Non-Regular Languages

CSCI 2210

2023-10-04

Regular Languages

So far we have studied **regular languages**, the string languages that are:

- decidable by finite automata, equivalently,
- describable by regular expressions.

We have studies operations on string languages that preserve regularity, including:

- complementation
- intersection
- union
- concatenation
- iteration

Regular Languages

A natural question to ask is:

- Are all string languages regular?

The answer turns out to be “no”.

A natural follow-up question is:

- How can we know that a language is *not* regular.

So far we have given criteria to show that a language *is* regular:

produce a finite automaton that decides it
or a regular expression that describes it.

But what if we can’t?

Maybe we’re just not being creative and clever enough.

Showing a Language is Not Regular

We can use the following strategy to prove that a string language is not regular:

- identify a property that all regular languages have,
- show that this language does not have that property.

There are a number of properties we could use. One of the easiest to understand involves cycles in state transition graphs of DFAs.

Forcing Cycles in Graphs

Theorem

If G is a nonempty finite graph with n vertices, and p is a path in G with $|p| \geq n$ then p must contain a *cycle* (a non-stationary path whose source and target vertices are equal).

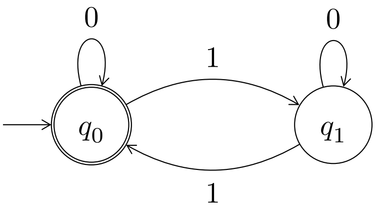
This is because the first edge in p visits two (not necessarily distinct) vertices, and each subsequent edge increases this number by one.

So a path of length n must visit $n + 1$ vertices.

But there are only n vertices in the graph, so they can't all be distinct.

Forcing Cycles in Strate Transition Graphs

Consider the following state transition graph for a DFA M:



M accepts the word $w := 0\,1\,1\,0$,
and $4 = |w| \geq |\mathrm{Q}| = 2$.

So the following path through the state transition graph must contain a cycle:

$$[(q_0), 0, (q_0), 1, (q_1), 1, (q_0), 0, (q_0)]$$

Pumping Paths in Graphs

Whenever we have a path that contains a cycle in a graph:

$$[(v_0), e_0, \cdots, \underbrace{(v_i), e_i, \cdots, (v_i)}_{\text{cycle}}, e_j, \cdots, (v_f)]$$

we can "pump" it by iterating the cycle any number of times.

We can do this in two ways:

pump down the path by *omitting* the cycle (iterating it zero times),

$$[(v_0), e_0, \cdots, \underbrace{(v_i)}_{\text{cycle omitted}}, e_j, \cdots, (v_f)]$$

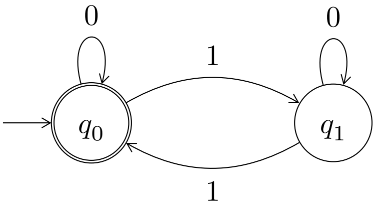
pump up the path by *repeating* the cycle (iterating it two or more times).

$$[(v_0), e_0, \cdots, \underbrace{(v_i), e_i, \cdots, (v_i), e_i, \cdots, (v_i)}_{\text{cycle repeated}}, e_j, \cdots, (v_f)]$$

Pumping alway results in a path that is parallel to the original path.

Pumping Cycles in State Transition Graphs

Because the DFA



accepts the word $0\,1\,1\,0$ whose path through the transition graph includes the cycle

$$[(q_0), 1, (q_1), 1, (q_0)]$$

it must also accept:

- the "pumped down" word $0\,0$ and
- the "pumped up" word $0\,1\,1\,1\,1\,0$.

Bounding Path Segment Lengths

Recall:

Theorem

If G is a nonempty finite graph with n vertices, and p is a path in G with $|p| \geq n$ then p must contain a cycle.

We can divide the path p into three consecutive segments:
a *prefix* x, a *cycle* y, and a *suffix* z, such that $p = x +\!\!+\, y +\!\!+\, z$.

What can we say about the lengths of these segments?

- If $|y| > n$ then we could have found a *smaller* cycle instead.
- If $|x| > n$ then we could have found an *earlier* cycle instead.
- If $|z| > n$ then we could have found an *later* cycle instead.

Bounding Path Segment Lengths ctd.

In fact, we can do a bit better:

Theorem

If G is a nonempty finite graph with n vertices, and p is a path in G with $|p| \geq n$ then there are paths x , y, and z such that y is a cycle, $p = x \mathbin{+\!\!+} y \mathbin{+\!\!+} z$, and either one of the following conditions are met:

- $|x \mathbin{+\!\!+} y| \leq n$,
- $|y \mathbin{+\!\!+} z| \leq n$

You can use the first one to get a “small, early” cycle and the second one to get a “small, late” cycle.

Pumping Lemma for Regular Languages

This is the justification for the following result:

Theorem (Pumping Lemma)

Every regular language $\mathrm{L} \subseteq \Sigma^*$ has a number $p(\mathrm{L}) \in \mathbb{N}$ (its “pumping length”) so that for any word $w \in \mathrm{L}$ if $|w| \geq p(\mathrm{L})$ then there are words $x, y, z \in \Sigma^*$ with $y \neq \varepsilon$ satisfying the conditions:

decomposition: $w = x \cdot y \cdot z$,

pumping: $\forall\, n \in \mathbb{N} \,.\, x \cdot y^n \cdot z \in \mathrm{L}$

bounding: $|x \cdot y| \leq p(\mathrm{L})$ (or, alternatively, $|y \cdot z| \leq p(\mathrm{L})$)

Showing a Language is Not Regular

We can use the pumping lemma to show that a language L is not regular as follows:

1. suppose that L *is* regular,
2. in this case L has a pumping length $p(\mathrm{L})$,
3. we don't know what $p(\mathrm{L})$ is, but we can use it to give an algorithm for constructing a carefully chosen word $w \in \mathrm{L}$ with $|w| \geq p(\mathrm{L})$,
4. apply the pumping lemma to get a decomposition for this word $w = x \cdot y \cdot z$,
5. use the *pumping* and/or *bounding* conditions of the pumping lemma to derive a contradiction,
6. conclude that L could not have been regular after all.

Example: A Nonregular Language

The language L whose words have any number of 0s followed by an equal number of 1s is not regular.

1. suppose L is regular,
2. then L has a pumping length p,
3. consider the word $w := 0^p \cdot 1^p$. Observe that $w \in \mathrm{L}$ and $|w| \geq p$,
4. by the pumping lemma we get a decomposition $w = x \cdot y \cdot z$.
5. - By the *pumping condition* we can pump down w to get $w' := x \cdot z \in \mathrm{L}$,
   - by the *bounding condition* $|x \cdot y| \leq p$ so x and y contain only 0s,
   - because $y \neq \varepsilon$, the word w' has fewer 0s than 1s, so $w' \notin \mathrm{L}$.
6. So L can't be a regular language.

Example: Another Nonregular Language

The language L of palindromes is not regular.

1. suppose L is regular,
2. then L has a pumping length p,
3. consider the word $w := 0^p \cdot 1 \cdot 0^p$. Observe that $w \in \mathrm{L}$ and $|w| \geq p$,
4. by the pumping lemma we get a decomposition $w = x \cdot y \cdot z$.
5.
   - By the *pumping condition* we can pump down w to get $w' := x \cdot z \in \mathrm{L}$,
   - by the *bounding condition* $|x \cdot y| \leq p$ so x and y contain only 0s,
   - because $y \neq \varepsilon$, the word w' has fewer 0s before the 1 than after, so $w' \notin \mathrm{L}$.
6. So L can't be a regular language.