Lecture 8: Mission Impumpsible II

Michael Engling

Department of Computer Science
Stevens Institute of Technology

CS 334 Automata and Computation
Fall 2015

OUTLINE: Non-Regular Languages

Impossible Objects

Partial vs. Entire

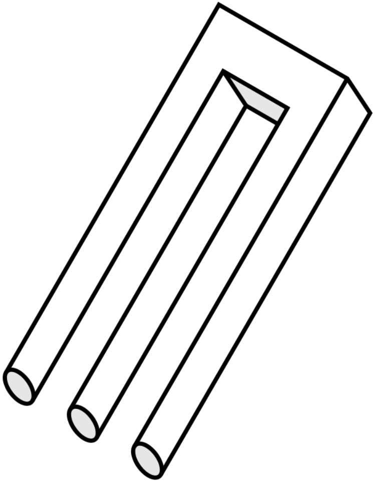
The Language $L = \{0^n 1 0^n | n \geqslant 0\}$ is not regular

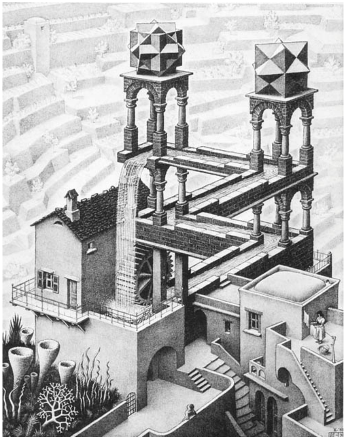
Impumpsible Strings

Another Non-Regular Language

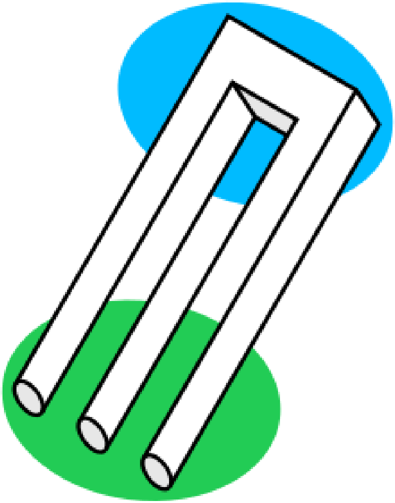
Whence Non-Regularity?

Impossible Objects

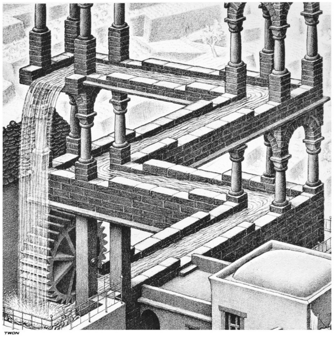

3 prongs, 2 arms


Water Doesn't Flow Up

Partial vs. Entire



3 prongs, 2 arms



Water Doesn't Flow Up

Impumpsible Objects

We will show that some supposedly possible mathematical objects are, in fact, impumpsible.

We will assume that a language is regular. That means it is recognized by a DFA/NFA.

We will then look at the whole machine, rather than just a little piece of it. We will find that our DFA/NFA *must* accept strings that are not in the language we assumed was regular.

This is the standard mathematical technique known as "Reductio ad Absurdum". We make a specific (local) assumption, and pull back to find that it violates a greater (global) truth.
We then *must* conclude that our original assumption was false.

The Language $L = \{0^n 1 0^n | n \geqslant 0\}$ is not regular

Suppose $\{L = 0^n 1 0^n | n \geqslant 0\}$ *is* regular. Then L is recognized by a DFA.

This DFA is after all, a DFA, hence it has a finite number of states, say, p where $p \geqslant 1$.

Now, using this p, the number of states in our DFA, we know that $0^p 1 0^p$ is accepted by our DFA.

In computing the first half of this string (p symbols), we must have used $p + 1$ states:

$$q_0 - 0 - q_i - 0 - q_j - 0 - \cdots - 0 - q_{n-1} - 0 q_n - 1 \ldots$$

There are too many pieces here for all of them to be unique. One of them (at least), must be a duplicate.

The Language $L = \{0^n 10^n | n \geqslant 0\}$ is not regular

$$q_0 - 0 - q_i - 0 - q_j - 0 - \cdots - 0 - q_{n-1} - 0 q_n - 1 \ldots$$

The duplicates might be anywhere. They might be q_0 and q_n in which case we're back where we started. They might be a stutterstep–perhaps q_j is repeated right after q_j. We don't know which state was repeated, but we know there is one that got repeated.

That repeated state forms a loop that comes back to itself. Maybe it includes every other state; maybe it just repeats itself like a hiccup, but from that state we deviated from a straight shot beginning to end and returned there.

What if we take that detour more than once?

The Language $L = \{0^n 1 0^n | n \geqslant 0\}$ is not regular

$$q_0 - 0 - q_i - 0 - q_j - 0 - \cdots - 0 - q_j - 0 q_n - 1 \ldots$$

What if we take that detour more than once?

$$q_0 - 0 - q_i - 0 - q_j - 0 - \cdots - 0 - q_j - 0 - \cdots - 0 - q_j - 0 q_n - 1 \ldots$$

The detour has some positive length–it computes some of our string. Call that length l. Now, if our DFA accepts $0^p 1 0^p$, it must also accept $0^{p+l} 1 0^p$ because a string that needs to take the loop twice. . . can.

If L is regular, then L is recognized by a DFA with $p \geqslant 1$ states. If that DFA accepts $0^p 1 0^p$ it also accepts $0^{p+l} 1 0^p$ for some $l > 0$. This latter string is *not* in L. This contradicts our assumption that L is recognized by a DFA.

Therefore, L is *not* recognized by a DFA.
$L = \{0^n 1 0^n | n \geqslant 0\}$ is not a regular language.

Mission Impumpsible

Ordinarily we use DFAs & NFAs to *compute* strings: We feed a string in and then check whether it ends in an accepting state.

We now bend the rules a bit, and use our machines to *generate* strings. When we show that our automaton *must* also generate bad strings, we conclude that our language is *not* regular.

1. We assume the language under consideration is regular and so is recognized by a DFA/NFA with p states,

2. We cleverly choose a long (i.e. length p or more) string,

3. We use that string to show that our DFA/NFA pumps out some other string(s) *not* in the language under consideration,

4. Having shown the machine does something impumpsible, we conclude the language is *not* regular.

The only troublesome part is being clever enough to find a string to set the machine against itself.

Mission Impumpsible II

If A is a regular language it is recognized by a DFA/NFA. Since the "F" stands for finite, any string 'long enough' must come from visiting at least one state more than once. We call this length p, the pumping length[1]. Any string in A of length at least p, can be written as $s = \alpha\beta\gamma$ satisfying the following conditions:

1. For each $i \geqslant 0$, $\alpha\beta^i\gamma \in A$, (We can blow βubbles.)
2. $|\beta| > 0$, (The βubbles aren't empty.)
3. $|\alpha\beta| < p$. (The βubble is in the first p symbols.)

This suggests that any string long enough has an initial part α, then a "βubble" segment, β that can be pumped up as many times as we like, and then a latter segment, γ (which may be very long). We'll exploit the βubble segment to make bad strings and show some languages are not regular.

[1] At most the number of states in a DFA/NFA recognizing it.

Another Non-Regular Language

Let P be the language of palindromes over $\Sigma = \{0, 1\}$.
P is not a regular language.

Suppose P is regular. It has a pumping length of p, that is, somewhere in the first p transitions any sufficiently long string enters some state twice.

Now, cleverly choose the palindrome: $s = 0^p10^p$;
s can be written as $s = \alpha\beta\gamma$.

Since the $\alpha\beta$ portion must be of length less than p, the βubble consists only of 0's. If we blow it a second time, we generate a new string $s' = \alpha\beta\beta\gamma$ with too many 0's in the first half to be a palindrome.

Now, *if* P is regular, *any* DFA that generates only palindromes also generates strings that are not palindromes. That is absurd.

Thus, P cannot be regular.

One More Non-Regular Language

Let D be the language of strings 0^i1^j, where $i > j$.
D is not a regular language.

Consider the string $s = 0^{p+1}1^p \in D$.

Now, $s = 0^p1^p$ can be written as $s = \alpha\beta\gamma$ as in the pumping lemma.

We can blow up the βubble as many times as we like, but it will only increase the number of 0s up front. That creates strings "even better" than s.

What if we "pop" the βubble instead of pumping it up?

Consider the string $s' = \alpha\beta^0\gamma = \alpha\varepsilon\gamma = \alpha\gamma$.

The βubble portion had at least one 0 in it, so $s' = 0^{p+1-|\beta|}\imath^p$, but that cannot be a string in D. A contradiction.

D cannot be a regular language.

Whence Non-Regularity?

What makes a language non-regular?

How can you tell a language *might* be non-regular?

What goes wrong?

Basically, our machines (DFAs & NFAs) have a bounded memory. They have n states, and are thus constrained to n possible recollections. They have no way to "think outside the box".

In the next chapters we will give our computing devices the ability to remember more than just what their individual states permit. Infinite memory may not be possible, but unbounded memory is within our grasp, and it will open new possibilities for us and our computing machines.