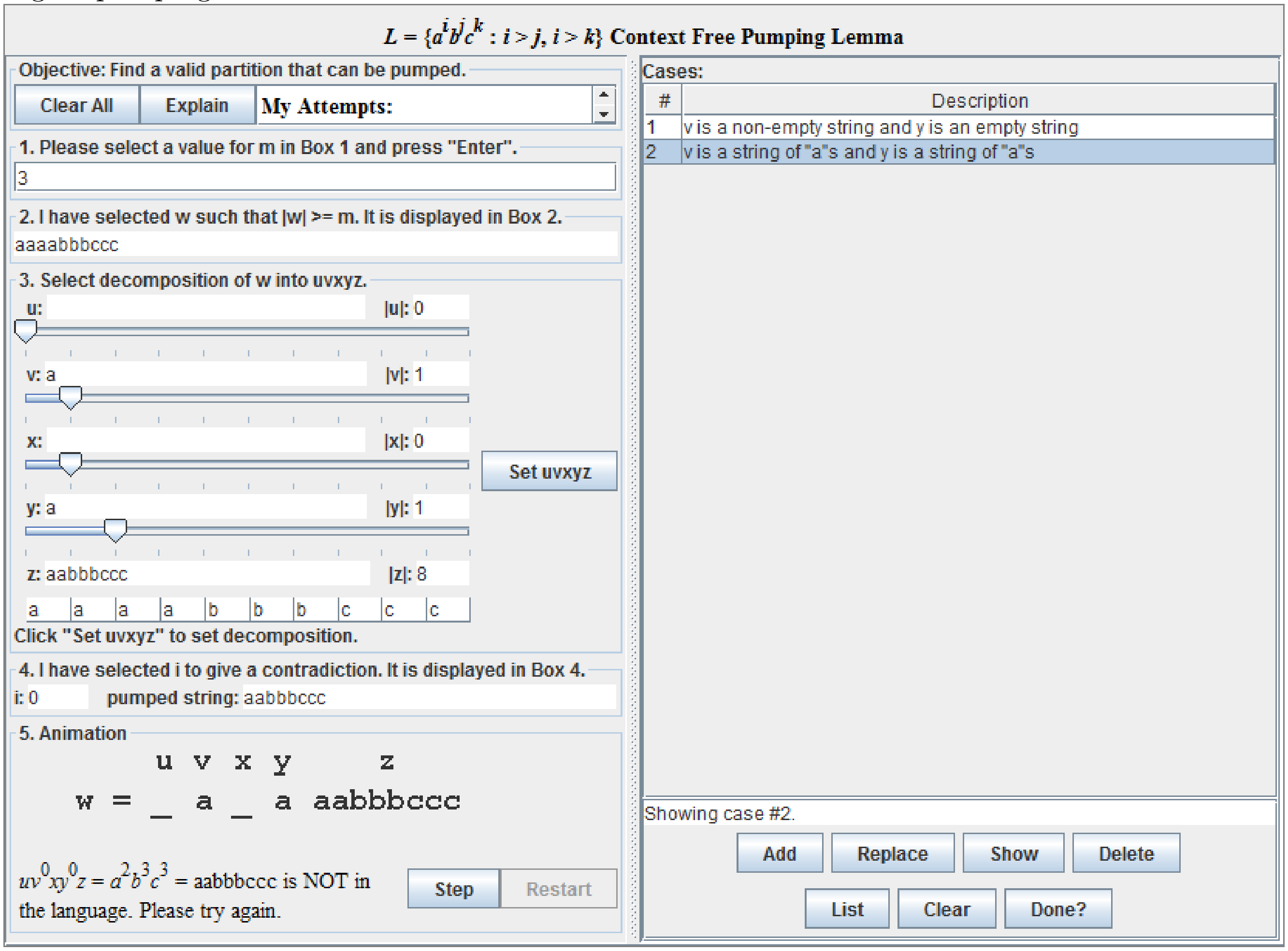


Upon being presented with the decomposition screen, we have to split this into various cases. Note that if we ever repeat a case, or if we have a case that is similar to one that has already been covered, JFLAP will be able to detect that.

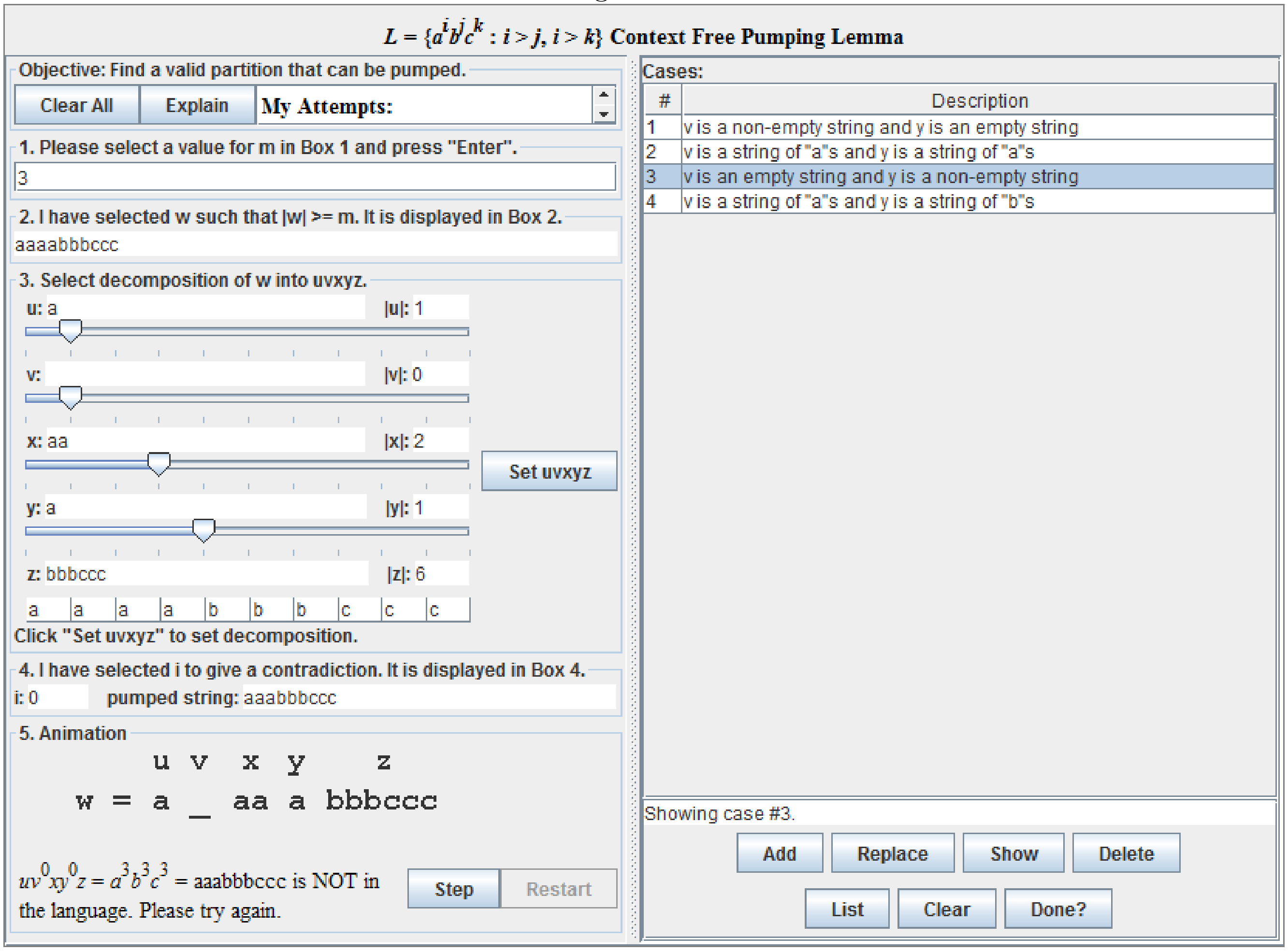
The first case would be to have u be empty, v be a some portion of the as and y be the string. It is easy to see that pumping down will result in a string that is not in the language, since it will lessen the number of as while retaining the same number of bs and cs.



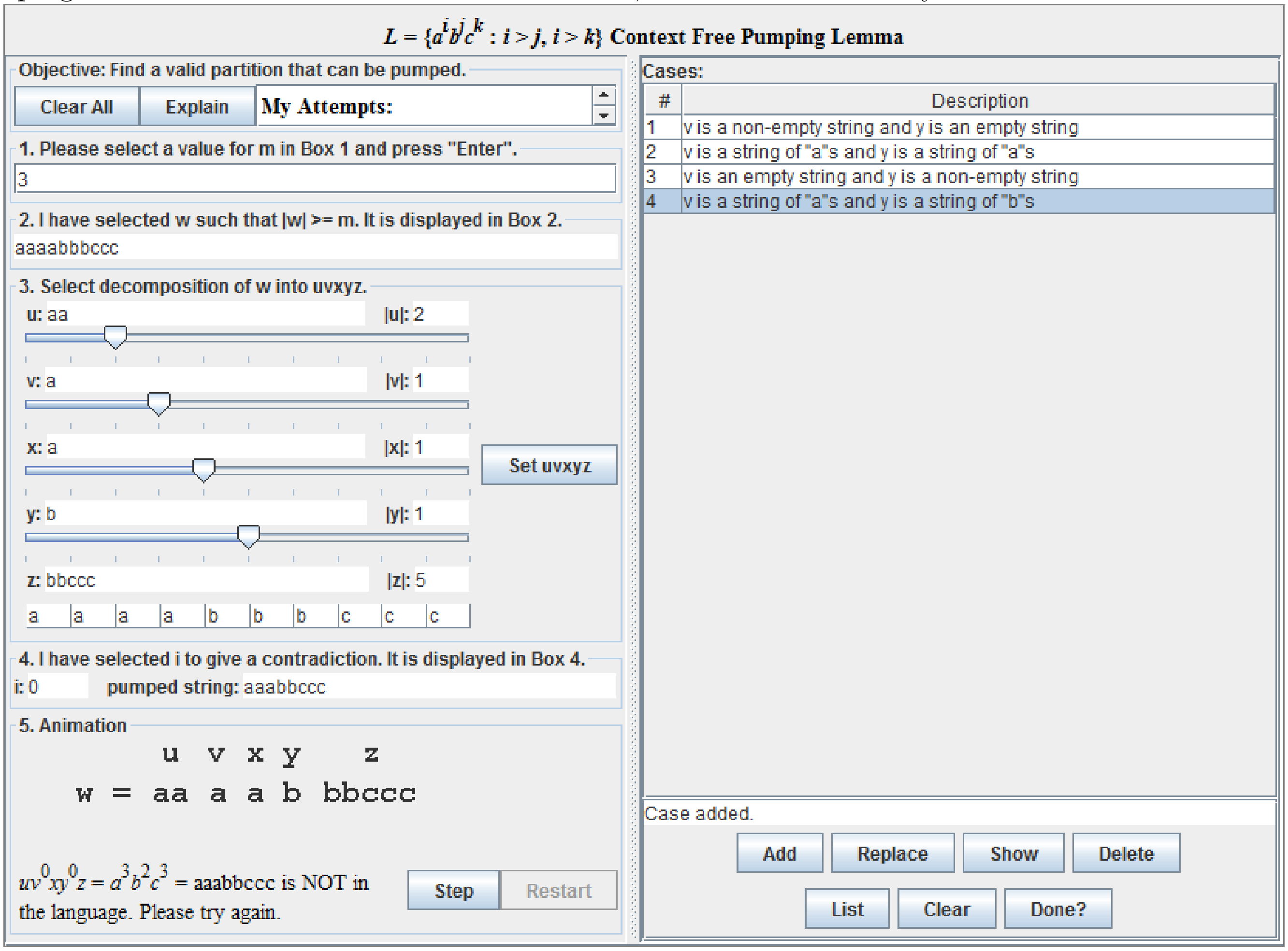
Now we slowly start moving v across the string. That is to say, make y also have some as. Again pumping down will work as shown here.



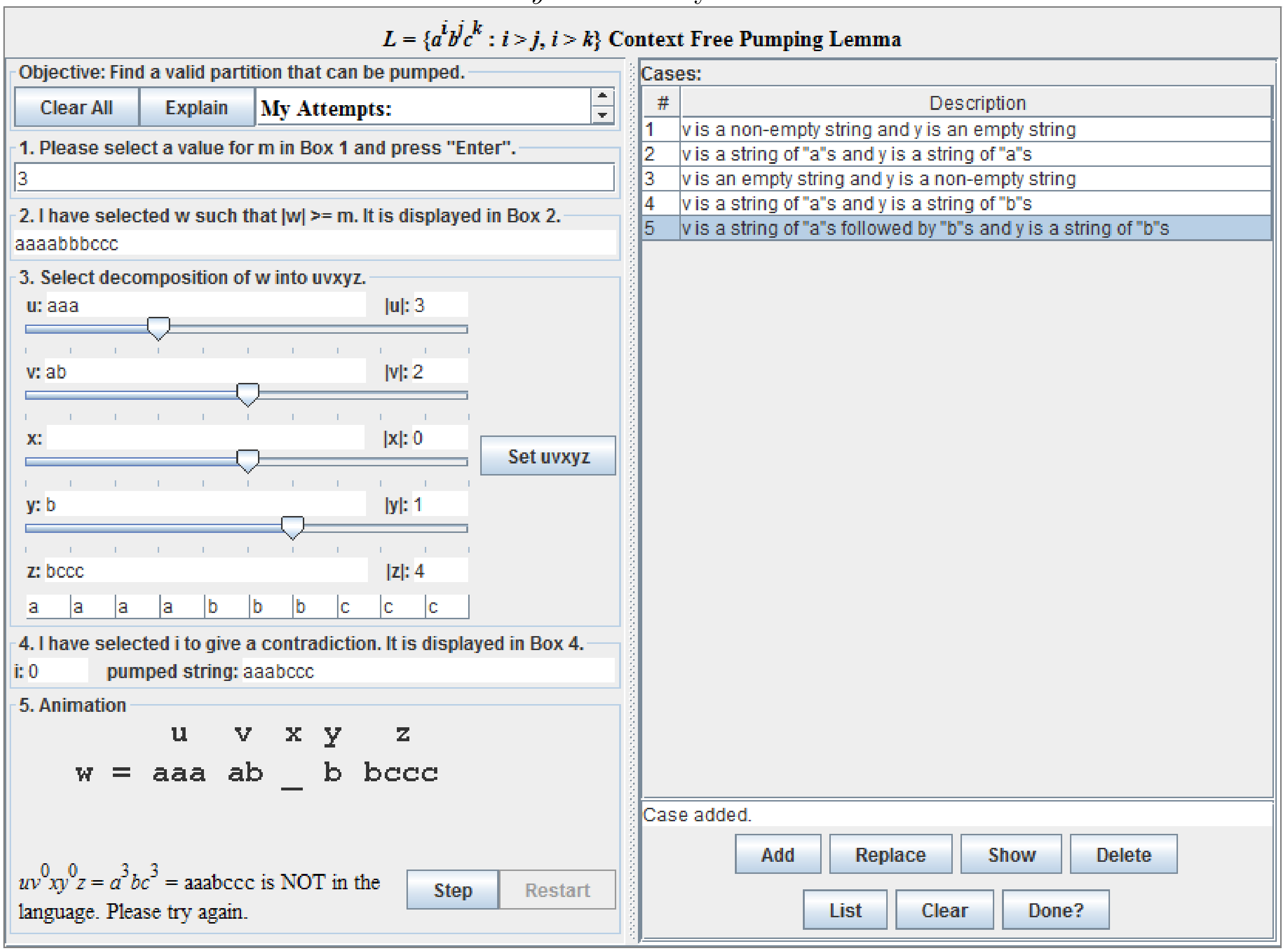
Then there is the case of v being empty but y having some as. Again pumping down works as it lessens the number of as while having the same number of bs and cs.



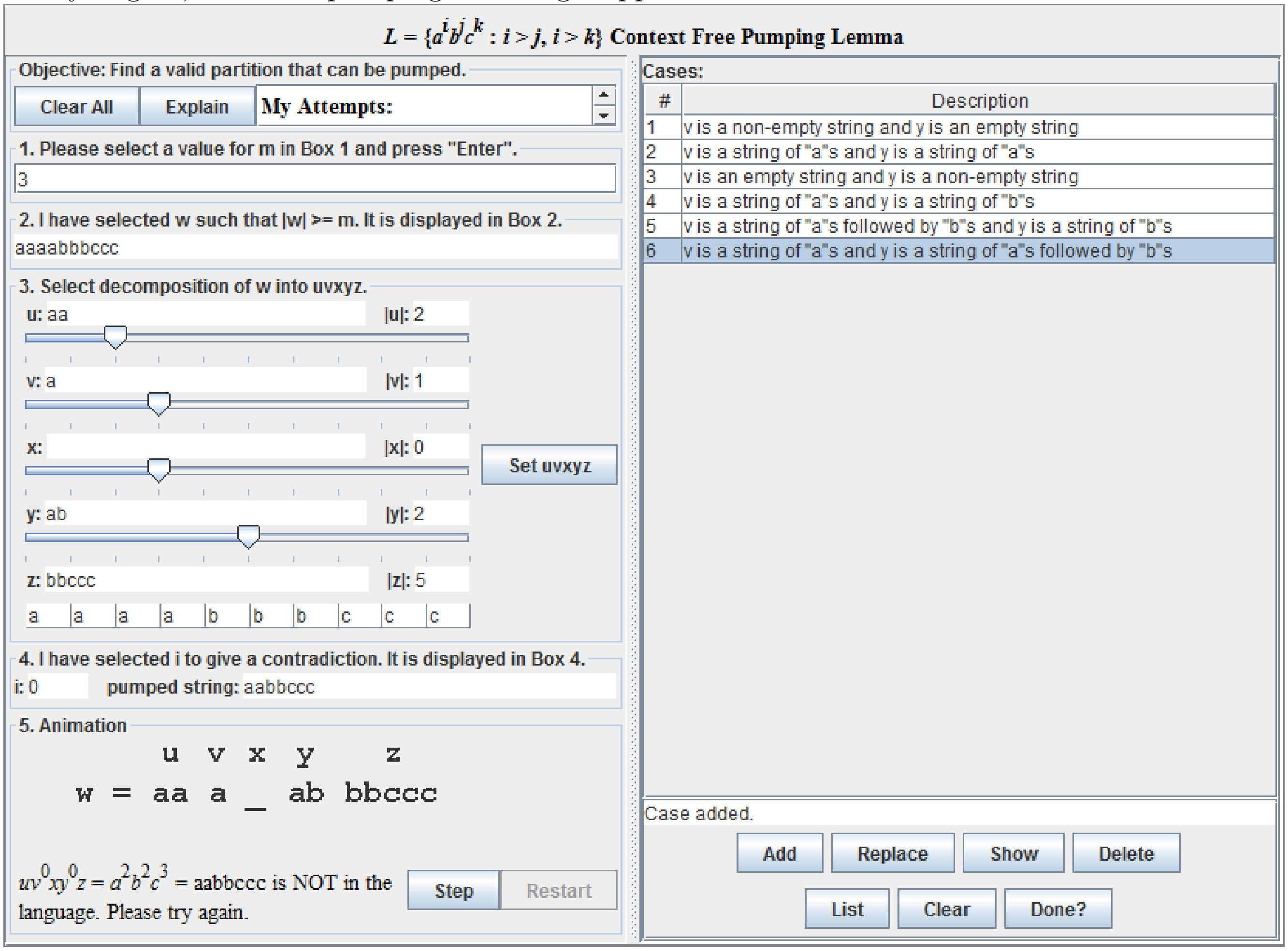
Now we move further and let y go into the b block. This case is interesting because while pumping down will reduce the number of as and bs, the number of cs stay the same.



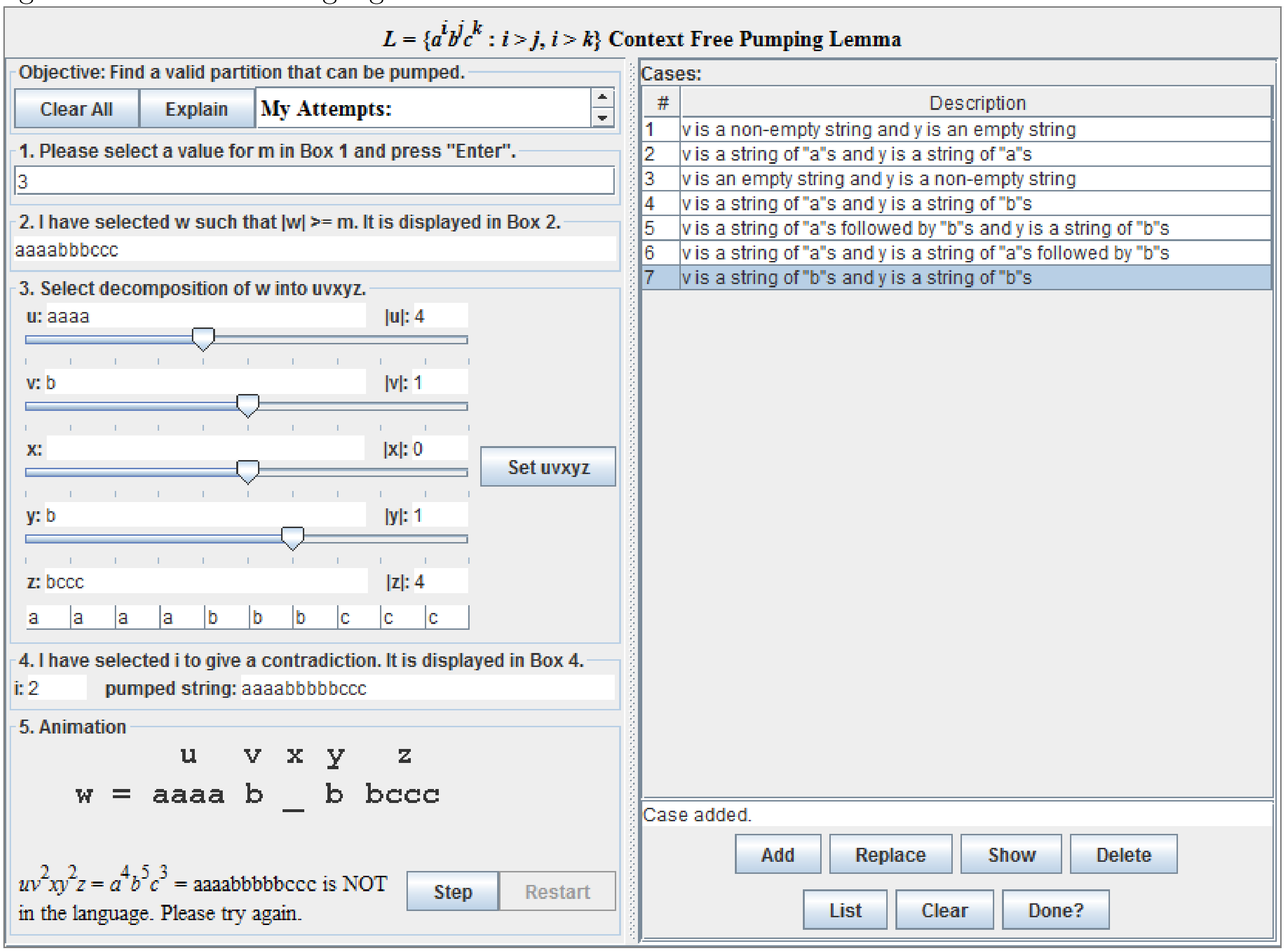
Now we let v straddle the a, b boundary. That is it contains some as and some bs. Again, pumping down works since v contains at least one a and therefore pumping down will bring down the number of as while neither v nor y contain any cs.



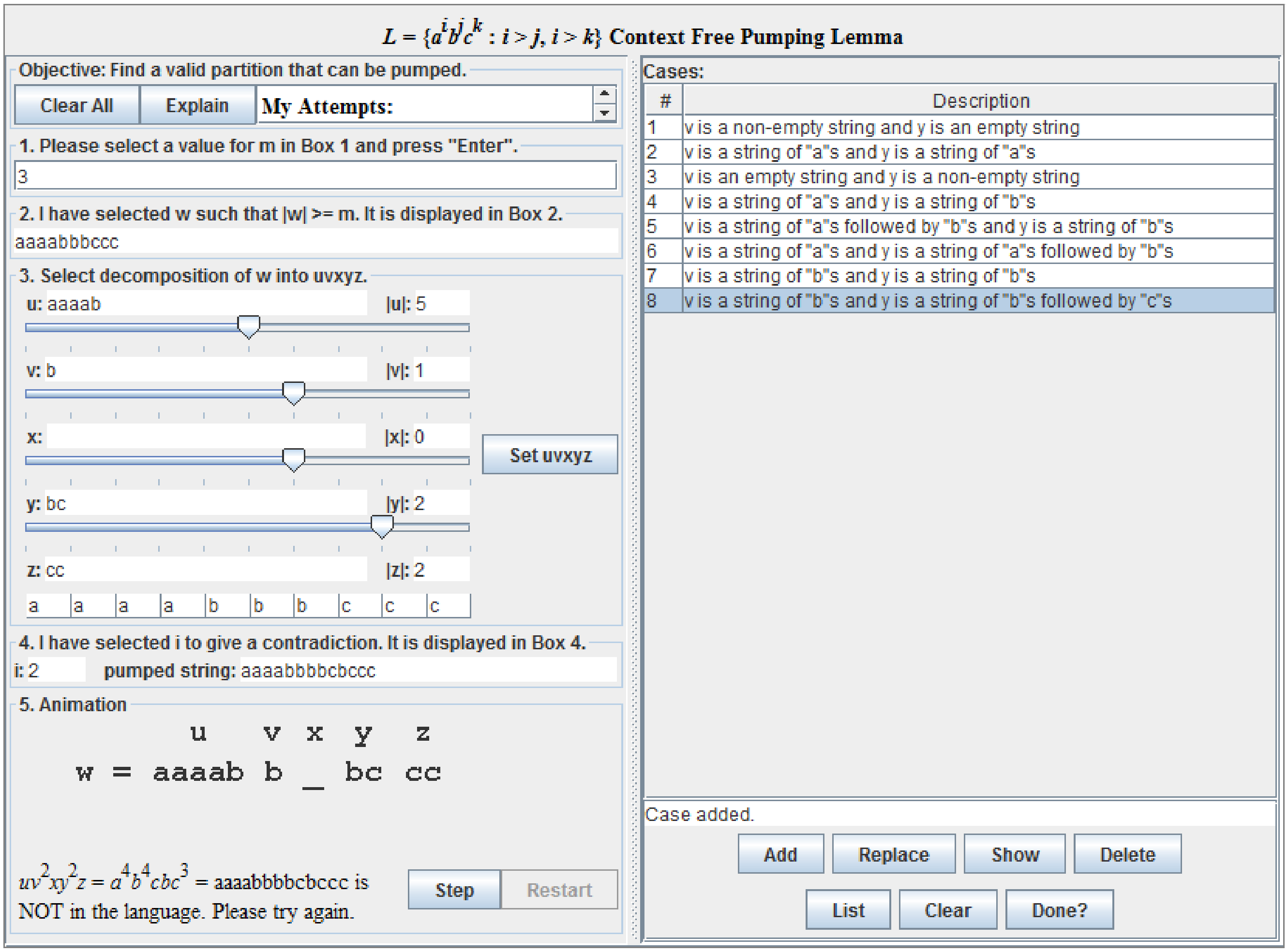
Similar to letting v straddle the a, b boundary, there is a case for y straddling the a, b boundary. Again, the same pumping down logic applies.



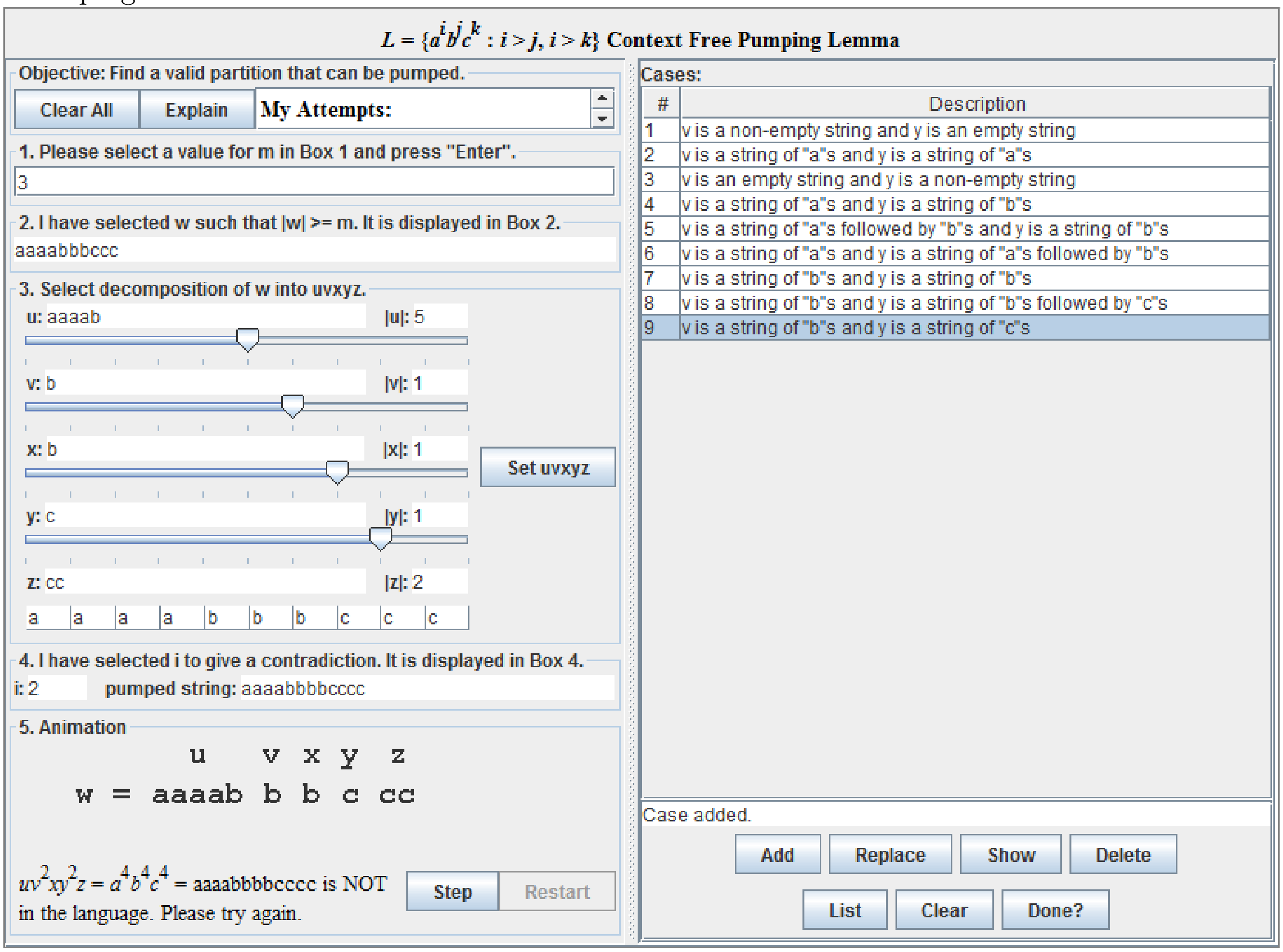
Now we move the v and y into the b block. Now pumping down might not work, but by pumping up we ensure that the number of bs go up at least by 2. Since we carefully chose the string so that the number of as is just one more than the number of bs, this pumped up string will not be in the language.



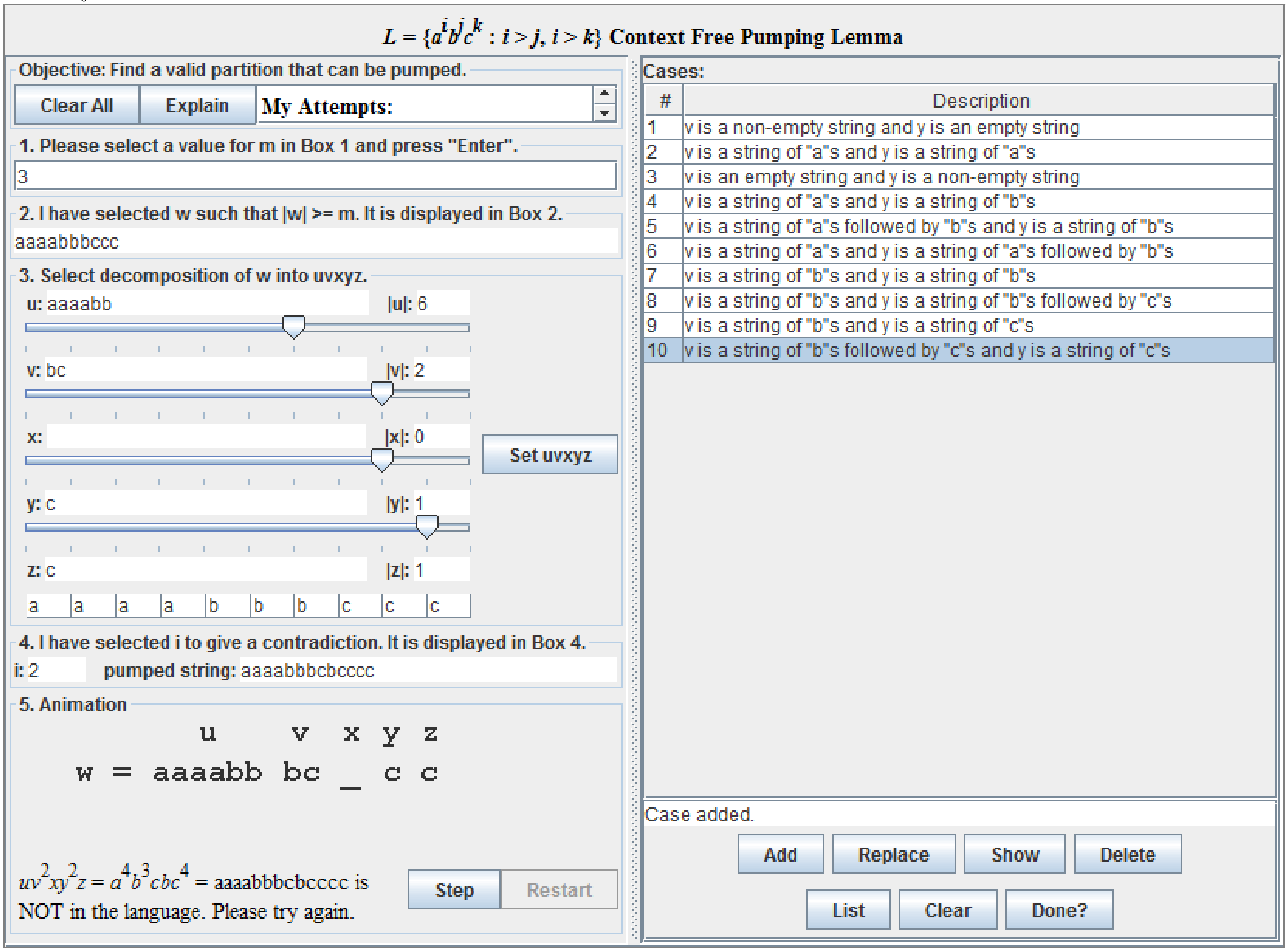
We then have the case of y going over the b and c boundary. Again, we need to pump up.



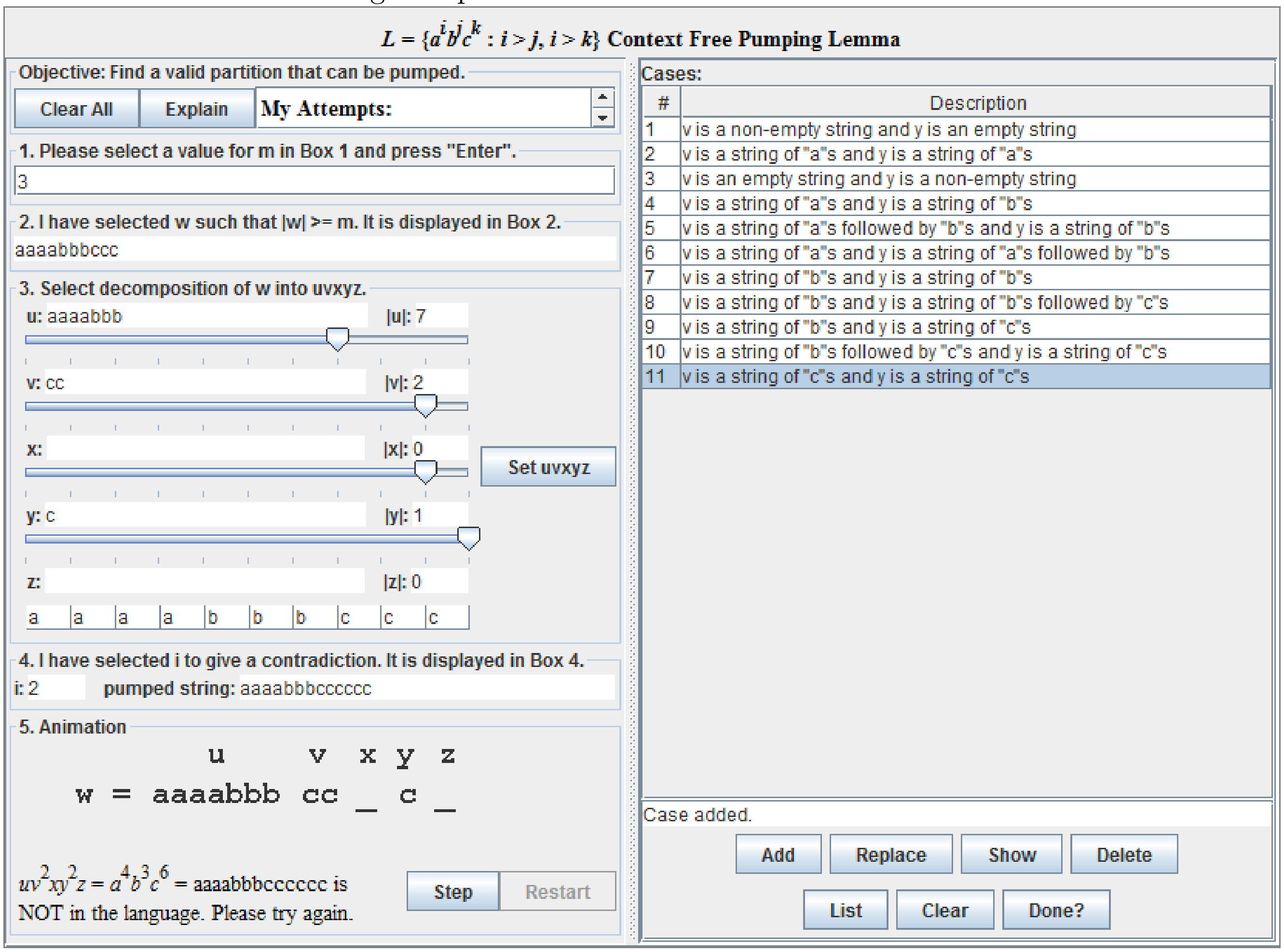
Then y moves further forward down the string and goes into the c region. Once again pumping up works, in this case it will increase both the number of bs and the number of cs while keeping the number of as the same.



Finally we move v further forward and make it have some bs and some cs.



The last case is the one where both v and y contain only cs. Again, we need to pump up and then we see the string cannot be part of the language because we have the same number of as but the number of cs has gone up.



Note that while it is useful to think of these individual cases and you can use JFLAP to see how the argument works for each case, it is also important to think about similarities between the cases. As you can see in the explanation section, the key to this particular argument not working out is that regardless of the split the v and y values together thus would have a maximum of two unique letters.