Regular Pumping Lemma
Martha Kosa

By now you have learned a lot about regular languages. They can be described by regular expressions, generated by regular grammars, and accepted by NFA's and DFA's (and DFA's with the minimum possible number of states).

Remember that a language over an alphabet Σ is a subset of Σ^*. How many subsets are possible? You have learned in discrete mathematics that a set with n elements has 2^n subsets. However, Σ^* is an infinite set. What does 2^∞ mean? The number of subsets of an infinite set is uncountably infinite. That means there is no way to list all the elements systematically. Thus, the number of languages over an alphabet is uncountable. If we could describe all the languages, we could count them. By using the logical contrapositive, since we cannot count them, we cannot describe them all.

Based on the above argument, regular expressions cannot describe every language, regular grammars cannot generate every language, and DFA's cannot accept every language. However, these models can support a countably infinite number of languages because of the power of recursive definitions.

Remember the recursive definition of a regular expression over alphabet Σ.
Basis: λ, $\varnothing$, and a are regular expressions for every symbol a $\in\Sigma$.
Recursive Steps: If x and y are regular expressions, then xy (concatenation) and $x \cup y$ (union) are regular expressions. If x is a regular expression, then x^* (Kleene star) is a regular expression. Parentheses can be used for grouping.
Closure: w is a regular expression if it can be derived from a basis element by a finite number of applications of the recursive steps.

Questions to Think About:
1. How many basis elements are there for the regular expressions over the alphabet {a}?
2. What are the basis elements for the regular expressions over the alphabet {0,1}?
3. How many basis elements are there for an arbitrary alphabet Σ?

Regular expressions obey several equivalence laws. Regular expressions are considered equivalent if they describe the same language. For example, union is commutative; that is, $x \cup y \equiv y \cup x$. $\varnothing$ is the identity for union; $x \cup \varnothing \equiv x$. λ is the identity for concatenation; $x\lambda \equiv \lambda x \equiv x$. Thus, there are many equivalent ways to describe the same regular language. Since every regular expression can be converted to an equivalent NFA (and then to an equivalent DFA), there are many equivalent ways to accept the same regular language. Since every DFA can be converted to an equivalent regular grammar, there are many equivalent ways to generate the same regular language.

We could develop (a non-terminating) algorithm for listing all possible regular expressions by listing the basis elements first and then systematically applying combinations of the recursive rules n times for each $n \geq 1$. Some of the expressions will be equivalent to each other, but it is still possible to produce an infinite number of non-equivalent regular expressions, just by seeing λ and a^n (a concatenated n times) for each $n \geq 1$.

Regular expressions are very powerful. With a small number of basic building blocks and combining operations, we can build up an infinite number of languages. However, many languages, actually an uncountable number, are missing. We will now explore some of these.

Noam Chomsky, a renowned professor at MIT in the area of computational linguistics, formalized a hierarchy of languages called the **Chomsky Hierarchy**. We now explore the first split in the language hierarchy between regular languages and non-regular languages.

What properties do all regular languages have? We have seen how they can be described by regular expressions, and these regular expressions can be converted to DFA's. Let's explore properties of regular languages as viewed through the lenses of DFA's.

All finite languages are regular, so we won't worry about these. We need to consider infinite languages then. Without loss of generality, let's consider the alphabet {0, 1}. What allows us to describe infinite sets with regular expressions? The Kleene star operator * does.

Questions to Think About:

1. What is the language described by λ^*?
2. What is the language described by $\varnothing^*$? This one is trickier.
3. What are the simplest infinite languages over our alphabet?
4. Is the regular expression 00* equivalent to the regular expression (00)*? Why or why not?
5. What is true about the length of every string described by the regular expression (00)*?
6. What do the minimum-state DFA's for those languages look like?

How do DFA's accept infinite languages? They accept infinite languages by having final states, of course, and by having loops in them, either directly or indirectly. If a language is infinite, it will have strings of arbitrary length, because otherwise the language would be finite. Notice that DFA stands for Deterministic **Finite** Automaton. What is finite? The number of states is finite. If we have accepted strings that are arbitrarily large, the number of symbols in the strings will exceed the number of states. This means that at least one state will be repeated when the DFA is processing such a string. This phenomenon is due to a mathematical principle called the **Pigeonhole Principle**. When processing a string, the DFA goes through a sequence of states and follows transitions. If the automaton is in state q_i and symbol a is processed, the next state will be q_j, where $\delta(q_i, a) = q_j$. All the DFA remembers is its current state. The string leading from the start state to the current state can be formed by concatenating all the characters labeling the transitions followed. When we encounter a repeated state q_k, the characters on the path taken between the first appearance of q_k and the second appearance of q_k can be removed and still lead to an accepted string, subject to the remainder of the string leading to a final state. We can also repeat those characters on that path as many times as we want and still obtain an accepted string, subject as before to the remainder of the string leading to the final state. This means that all sufficiently long accepted strings can be decomposed into three parts: the part before the first occurrence of the repeated state, the part in between the two occurrences, and the part after the repeated state (there may be more repetitions, but we don't care about those). This leads to the famous **Regular Pumping Lemma**, which we now state.

If L is a regular language, then there exists an integer $m > 0$ such that any $w \in L$ with $|w| \geq m$ can be decomposed as the concatenation $w = xyz$, with $|xy| \leq m$, $|y| \geq 1$, and $xy^iz \in L$ for all $i \geq 0$.

x represents the part of the string before the first occurrence of the repeated state, y represents the part in between the two occurrences, and z represents the part after the repeated state.

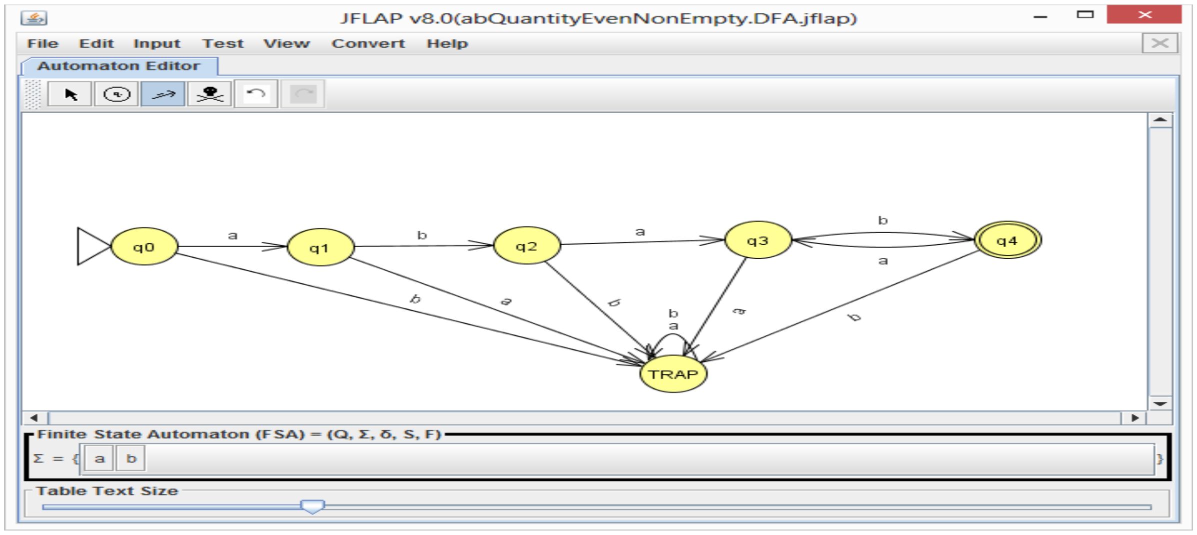
This Pumping Lemma is an implication. Remember that "if p then q" corresponds to "p implies q". Remember that every implication is equivalent to its contrapositive. The contrapositive of "p implies q" is "not p implies not q".

The contrapositive of the Pumping Lemma is the following (work through the quantifiers to convince yourself):

For some m > 0, if there is some $w \in L$ with $|w| \geq m$, such that w **cannot** be decomposed as the concatenation $w = xyz$, with $|xy| \leq m$, $|y| \geq 1$, and $xy^iz \in L$ for all $i \geq 0$, then L is **not** regular.

We can then use the Pumping Lemma to deduce that many languages are not regular. DFA's cannot count properly because they have no memory of the strings that they have processed. They can keep track of remainders easily because of the **Quotient-Remainder Theorem**, but they cannot maintain actual counts.

Consider the set of nonempty strings over {a,b} where every string repeats the pattern ab n times for some positive even integer n. The following is an DFA accepting this language. It is available in the file **abQuantityEvenNonEmpty.DFA.jflap**.


JFLAP v8.0(abQuantityEvenNonEmpty.DFA.jflap)
File Edit Input Test View Convert Help
Automaton Editor
q0
q1
q2
q3
q4
TRAP
a
b
a
b
a
b
a
b
a
b
a
b
Finite State Automaton (FSA) = (Q, Σ, δ, S, F)
Σ = { a b }
Table Text Size

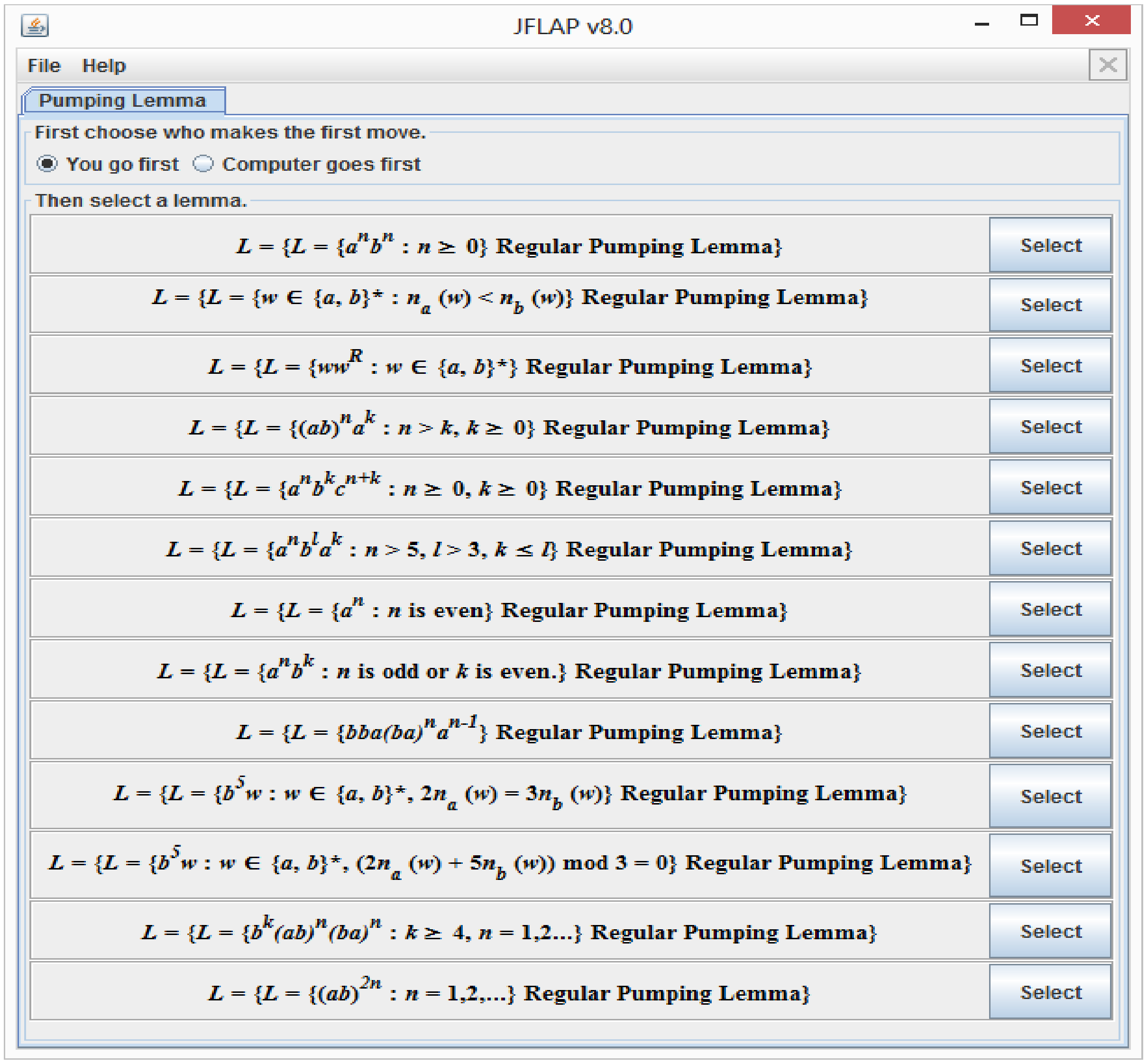
We will need to find a minimum positive integer m such that any accepted string w with $|w| \geq m$ can be decomposed into appropriate x, y, and z such that xy^iz is accepted for all $i \geq 0$.

We want to make sure that state q_4 is reached. How can we do this? We need to start at state q_0, then visit state q_1, then visit state q_2, then visit state q_3, and then finally visit state q_4. To be accepted, state q_4 needs to be the last state visited. This involves a path of length at least 4. Thus, m should be 4. Any other accepted strings will be of even length longer than 4, producing $4 + 2p$ for some $p \geq 1$.

We can use JFLAP to experiment with this.

Try It!

1. If JFLAP is not already active, start JFLAP and click the **Regular Pumping Lemma** button. Your view should look like the following.


JFLAP v8.0

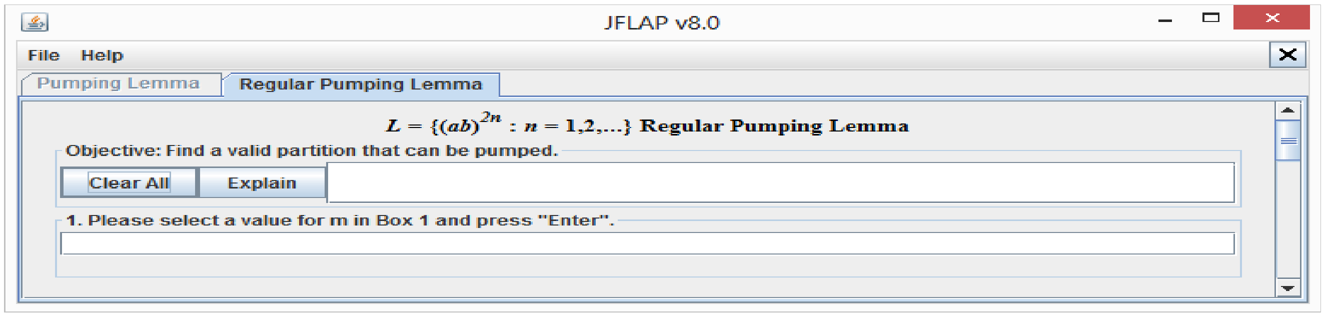
First choose who makes the first move.

● You go first ○ Computer goes first

Then select a lemma.

$L = \{L = \{a^n b^n : n \geq 0\}$ Regular Pumping Lemma}	Select
$L = \{L = \{w \in \{a, b\}^* : n_a(w) < n_b(w)\}$ Regular Pumping Lemma}	Select
$L = \{L = \{ww^R : w \in \{a, b\}^*\}$ Regular Pumping Lemma}	Select
$L = \{L = \{(ab)^n a^k : n > k, k \geq 0\}$ Regular Pumping Lemma}	Select
$L = \{L = \{a^n b^k c^{n+k} : n \geq 0, k \geq 0\}$ Regular Pumping Lemma}	Select
$L = \{L = \{a^n b^l a^k : n > 5, l > 3, k \leq l\}$ Regular Pumping Lemma}	Select
$L = \{L = \{a^n : n$ is even} Regular Pumping Lemma}	Select
$L = \{L = \{a^n b^k : n$ is odd or k is even.} Regular Pumping Lemma}	Select
$L = \{L = \{bba(ba)^n a^{n-1}\}$ Regular Pumping Lemma}	Select
$L = \{L = \{b^5 w : w \in \{a, b\}^*, 2n_a(w) = 3n_b(w)\}$ Regular Pumping Lemma}	Select
$L = \{L = \{b^5 w : w \in \{a, b\}^*, (2n_a(w) + 5n_b(w)) \bmod 3 = 0\}$ Regular Pumping Lemma}	Select
$L = \{L = \{b^k (ab)^n (ba)^n : k \geq 4, n = 1,2...\}$ Regular Pumping Lemma}	Select
$L = \{L = \{(ab)^{2n} : n = 1,2,...\}$ Regular Pumping Lemma}	Select

2. Find the language corresponding to our example and click the **Select** button. Your view should look similar to the following. You can click the **Explain** button for additional help.

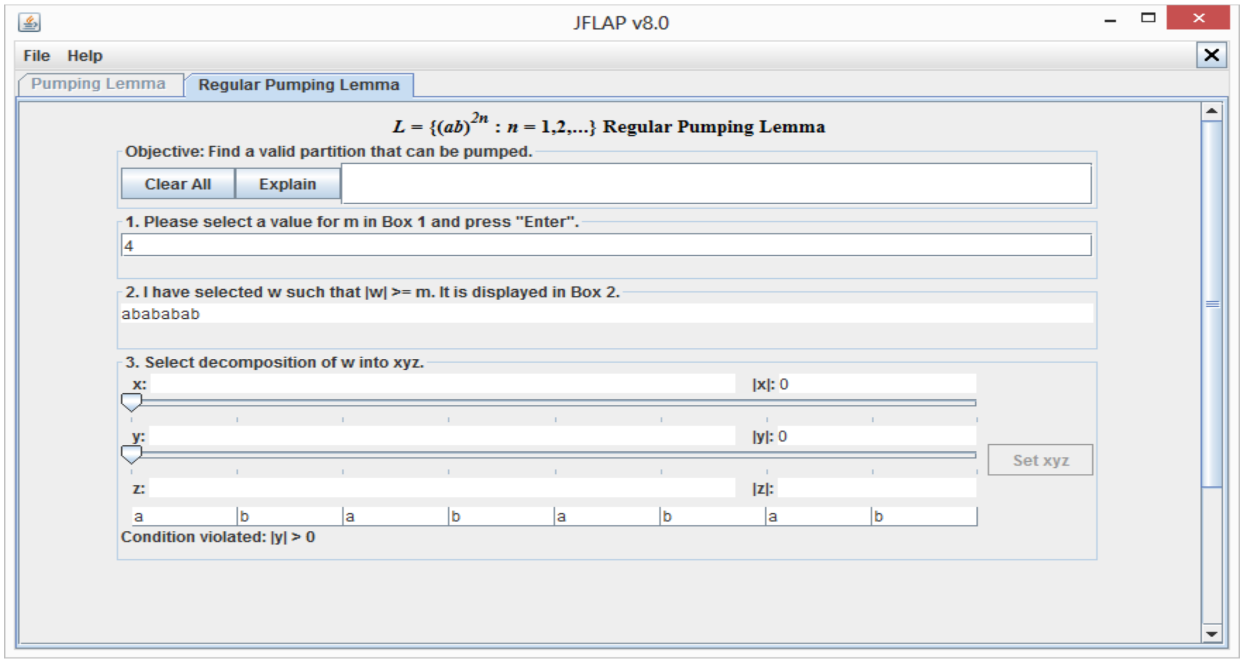

JFLAP v8.0

$L = \{(ab)^{2n} : n = 1,2,...\}$ Regular Pumping Lemma

Objective: Find a valid partition that can be pumped.

1. Please select a value for m in Box 1 and press "Enter".

3. Enter **4** in Box 1. Your view should look similar to the following.



$L = \{(ab)^{2n} : n = 1,2,\ldots\}$ Regular Pumping Lemma

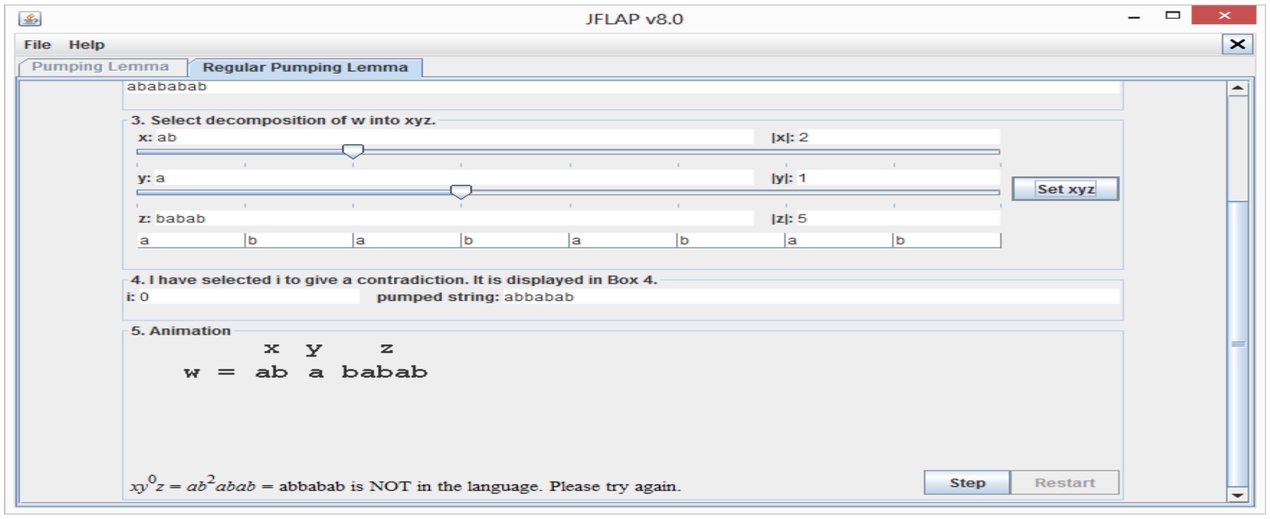
Objective: Find a valid partition that can be pumped.

1. Please select a value for m in Box 1 and press "Enter".
4

2. I have selected w such that |w| >= m. It is displayed in Box 2.
abababab

3. Select decomposition of w into xyz.
x: |x|: 0
y: |y|: 0
z: |z|:
a b a b a b a b
Condition violated: |y| > 0

4. Use the sliders to decompose w into x = ab, y = a, and z = babab. Click the **Set xyz** button. Your view should look similar to the following.



abababab

3. Select decomposition of w into xyz.
x: ab |x|: 2
y: a |y|: 1
z: babab |z|: 5
a b a b a b a b

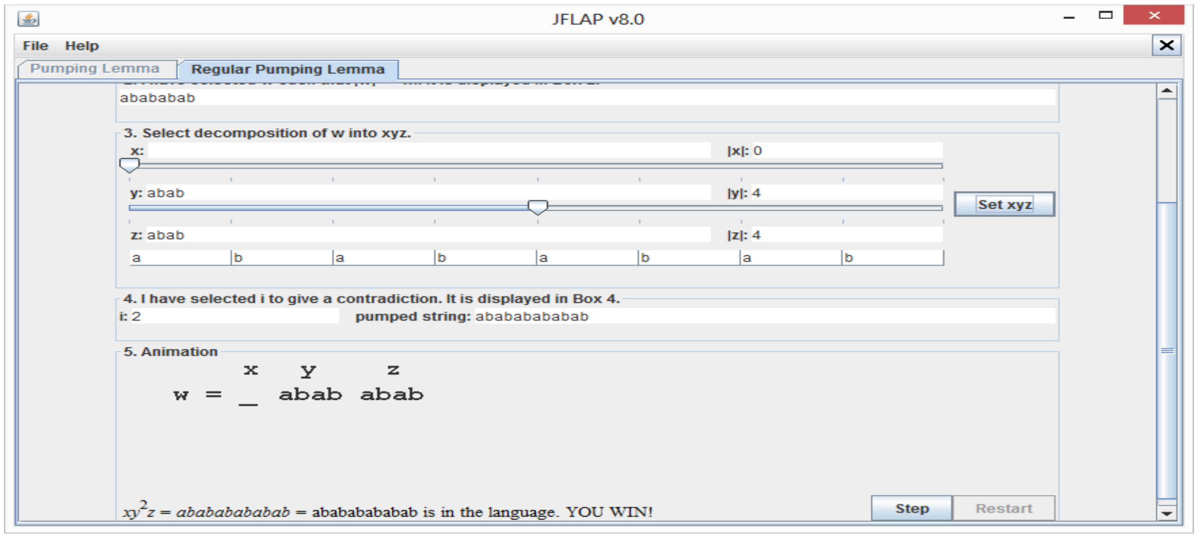
4. I have selected i to give a contradiction. It is displayed in Box 4.
i: 0 pumped string: abbabab

5. Animation

```
        x   y    z
w  =   ab   a  babab
```

$xy^0z = ab^2abab =$ abbabab is NOT in the language. Please try again.

5. Why doesn't this decomposition work? Think about our DFA, which is the minimum-state DFA accepting our language. Determine another decomposition that works. Remember that |xy| must be less than or equal to m, y cannot be empty, but x can be empty. Your view should be similar to the following.



ababab ab

3. Select decomposition of w into xyz.

x: |x|: 0

y: abab |y|: 4

z: abab |z|: 4

a	b	a	b	a	b	a	b

4. I have selected i to give a contradiction. It is displayed in Box 4.

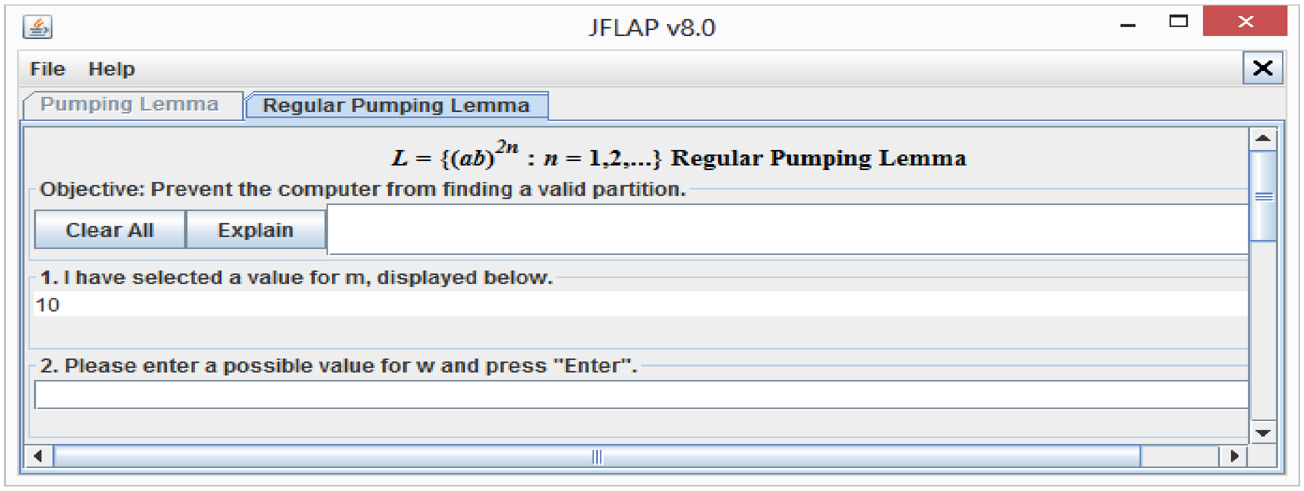
i: 2 pumped string: abababababab

5. Animation

```
          x    y     z
w  =  _  abab  abab
```

$xy^2z = abababababab$ = abababababab is in the language. YOU WIN!

6. Select ***File > Dismiss Tab***. Save the file with a descriptive name if you wish.
7. Select the **Computer Goes First** radio button and select the same language that you selected earlier. Your view should look similar to the following.



$L = \{(ab)^{2n} : n = 1,2,\ldots\}$ **Regular Pumping Lemma**

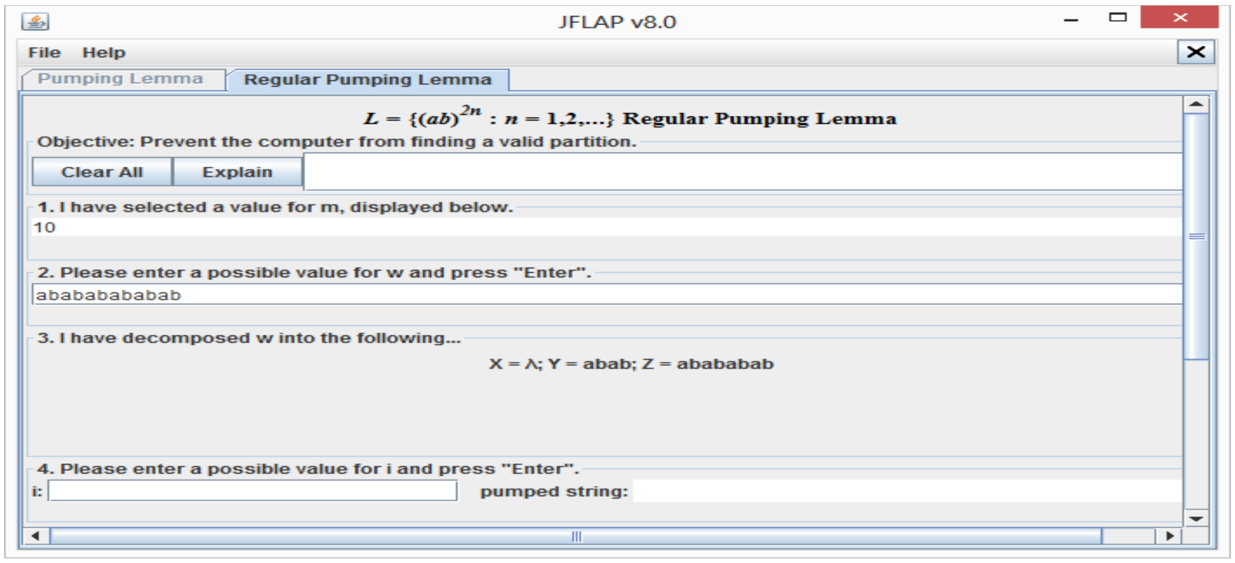
Objective: Prevent the computer from finding a valid partition.

1. I have selected a value for m, displayed below.

10

2. Please enter a possible value for w and press "Enter".

8. Enter a valid string from the language with length at least the value of the m chosen for you. Your view should look similar to the following.


JFLAP v8.0

File Help

Pumping Lemma | Regular Pumping Lemma

$L = \{(ab)^{2n} : n = 1,2,\ldots\}$ **Regular Pumping Lemma**

Objective: Prevent the computer from finding a valid partition.

Clear All | Explain

1. I have selected a value for m, displayed below.

10

2. Please enter a possible value for w and press "Enter".

abababababab

3. I have decomposed w into the following...

X = λ; Y = abab; Z = abababab

4. Please enter a possible value for i and press "Enter".

i: pumped string:

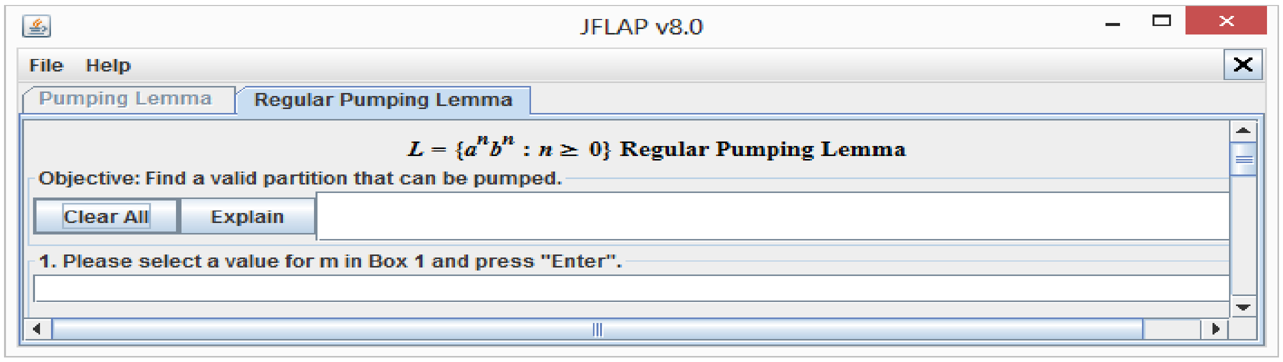
9. Experiment with different nonnegative values of i. What happens when you enter 1? Click **Step** to see the effects of repeating (or **pumping** the string). All the resulting strings are still in the language. Why? Because the language is regular, the pumping lemma will be satisfied. You will never be able to beat the computer.
10. Select ***File > Dismiss Tab***. Save the file with a descriptive name if you wish.

When will you be able to beat the computer? You will win when you give the computer a non-regular language. Assuming that the language is regular will result in a violation of the Pumping Lemma.

Earlier we discussed that DFA's can't count. Let's find a simple language where counting appears to be required. Consider the language $\{a^n b^n \mid n \geq 0\}$. What strings are in this language? They are all strings consisting of a sequences of a's followed by a sequence of b's of the same length. The lengths of all these strings will be even. The number of b's at the end must be the same as the number of a's at the beginning. This involves counting. If we assume the language is regular, it will satisfy the Pumping Lemma. We now will see via JFLAP that it does not, yielding a contradiction. The assumption that the language is regular is false. Thus, the language is not regular. No regular expression can describe that language, no DFA can accept that language, and no regular grammar can generate that language.

Try It!

1. If JFLAP is not already active, start JFLAP and click the **Regular Pumping Lemma** button.
2. Click the topmost **Select** button. Your view should look similar to the following.


JFLAP v8.0

File Help

Pumping Lemma | Regular Pumping Lemma

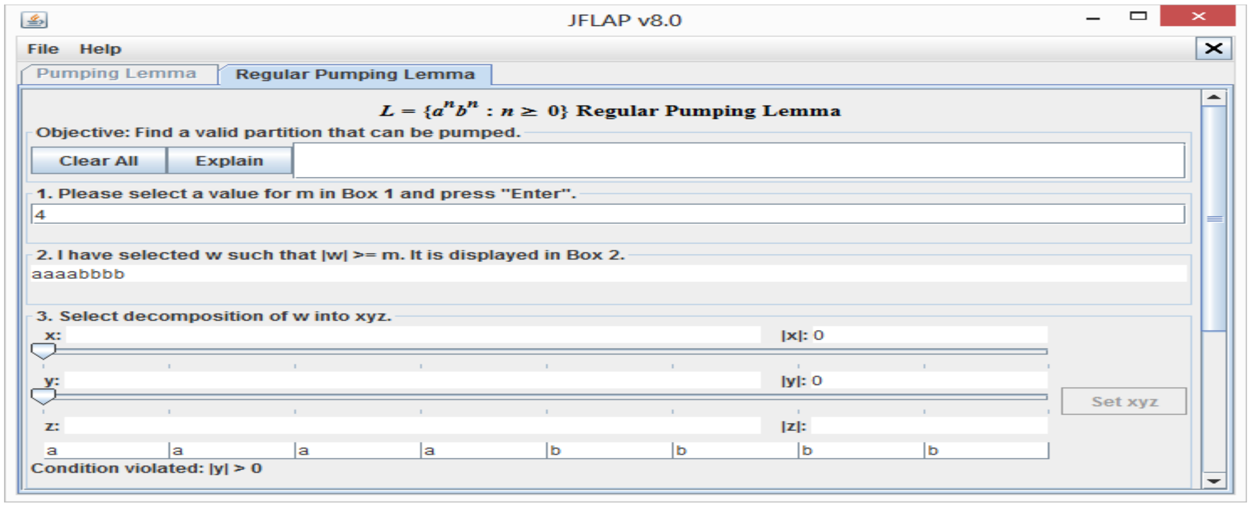
$L = \{a^n b^n : n \geq 0\}$ **Regular Pumping Lemma**

Objective: Find a valid partition that can be pumped.

Clear All | Explain

1. Please select a value for m in Box 1 and press "Enter".

3. Enter a reasonably large value for m, such as **4**. Your view should look similar to the following.



$L = \{a^n b^n : n \geq 0\}$ Regular Pumping Lemma

Objective: Find a valid partition that can be pumped.

1. Please select a value for m in Box 1 and press "Enter".

4

2. I have selected w such that |w| >= m. It is displayed in Box 2.

aaaabbbb

3. Select decomposition of w into xyz.

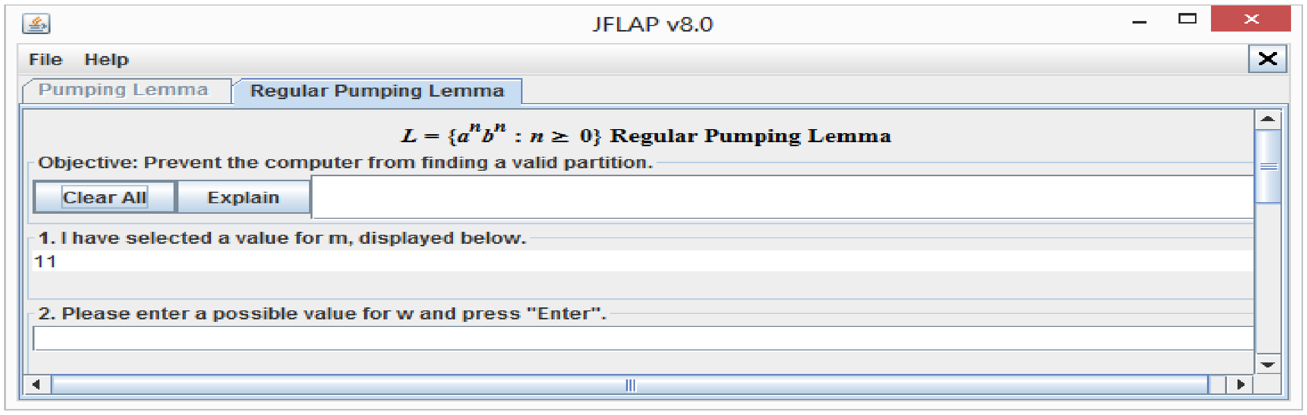
x: |x|: 0

y: |y|: 0

z: |z|:

Condition violated: |y| > 0

4. Use the sliders to decompose w into x and y satisfying the constraints. Remember that |xy| must be at least 1 and at most the value of m that you chose. Since |x| can be 0, |y| can range between 1 and m inclusive. You can try all possibilities, clicking the **Set xyz** button as needed, and viewing the pumped string by clicking the **Step** button. In all cases, the computer found a value of i for which xy^iz does not belong to the language.
5. Select ***File > Dismiss Tab***. Save the file with a descriptive name if you wish.
6. Select the **Computer Goes First** radio button and select the same language that you selected earlier. Your view should look similar to the following.



$L = \{a^n b^n : n \geq 0\}$ Regular Pumping Lemma

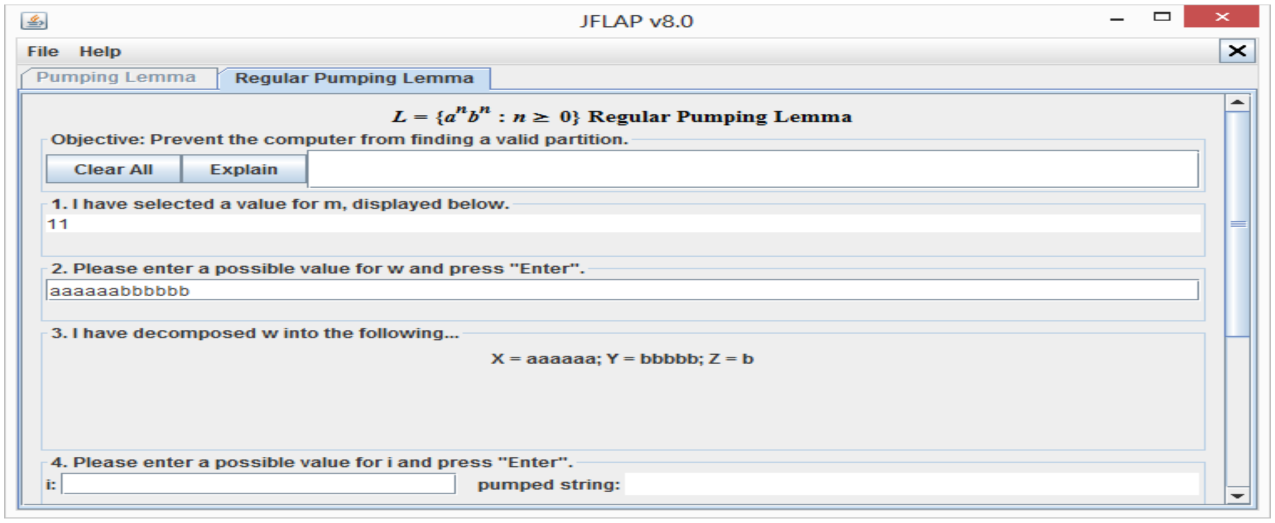
Objective: Prevent the computer from finding a valid partition.

1. I have selected a value for m, displayed below.

11

2. Please enter a possible value for w and press "Enter".

7. Enter a valid string from the language with length at least the value of the m chosen for you. Your view should look similar to the following.



$L = \{a^n b^n : n \geq 0\}$ **Regular Pumping Lemma**

Objective: Prevent the computer from finding a valid partition.

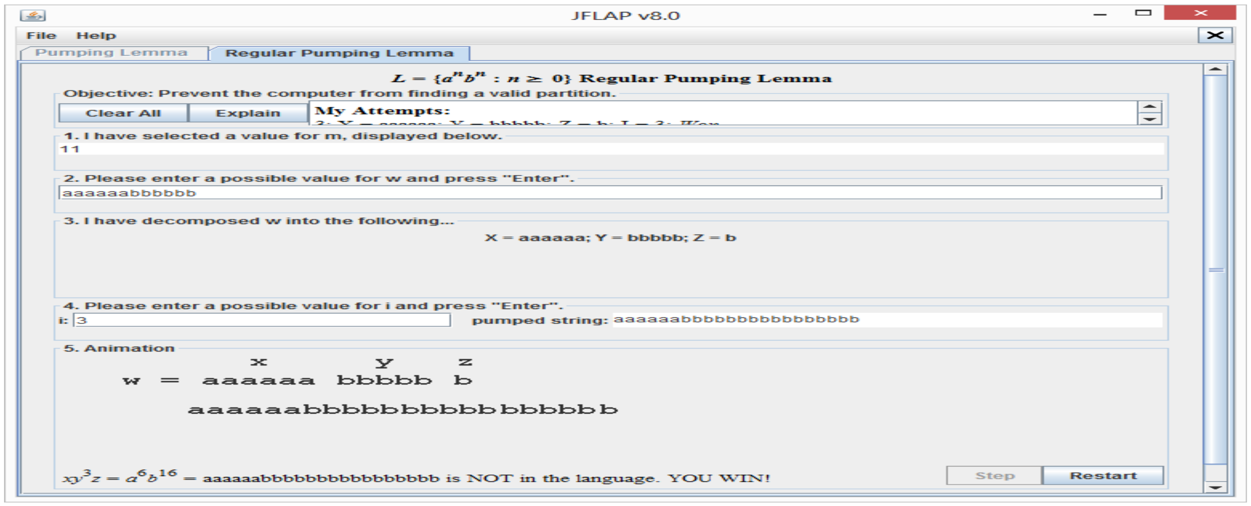
1. I have selected a value for m, displayed below.
11

2. Please enter a possible value for w and press "Enter".
aaaaaabbbbbb

3. I have decomposed w into the following...
X = aaaaaa; Y = bbbbb; Z = b

4. Please enter a possible value for i and press "Enter".
i: pumped string:

8. Enter a nonnegative value of i that is not equal to 1 that will cause xy^iz to be an invalid string. It will always be possible to do so because our language is not regular. You should see a view similar to the following.



$L = \{a^n b^n : n \geq 0\}$ **Regular Pumping Lemma**

Objective: Prevent the computer from finding a valid partition.

My Attempts:

1. I have selected a value for m, displayed below.
11

2. Please enter a possible value for w and press "Enter".
aaaaaabbbbbb

3. I have decomposed w into the following...
X = aaaaaa; Y = bbbbb; Z = b

4. Please enter a possible value for i and press "Enter".
i: 3 pumped string: aaaaaabbbbbbbbbbbbbbbb

5. Animation

```
          x        y     z
w  =  aaaaaa  bbbbb  b

     aaaaaabbbbbbbbbbbbbbbb
```

$xy^3z = a^6b^{16}$ = aaaaaabbbbbbbbbbbbbbbb is NOT in the language. YOU WIN!

Congratulations! You have explored the Regular Pumping Lemma, and have seen your first example of a non-regular language.

Questions to Think About:

1. True or false: Every subset of $\{a^nb^n \mid n \geq 0\}$ is a non-regular language.
2. True or false: $\{a^{2n}b^{2n} \mid n \geq 0\}$ is a regular language.
3. $\{a^nb^n \mid n \geq 0\}$ is a subset of $\{a^{2n}b^{2n} \mid n \geq 0\}$.
4. $\{a^{2n}b^{2n} \mid n \geq 0\}$ is a subset of $\{a^nb^n \mid n \geq 0\}$.