The Pumping Lemma: limitations of regular languages

Informatics 2A: Lecture 8

Mary Cryan

School of Informatics
University of Edinburgh
mcryan@inf.ed.ac.uk

3 October, 2018

Recap of Lecture 7

- Lexical classes in programming languages may typically be specified via regular languages.
- The lexing algorithm runs a parallel NFA in order to find the next lexeme using the principle of longest match.
- Regular language theory can also be used in verifying subtle properties of large finite-state systems (e.g. those arising from interactions of several simpler systems).

Non-regular languages

We've hinted before that not all languages are regular. E.g.

- Java (or any other general-purpose programming language).
- The language $\{a^n b^n \mid n \geq 0\}$.
- The language of all *well-matched* sequences of brackets (,).
  N.B. A sequence x is well-matched if it contains the same number of opening brackets '(' and closing brackets ')', and no initial subsequence y of x contains more)'s than ('s.

But how do we know these languages aren't regular?

Can we come up with a general technique for proving the non-regularity of languages?

The basic intuition: DFAs can't count!

Consider $L = \{a^n b^n \mid n \geq 0\}$. Just suppose, hypothetically, there *were* some DFA M with $\mathcal{L}(M) = L$.

Suppose furthermore that M had just processed a^n, and some continuation b^m was to follow.

Intuition: M would need to have *counted* the number of a's, in order to know how many b's to require.

More precisely, let q_n denote the state of M after processing a^n. Then for any $m \neq n$, the states q_m, q_n must be different, since b^m takes us to an accepting state from q_m, but not from q_n.

In other words, M would need infinitely many states, one for each natural number. Contradiction!

Exercises

Which of the following languages are regular?

1. Strings with an odd number of a's and an even number of b's.
2. Strings containing strictly more a's than b's.
3. Strings such that (no. of a's)$\times$(no. of b's) $\equiv 6$ mod 24.
4. Strings over $\{0, \ldots, 9\}$ representing integers divisible by 43.

Exercises

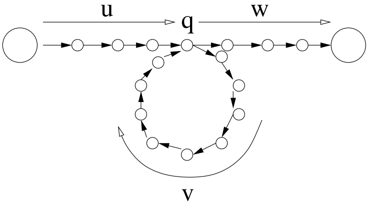
Which of the following languages are regular?

1. Strings with an odd number of a's and an even number of b's.
2. Strings containing strictly more a's than b's.
3. Strings such that (no. of a's)$\times$(no. of b's) $\equiv 6 \bmod 24$.
4. Strings over $\{0, \ldots, 9\}$ representing integers divisible by 43.

Answer: 1 is regular (see similar example on slide 13 of Lecture 4).
2 isn't regular: intuitively, we'd need to keep track of difference between no. of a's and no. of b's, which could be any integer.
3 is regular: we only need to keep track of no. of a's mod 24 and no. of b's mod 24, which we can do with $24 \times 24 = 576$ states.
4 is regular: we can keep track of the number read so far mod 43.
On reading a new digit d, we go from state i to state $(10i + d) \mod 43$.

Loops in DFAs

Let M be a DFA with k states. Suppose, starting from some state of M, we process a string y of length $|y| \geq k$. We then pass through a sequence of $|y|+1$ states. So there must be some state q (at least one) that's visited *at least twice*:



(Note that u and w might be ϵ, but v definitely is not.)

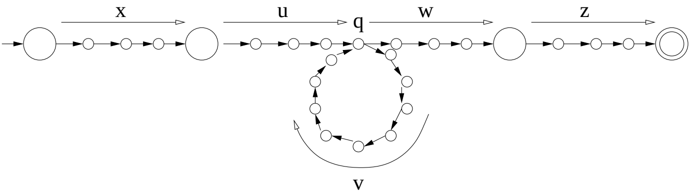
So any string y with $|y| \geq k$ can be decomposed as uvw, where

- u is the prefix of y that leads to the first visit of q
- v takes us once round the loop from q to q,
- w is whatever is left of y after uv.

A general consequence

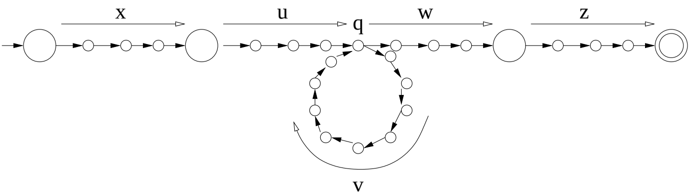
If L is *any* regular language, we can pick *some* DFA M for L, and it will have some number of states — say k. This has to be the case, as we have a **Finite** automaton.

Suppose we run M on a string $xyz \in L$, where $|y| \geq k$. There must be at least one state q visited twice in the course of processing y:



(There may be other 're-visits' to states not shown in the picture.)

The idea of 'pumping'



So y (the middle substring, of length $\geq k$) can be decomposed as uvw, where

- xu takes M from the initial state to q,
- $v \neq \epsilon$ takes M once round the loop from q to q,
- wz takes M from q to an accepting state.

But now M will be oblivious to whether, or how many times, we go round the v-loop! So the DFA also accepts $x(uw)z, x(uv^2w)z, \ldots$

So we can 'pump in' as many copies of the substring v as we like, knowing that we'll still end in an accepting state.

The pumping lemma: official form

The pumping lemma basically summarizes what we've just said.

Pumping Lemma. Suppose L is a regular language. Then L has the following property.

> *(P) There exists $k \geq 0$ such that, for all strings x, y, z with $xyz \in L$ and $|y| \geq k$, there exist strings u, v, w such that $y = uvw$, $v \neq \epsilon$, and for every $i \geq 0$ we have $xuv^iwz \in L$.*

(note this is also true if we forget the x and z and just have strings y with $|y| \geq k$. We have our reasons for complicating matters ...)

The pumping lemma: contrapositive form

Since we want to use the pumping lemma to show a language *isn't* regular, we usually apply it in the following equivalent but back-to-front form.

Suppose L is a language for which the following property holds:

> ($\neg P$) *For all $k \geq 0$, there exist strings x, y, z with $xyz \in L$ and $|y| \geq k$ such that, for every decomposition of y as $y = uvw$ where $v \neq \epsilon$, there is some $i \geq 0$ for which $xuv^iwz \notin L$.*

Then L is not a regular language.

N.B. The pumping lemma can only be used to show a language *isn't* regular. Showing L satisfies (P) doesn't prove L is regular!

To show that a language *is* regular, need to give some DFA or NFA or regular expression that defines it.

The pumping lemma: a user's guide

So to show some language L is not regular, it's enough to show that L satisfies $(\neg \mathrm{P})$.

Note that $(\neg \mathrm{P})$ is quite a complex statement:

$$\forall k \cdots \ \exists x, y, z \cdots \ \forall u, v, w \cdots \ \exists i \cdots$$

Helpful intuition: Values for the variables quantified by $\forall$ are chosen by an imaginary 'opponent' who is claiming that P is true. Values for the variables quantified by $\exists$ are chosen by you.

We'll look at a simple example first, then offer some advice on the general pattern of argument.

Example 1

Consider $L = \{a^n b^n \mid n \geq 0\}$.

We show that L satisfies $(\neg \text{P})$.

Suppose we're given $k \geq 0$. (Opponent chooses the value of k. Our argument has to work for *all* k.)

Consider the strings $x = \epsilon$, $y = a^k$, $z = b^k$. Note that $xyz \in L$ and $|y| \geq k$ as required. (We make a clever choice of x, y, z.)

Suppose now we're given a decomposition of y as uvw with $v \neq \epsilon$. (Opponent chooses this decomposition. Our argument has to work for all such u, v, w.)

Let $i = 0$. (We make a clever choice of i.)

Then $uv^i w = uw = a^l$ for some $l < k$. So $xuv^i wz = a^l b^k \notin L$. (We win!)

Thus L satisfies $(\neg \text{P})$, so L isn't regular.

Use of pumping lemma: general pattern

On the previous slide, the formal argument is in black, whereas the parenthetical comments in blue are for intuition only.

The comments emphasise the care that is needed in dealing with the quantifiers in the property ($\neg$P). In general:

- You are not allowed to choose the number $k \geq 0$. Your argument has to work for every possible value of k.
- You have to choose the strings x, y, z, which might depend on k. You must choose these to satisfy $xyz \in L$ and $|y| \geq k$. Also, y should be chosen adaptively to 'disallow pumping' ...
- You are not allowed to choose the strings u, v, w. Your argument has to work for every possible decomposition of y as uvw with $v \neq \epsilon$.
- You have to choose the number i ($\neq 1$) such that $xuv^iwz \notin L$. Here i might depend on all the previous data.

Example 2

Consider $L = \{a^{n^2} \mid n \geq 0\}$.
We show that L satisfies ($\neg$P):

Suppose $k \geq 0$.
Let $x = a^{k^2-k}$, $y = a^k$, $z = \epsilon$, so $xyz = a^{k^2} \in L$.
Given any splitting of y as uvw with $v \neq \epsilon$, we have $1 \leq |v| \leq k$.
Take $i = 2$. Since $xuvwz = a^{k^2}$, we have that $xuv^2wz = a^{k^2+|v|}$.
And $1 \leq |v| \leq k$ means that $k^2 + 1 \leq k^2 + |v| \leq k^2 + k$.
However, there are no perfect squares between k^2 and $k^2 + 2k + 1$.
So the length of xuv^2wz isn't a perfect square. Thus $xuv^2wz \notin L$.

Thus L satisfies ($\neg$P), so L isn't regular.

Subtle point: what are the x and z for?

All the action seems to happen within $y = uvw$. Do we really need x and z?

Often, we can get away with taking $x = z = \epsilon$. But other choices may of x, z may give us more control over where the 'loop' occurs.

Example: L is the set of strings containing more a's than b's.

- First approach: Given k, take $x = \epsilon$, $y = a^{k+1}b^k$, $z = \epsilon$. If $y = uvw$, we have three cases to consider, according to where v begins and ends.
- Second approach: Given k, take $x = a^{k+1}$, $y = b^k$, $z = \epsilon$. Only one case to consider. Taking $i = 2$ always works.

Reading and "rest of this week"

Relevant reading: Kozen chapters 11, 12.

That concludes the course material on (formal) regular languages.

I am away Thursday and Friday of this week.

- My office hour (normally Thursdays) is cancelled this week.
- Shay is going to take Friday's lecture, introducing context-free languages.