“CS 374” Fall 2015 — Infinite Fooling Sets Review Guide

This review intends to help your intuition about *why* the infinite fooling set method is valid. We won’t try to formally prove anything here. This shouldn’t take the place of the lecture slides or course notes, which are much more precise and detailed.

The infinite fooling set method helps you prove that a language is not regular. If you want to prove a language *is regular*, you need to do something else. We’ll discuss that briefly as well.

Never give up, never surrender! 화이팅! Viel Glück! 加油! Allons-y! がんばって！
— CS 374 TAs

When you’re done reading, see also:

- Lab 05: Non-Regularity, September 11, 2015
- CS 374 lecture slides from week of September 7
- Jeff Erickson’s `03-automata.pdf` (“Finite State Machines”). Pay special attention to these sections:
  - 3.8: “Fooling Sets”
  - 3.9: “The Myhill-Nerode Theorem”

Note: In the DFA diagrams in this document, any unmarked transitions can be assumed to lead to a “reject” (inescapable, non-accepting) state.

1 Background info

1.1 The connection between regular languages and finite automata

Based on the recursive definition of a regular language as defined in the lecture and Jeff’s notes, we know these things are true:

- You can always represent a regular language as a regular expression. If you write a valid regular expression, it represents a regular language.
- You can always convert a regular expression to an NFA, and vice versa.
- You can always convert an NFA to a DFA, and vice versa.

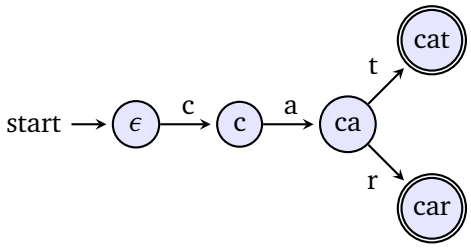
That means you can prove that a language *is regular* by creating a DFA, NFA, or regular expression for it. (Conversely, if you have a properly-formed regular expression, NFA, or DFA, then it describes a regular language.) If you want to prove that a language is *not regular*, you need to prove that none of these constructions can exist for the language. The infinite fooling set method does that.

1.2 Finite states and black-and-white answers

A regular language is a language that can be *decided* by a primitive computer that has a finite amount of states and no memory apart from the unique, finite states themselves (e.g., there is no RAM or stack memory). By "decided" we mean that for a given input string, the machine can determine whether the string is in the language (accept) or not in the language (reject)—there is no ambiguity. Since there is no other kind of output from the machine, the only way it can indicate any difference between two strings is if one is accepted while the other is rejected. Also, the machine *must* eventually halt and make a decision, because a given input string must be finite, and the machine has finite states, so after transitioning a finite number of times, the machine is done processing.

1.3 Finite languages are regular

If a language is a finite set of strings, it's definitely regular. You could build a DFA with enough states to recognize each string in the language. Even if your design is not efficient and you don't combine redundant states where possible, creating a very *large* machine, you only need a finite number of states, so the machine can definitely be built.



1. DFA for $L = \{car, cat\}$ with one more accepting state than needed.

1.4 Infinite languages aren't always regular

If a language can generate infinite acceptable strings, it might not be regular. Based on the regular expression yaa^*s, we could produce strings like yas, $yaas$, $yaaaaaas$, etc. It's a regular language with infinite cardinality. (Intuitively, this is possible because we could build a loop into a DFA somewhere and use the loop as many times as we want to read an arbitrarily-long string using a finite number of states.)[1]

But other infinite-size languages like $L = \{ a^i b^i \mid i \geq 1 \}$ are not regular. A DFA has no memory to track how many a symbols it has seen in order to check for a matching number of b symbols afterward. If you wanted to make a DFA to read this language, you'd need an infinite number of branches in your DFA: at least one branch for each string length $2i$, where i goes to infinity. That's impossible.

The difference between these infinite languages is simple: one kind can be read and decided with a finite automaton, and the other can't. So we need a way to tell whether an infinite language actually requires infinite states to figure out if it's a regular language.

2 How many states do you need?

2.1 Distinguishing suffixes: Accept this and not that

Think of the $L = yaa^*s$ language again: it allows strings like yas, $yaas$, and $yaaaaaas$, but are they really different strings as far as this regular language is concerned? If we build a DFA for yaa^*s, can it

[1] This is the reasoning behind the "pumping lemma" for regular languages. It's not in the syllabus for this course, but you might be interested to look it up.

tell the difference between these strings?

If you take a certain suffix and concatenate it with two different input strings, it might cause one of the strings to be accepted and the other to be rejected. In that case, the strings are "distinguishable" by the suffix. If there isn't *any* suffix that can do this, the strings are considered to be in the same "equivalence class." (Note that just because a *particular* suffix fails to distinguish two strings, it doesn't mean the strings aren't distinguishable! Some other suffix might do it.)

Let's look at some pairs of different prefixes from the language $L = yaa^*s$ and see if there's any suffix that would cause the accept/reject results to differ. Call the prefixes u and v. We'll make up some suffix called x.

$u : y$
$v : ya$
$x : s$
$ux : ys \notin L$
$vx : yas \in L$

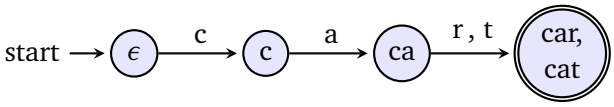
One rejected, one accepted: So y and ya are definitely distinguishable.

$u : ya$
$v : yaa$
$x : s$
$ux : yas \in L$
$vx : yaas \in L$

Both accepted: ya and yaa aren't distinguishable with this particular suffix, but maybe with others. (Actually, they're never distinguishable, but you can't *prove* it with one example.)

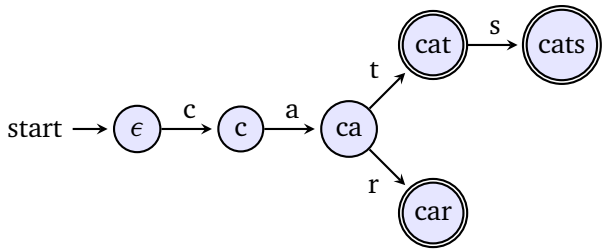
Distinguishing suffixes are important. When you're trying to build a DFA for a language, each time you can show an additional string prefix that can be distinguished from all the others when a certain suffix is added, you've shown that the DFA needs to be designed with at **least one additional state** to recognize the difference. We can summarize it this way:

If two strings cannot be distinguished, it suggests some state(s) can be combined in the DFA. Since the DFA gives the same result for both strings, we can merge one or more states and simplify the DFA without changing the language it accepts. Consider $L = \{cat, car\}$. After reading "ca," the machine can read either a "t" or an "r" and accept, so these options can be allowed on a single transition to an accepting state.



2. Revising the DFA from Fig. 1: In $L = \{car, cat\}$, "car" and "cat" aren't distinguishable, so we can use a combined state for them.

If two strings can be provably distinguished, it means extra state(s) are needed to resolve the difference. Since the DFA's ultimate response to the two prefix strings can change depending on what suffix is read next, the machine needs separate states to allow for this status.



3. DFA for $L = \{car, cat, cats\}$. This time, "car" and "cat" are distinguished by the suffix **-s**. The *car* and *cat* states can't be combined, because that would allow "cars," which isn't in the language.

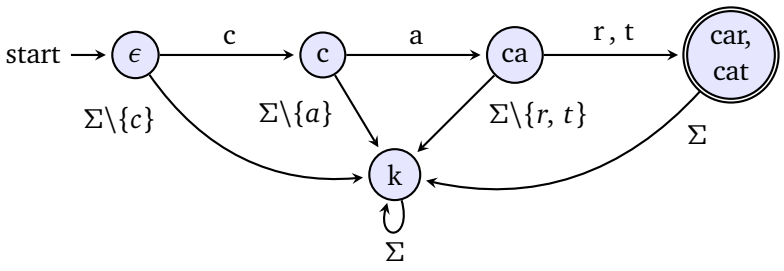
So the more distinguishable strings we can show for a given language, the larger its supposed DFA needs to be. If we had infinite distinguishable strings, the DFA would need infinite states—so no such DFA could exist.

2.2 Fooling sets

A set of strings that are all mutually distinguishable from each other in your language (i.e., after an appropriate suffix is added) is called a **fooling set**. Not every string in a fooling set has to be a string in the language itself! It's just a set of prefixes (or a single non-prefix) that the language can distinguish after the right suffix is added. Here are some things that might be in a fooling set for language L:

- Prefixes of strings in L, such as:
  - The empty string ϵ
  - Whole strings that are in L
  - Only one element of each equivalence class may be included (i.e. you can't have any two strings that are indistinguishable)
- Exactly one string that isn't a prefix of L
  - Can you prove why only one of these can be in the same fooling set?
  - If you're paying attention and keeping score: This entry would account for the inescapable reject state a DFA might have. If you're trying to figure out what the smallest possible DFA would be, that matters. You'll see why in the next section.

Here's an example. For the language $L = \{car, cat\}$, we have the fooling set $S = \{\epsilon, c, ca, car, k\}$, where k is a non-prefix of the language (obviously). This is actually the largest fooling set we can make for that language, and it has **five** elements. Notice something about the smallest DFA for this language: it has **five** states! Also, we can't put both "car" and "cat" in the fooling set at the same time, because all of the entries have to be distinguishable from each other.



4. DFA for $L = \{car, cat\}$ with the minimum number of states. We show the "reject" state explicitly.

2.3 The minimum number of states: To infinity and beyond

The Myhill-Nerode theorem establishes this connection between the maximum size of a fooling set (actually the number of equivalence classes in a language) and the size of a DFA. We can summarize it with this equality:

$$\text{maximum fooling set size} = \text{equivalence classes} = \text{minimum states}$$

If we could prove that there exists an infinite number of distinguishable strings for a language, it would prove that the theoretical DFA would need an infinite number of states to recognize the language—which is impossible! Therefore, the language isn't regular.

Note that we don't actually need to find the maximum fooling set size! Since the size of any fooling set we can show becomes a **lower bound** on the number of states in the DFA, as soon as we've shown any infinite fooling set, we've proven the language non-regular. That is, there might be a "bigger infinity" of states necessary than we've even shown—but hey, infinite is infinite in this case.[2]

Basically, when we find any infinite fooling set, we're saying "Since the maximum fooling set is *at least* as big as the infinite one we've found so far, the DFA certainly has to be infinite in size, which is impossible. The DFA can't exist, so the language isn't regular." So we're not trying to minimize the number of states; we're trying to prove that you can't really minimize the number of states. That's fooling... *with style*.

3 Using the infinite fooling set technique

3.1 Know when to try the technique

Remember that the infinite fooling set technique is intended to prove that a language is non-regular. Before you try to use it, look for clues that the language actually *is* non-regular. You don't want to waste time on an exam. The most obvious clue is anything that would implicitly require a stack (memory) to track in your machine. On the other hand, if a language is finite or can be represented as a regular expression, it's regular!

- $L = \{ a^i b^i \mid i \geq 1 \}$: This language tries to match some number of a and b symbols. There's no way for a DFA to keep track of an *arbitrary* count of a symbols in order to check for the same number of b symbols! **Probably not regular. (Proven below.)**

- $L =$ {strings of properly-matched open and closing parentheses}: E.g., (((())()())). This language tries to check for proper nestings of paired symbols, but to do this propery, the machine would need to track how many levels are currently waiting to be closed as it goes. **Probably not regular. (Proven below).**

- $L = \{ a^i b^i \mid 1 \leq i \leq 100 \}$: This looks similar to the first example, but since the exponent i has a finite upper bound, there are a finite number of possible strings. The DFA would be large, but finite. **Regular!**

- $L = \{(ab)^i (ab)^{i-1} \mid i \geq 1\}$: After some manipulation, you'll see that this is the same as the regular expression $(abab)^* ab$. **Regular!**

[2] It is true that some "infinities" are bigger than others. Remember the difference between countably infinite and uncountably infinite.

3.2 Choosing a fooling set

There are a few guidelines to get started picking an infinite fooling set.

- The set must be infinite, otherwise there's no point! This means you'll probably use a Kleene star or refer to some other infinite set (e.g., the set of all prime numbers).
- The set should contain prefixes from the language that you could predictably "make complete" with a suffix to produce an acceptable string in the language. See the first worked example below. If your language has strings beginning with various characters, remember that you only need to show any *one* infinite fooling set, so you don't need to cover all the possibilities. (Look at the binary palindrome example problem below.)
- In order for the fooling set to be valid, **every** pair of distinct strings in the set must be distinguishable. In order to prove this, you limit the contents of the fooling set in a way that lets you predict what any given pair would look like, then craft a formulaic suffix that could always work to distinguish the pair. **Your careful choice of the fooling set is the only way you can control what any two distinct strings chosen from that set could be.**

3.3 Choosing the suffix formula

Once you have an infinite fooling set in mind, you need to demonstrate that every single pair of distinct strings in the fooling set has some distinguishing suffix. *Otherwise, it's not really a fooling set!* You don't need to find a single finite suffix that always works, but rather a formula or method that could produce some distinguishing suffix for any pair of strings from S that you are given.

The point of this is to establish that the cardinality of the infinite set you gave is truly a lower bound on the number of DFA states. If some of the strings in your set are not distinguishable, then they are in the same equivalence class; then they could be represented with some combined states in the DFA (fewer than you intended) and you haven't proven anything.

Choose the suffix like this: Based on your selection of the fooling set S, you know what form any two strings chosen from S will take. Call those strings u and v. Then design your suffix x so that when it's added to the end of u and v, producing ux and vx, **exactly one** of these two completed strings will be in the language.

4 Examples

1. $L = \{ a^i b^i \mid i \geq 1 \}$

   Solution: Strings in this language begin with a and end with b. Let's pick the infinite set of prefixes $S = \{a^k \mid k \geq 1\} = aa^*$. Then any distinct strings $u, v \in S$ take the form $u = a^i$ and $v = a^j$. Since u and v are distinct, we know $i \neq j$; assume without a loss of generality that $i < j$. Pick the suffix $x = b^i$, designed to "complete" the string u. Then the string $ux = a^i b^i \in L$, while $vx = a^j b^i \notin L$, so all distinct $u, v \in S$ are distinguishable in L, so S is an infinite fooling set for L; therefore L is not regular. ■

2. $L =$ {strings of properly matched open and closing parentheses} E.g., (((())()()). Source: Lab 5.

   Solution: Pick the infinite set of prefixes $S = \left\{ (^k \;\middle|\; k \geq 0 \right\} = (^*$. Then any distinct strings $u, v \in S$ take the form $u = (^i$ and $v = (^j$. Since u and v are distinct, we know $i \neq j$; assume without a loss

of generality that $i < j$. Pick the suffix $x =)^i$. Then the string $ux = (^i)^i \in L$, while $vx = (^j)^i \notin L$, so all distinct $u, v \in S$ are distinguishable in L, so S is an infinite fooling set for L; therefore L is not regular. ■

3. $L = \{$palindromes over the binary alphabet $\Sigma = \{0, 1\}\}$. A palindrome is a string that is equal to its reversal, e.g. *10001* or *0110*. Source: Lab 5.

   Solution: Note that strings in this language might begin with 0 or 1 (and the empty string is in L as well), but we only need to pick one set of prefixes to build our infinite fooling set. Choose the infinite set $S = (01)^*$. Then suppose for distinct $u, v \in S$, we have $u = (01)^i$ and $v = (01)^j$, $i \neq j$. Suppose without a loss of generality that $i < j$. Pick a suffix $x = (10)^i$. Then $ux = (01)^i(10)^i \in L$ but $vx = (01)^j(10)^i \notin L$, so all distinct $u, v \in S$ are distinguishable in L, so S is an infinite fooling set for L; therefore L is not regular. ■

4. $L = \{0^p \mid p$ is a prime number$\}$. Source: the Lecture 6 slides on Non-Regularity.

   Solution: This one is harder. We rely on these facts:

   - There are infinitely many prime numbers. (https://primes.utm.edu/notes/proofs/infinite/euclids.html)
   - There exist pairs of successive prime numbers with arbitrarily large gaps. (https://primes.utm.edu/notes/gaps.html)

   Pick the infinite set $S = 0^*$. Then suppose for distinct $u, v \in S$, we have $u = 0^i$ and $v = 0^j$, $i \neq j$. Suppose without a loss of generality that $i < j$. We need to find a non-negative number k such that exactly one of $i + k$ or $j + k$ is prime.

   Let $d = j - i$. Let p_1 and p_2 be successive primes such that $p_1 \geq i$ and $p_2 - p_1 > d$. (We know such a pair exists because there exist pairs of successive prime numbers with arbitrarily large gaps.) Let $k = p_1 - i$. Then $i + k = p_1$ so $i + k$ is prime. On the other hand, $j + k = (p_1 - i) + (d + i) = p_1 + d$. Note that $p_1 + d < p_2$ based on our choice of p_1 and p_2. Also, since $i < j$, $d > 0$, so $p_1 + d > p_1$. Then overall, $p_1 < p_1 + d < p_2$, so $j + k$ is between our two successive primes, so $j + k$ is not prime.

   Choose the suffix $x = 0^k$. Then $ux = 0^i 0^k = 0^{i+k}$, where $i + k$ is prime; $vx = 0^j 0^k = 0^{j+k}$, where $j + k$ is not prime. Therefore $ux \in L$ and $vx \notin L$, so all distinct $u, v \in S$ are distinguishable in L, so S is an infinite fooling set for L; therefore L is not regular. ■