Non-Regular Languages

We know that some languages are not regular, because there are only countably many regexes and uncountably many languages. But are there any *useful* non-regular languages? The answer is Yes. Here is an example.

Suppose a word is built up from open brackets, written a, and closed brackets, written b, and we want to know whether it's well-bracketed. This is a commonly arising problem; for example, when you use IntelliJ to write a Java program, IntelliJ checks whether your brackets match correctly. This can be done using a stack. When you read a, you push a pebble onto the stack, and when you read b you pop the pebble off the stack. If you reach the end of the word, and the stack is empty, then you know the word is well-bracketed. But if it's not empty, or you read b when the stack is empty, the word is not well-bracketed.

Let's think for a moment about your computer. It has a finite memory, and therefore only finitely many states. It can try to run the above program, allocating part of its memory to represent the stack. But if the word begins with a very large number of a's, the stack will overflow.

Can your computer run a program that works for *all* words? The program should read in a word, letter by letter, and then announce whether or not the word is well-bracketed.

One solution is to use an external stack. That way, even though your computer has only finite memory, there's no limit on the amount of memory that the stack can occupy. But what if you don't have access to external memory? Can you install such a program on your computer? The answer is No. We shall now prove this.

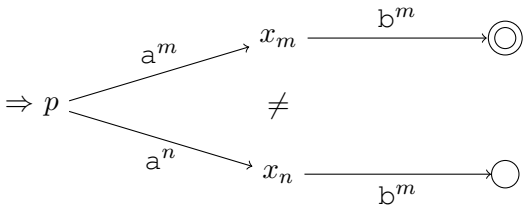
To put this more technically, we shall prove that the set L of well-bracketed words is not regular.

Suppose L is regular; then there's a DFA $(X, p, \delta, \mathsf{Acc})$ that recognizes it. Let's say that

- x_0 is the initial state p
- x_1 is the state that we reach when we start at p and read a
- x_2 is the state that we reach when we start at p and read aa
- x_3 is the state that we reach when we start at p and read aaa
- etc.

In summary, x_n is the state that we reach when we start at p and read $\mathtt{a}^n$. Now we're going to show that these states are all distinct, which implies that there are infinitely many states, a contradiction.

Suppose m and n are natural numbers with $m < n$. We want to show that $x_m \neq x_n$. If we start at x_m and read $\mathtt{b}^m$ we reach an accepting state, because $\mathtt{a}^m\mathtt{b}^m \in L$, but if we start at x_n and read $\mathtt{b}^m$ we reach a non-accepting state, because $\mathtt{a}^n\mathtt{b}^m \notin L$. So x_m and x_n can't be the same.



This is a general method to prove that a language is not regular. For a different language, you'll need to adjust the definition of x_n, and adjust the way you show $x_m \neq x_n$ for $m < n$.

Example: let the alphabet be $\{\mathtt{a}, \mathtt{b}\}$. Prove that the set of words in which a occurs more times than b is not regular.

Let's repeat the main point: a computer with finitely many states, and no access to external memory, cannot solve the matching problem for any non-regular language. For example, it can't determine whether a word (read in letter by letter) is well-bracketed.