SE 3310b

Theoretical Foundations of Software Engineering

Week 4:

Closure Properties of Regular Languages. Regular Expressions. Pumping Lemma for Regular Languages.

Aleksander Essex

Western Engineering

Closure Properties of Regular Languages

Closure

Let S be a set and let f be an operation defined for all elements of that set. Closure refers to the situation in f always produces an output that is an element of S, when given elements of S as input.

So for example if $S = \mathbb{Z}$, we say S is closed under addition, since if you only ever add integers, you only will receive an integer as output. On the other hand S is *not* closed under divition: dividing two integers can produce a result that is not, itself, an integer.

Western Engineering

Closure

Definition 1 (Regular Language Closure)**.**

Let $\mathcal{REG}$ be the set of regular language. Let $f(\cdot)$ be an operation that accepts one (or more) langauges and produces a language. We say the set of regular languages $\mathcal{REG}$ is *closed* under $f(\cdot)$ if, for all $L_1 \ldots L_n \in \mathcal{REG}$, it is the case that $f(L_1 \ldots L_n) \in \mathcal{REG}$.

In other words, if $f(\cdot)$ always outputs a regular language given only regular languages as input, we say f is closed under the regular languages. So if the regular languages are closed under e.g., complement, then if $\bar{L}$ is the complement of a regular language L, then $\bar{L}$ is a regular language.

Western Engineering

Regular Language Closures

Regular languages are closed under a wide variety of operations:

- Complement: $L' = \{s : s \notin L\}$
- Reversal: $L' = \{s : \text{reverse}(s) \in L\}$
- Union: $L' = \{s : s \in L_1 \text{ or } s \in L_2\}$
- Intersection: $L' = \{s : s \in L_1 \text{ and } s \in L_2\}$
- Concatenation: $L' = \{st : s \in L_1, t \in L_2\}$
- Kleene star: $L' = \{s^* : s \in L\}$
- and many more...

Western Engineering

Proofs of Closures

Let's prove a few of the more common closures: union, concatenation and Kleene star. The idea is to take two NFAs (or DFAs), and then apply the operation to get a third automaton. There are two things to check we have to do to the result:

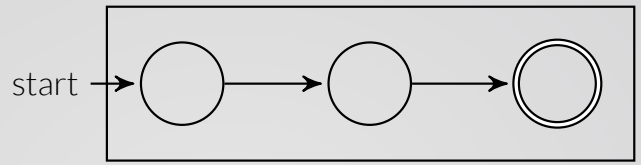
1. does the resulting automaton produce the intended language
2. is the automaton an NFA (or DFA)?

Western Engineering

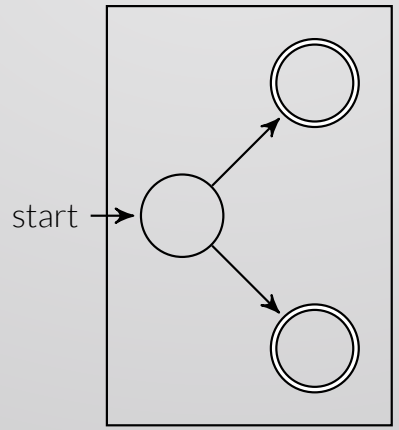
Proofs of Closures

Let's start by considering the following two languages.

L1



L2



Western Engineering

Union of Languages

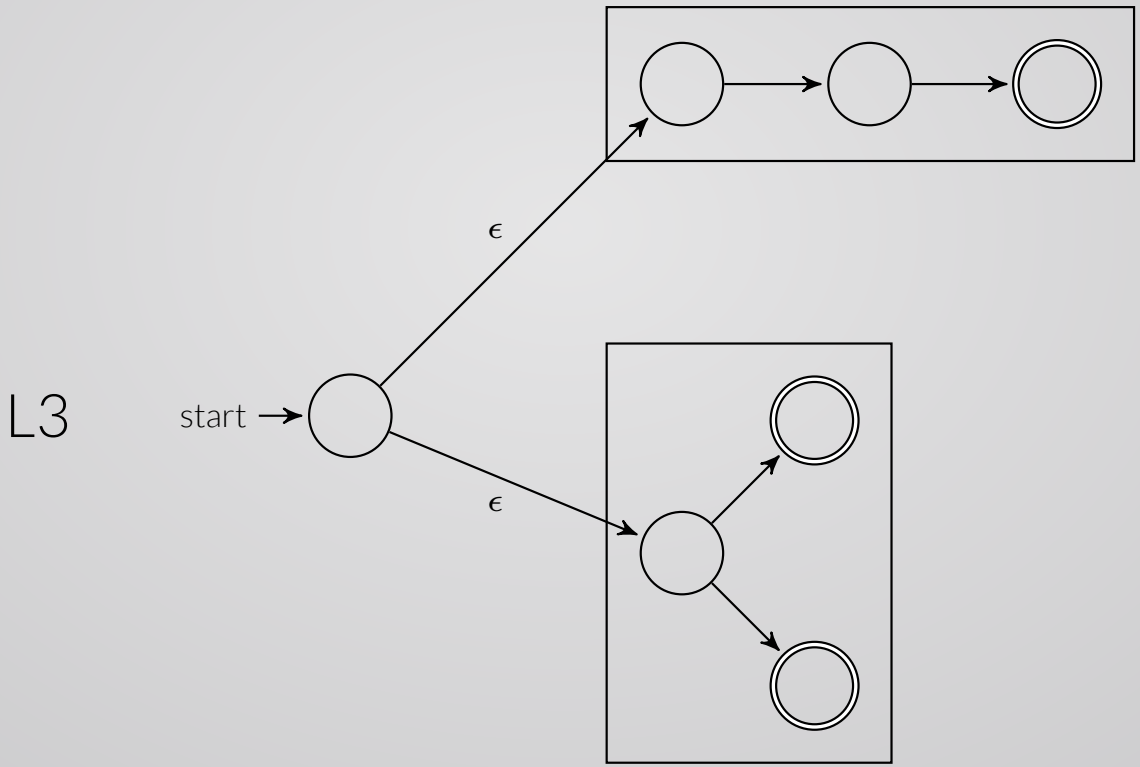
Let $L_3 = L_1 \cup L_2$, i.e., string s is in language L_3 if it's in L_1 or L_2.
Given the NFAs for L_1 and L_2 we can construct L_3 in two steps:

1. Define a new "joint" start state
2. Add ϵ-transitions from the new joint start state into the states of L_1 and L_2 respectively
3. Convert the start states of L_1 and L_2 to non-starting states.

Western Engineering

Union of Languages

The result of $L_3 = L_1 \cup L_2$:



Union of Languages

Sanity check:

1. Does L_3 recognize the union of L_1 or L_2?
2. Is the machine that recognizes L_3 a valid NFA/DFA?

Finally, is our concatenation algorithm a ”black box” solution to any arbitrary pair of languages? That is, does it always work for any regular language? If yes to all three questions, then we can say the regular languages are closed under concatenation.

Western Engineering

Concatenation of Languages

Let $L_4 = L_2 L_1$, i.e., L_4 the set of strings that are the concatenation of a string in L_2 with a string in L_1. Strategy:

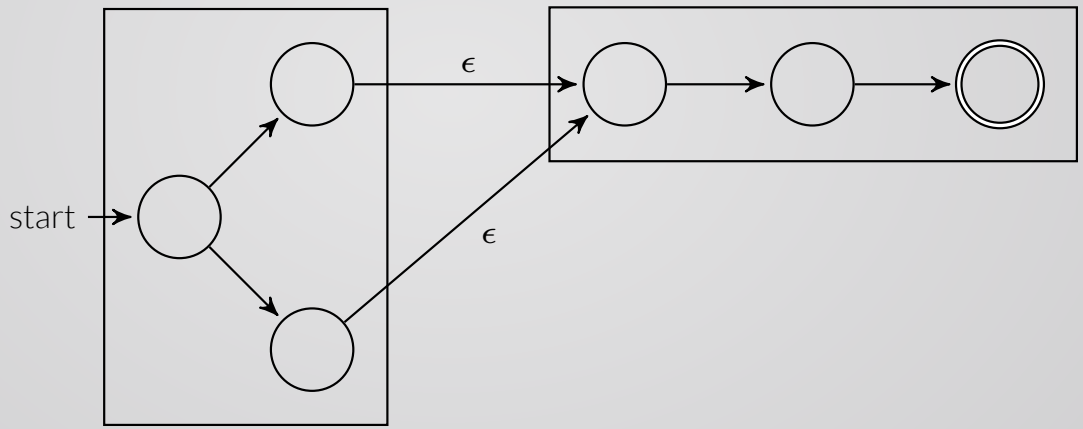
1. Convert all accept states in the first language (L_2 in this case) in to regular states
2. Extend an ϵ-transition from all former accept states in L_2 to the start state of the second language (L_1 in this case). Convert the start state of the second language to a regular state.

Western Engineering

Concatenation of Languages

The result of $L_4 = L_2 L_1$:

L4



Kleene Star

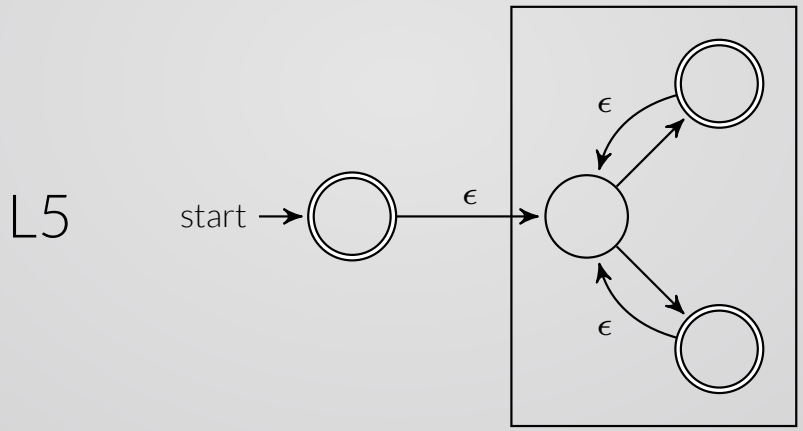
Let $L_5 = L_2^*$, i.e., for all $s \in L_2$, we have $s^* \in L_4$. Strategy:

1. Define a new start state (that also accepts) to handle the 0-length string and have this new state ϵ-transition in to the (former) start state of L_2
2. To handle non 0-length repetitions, make all the accept states of L_2 ϵ-transition back to the former start state of L_2.

Western Engineering

Kleene Star of Languages

The result of $L_5 = L_2^*$:



Regular Expressions

Regular Expression

Definition 2 (Regular Expression)**.**

R is a regular expression if R is:

1. $a \in \Sigma$ (a symbol)
2. ε (the empty string)
3. ϕ (the empty set)
4. $R_1 \cup R_2$ (the union of two REs)
5. $R_1 R_2$ (the concatenation of two REs)
6. R_1^* (an RE repeated 0 or more times)

Western Engineering

Regular Expression

Here are examples of languages as defined by a regular expression:

- 0^*10^*: the set of strings that contain *one* 1.
- $1^*(01^+)^*$: the strings where every zero is followed by *one or more* 1.
- $(\Sigma\Sigma)^*$

We use the notation R^+ to denote RR^*, i.e., the expression R repeated *one* or more times. Following with Sipser, we use the (somewhat lax) notation Σ to denote any single symbol $s \in \Sigma$.

Western Engineering

Regular Expression

Theorem 3.

A language is regular if and only if some regular expression describes it.

Proof. See textbook.

Western Engineering

Pumping Lemma for Regular Languages

Non-regular Languages

So far we have examined regular languages, i.e., the set of languages that can be described by some DFA, NFA or RE. Certainly it would seem to make sense that there are languages for which no DFA/NFA or RE exists to describe them. We call a language of this kind **non-regular**.

Suppose suppose someone gave you the description of a and language asked, "**is it regular, or not?**" How would you decide? How could you convince someone else?

Western Engineering

Non-regular Languages

Consider the language $L = \{0^n 1^n : n \geqslant 0\}$, i.e., the set of all strings beginning with a run of 0's followed by the same number of 1's.

Could you provide a DFA, NFA or RE to recognizes L? It would seem, **no**: in order to be able to tell if the number of 0's matched the number of 1's, you would seem to have to be able to somehow *remember* how many 0's you had seen.

But in order to keep track of an unlimited number of possibilities, you would need an unlimited number of states. But DFAs and NFAs are restricted to finitely many states!

Western Engineering

Proving a Regular Language is Regular

Ok, let's set aside proving a non-regular language is non-regular for a moment. How do we prove a regular language is regular? That's straightforward enough: give a DFA, NFA or RE that recognizes the language.

Western Engineering

Proving a Non-regular Language is Non-regular

How can we prove a given language is non-regular? Maybe the language actually is regular, we just haven't been able to come up with an equivalent DFA, NFA or RE yet. But if the language is non-regular, we might spend forever trying to come up with a DFA that doesn't exist. We need a better way.

Pumping Lemma: The Pumping lemma is a way to prove a non-regular language is, in fact, non-regular. It makes use of a special property of all regular languages, and if you can prove a language does not have this property, then **it is not regular**.

Western Engineering

The Pumping Lemma

Theorem 4 (Pumping Lemma)**.**

If L is a reg. language, then for all string $s \in L$ where $|s| \geqslant p$ (a number called the **pumping length**), then s can be divided in to three parts $s = xyz$ where:

1. $xy^iz \in L$ for $i \geqslant 0$
2. $|y| > 0$ (i.e., |y| must be non-zero length)
3. $|xy| \leqslant p$ (i.e., |xy| must be no greater than the pumping length)

Western Engineering

Pumping Lemma: The Idea

The pumping lemma arises from the pigeonhole principle: if $n + 1$ pigeons occupy n holes, then at least one hole contains more than one pigeon.

Looping through an automaton. Any string $s \in L$ defines a computational path through the automaton. Suppose you can express that automaton as a DFA (or NFA) with p states. If we feed in a string of length p or greater, by the pigeonhole principle we must visit at least one state more than once, implying the existence of a "loop," or cycle. And if there's a loop, then we could build a string that repeats this loop as many times as we want.

Western Engineering

Pumping Lemma: The Idea

By *pumping* we're talking about "pumping" up a string, like inflating a balloon:

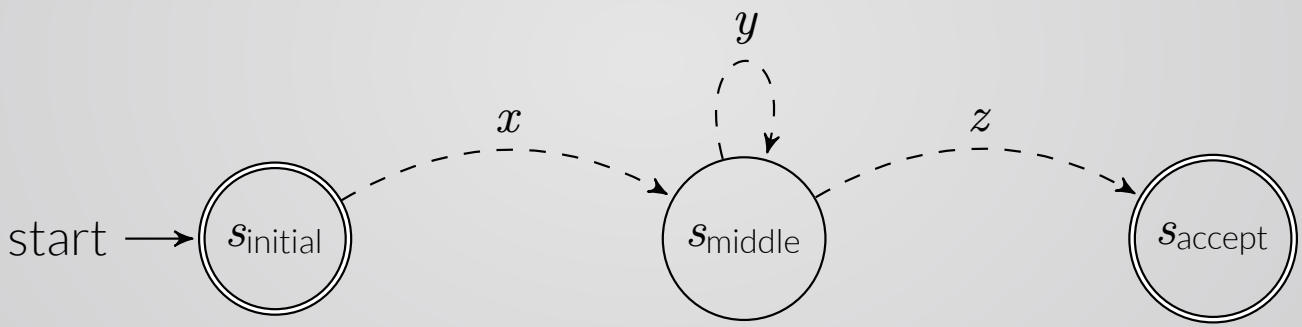
- Each string of a regular language (no smaller than the pumping length) has a middle portion then can be **pumped**
- By *pumped*, we me that this middle portion can be repeated arbitrarily many times, and that the result **will also be in the language**.

Ultimately we're still talking about looping, so "looping lemma" may be a better name for it.

Western Engineering

Visualizing the Pumping Lemma

The pumping lemma says if the language is regular, then for any string long enough, the computational path will have 3 parts:



The idea here is that there exists a segment of the computational path, y (which starts is s_{middle} and returns to s_{middle}) that can be repeated, or "pumped" arbitrarily many times before finally continuing on to the accept state.

Western Engineering

Strategy for Proving L Non-regular

If a regular language must have this middle, repeatable "y" portion, to show L is non-regular, it suffices to show that this y-portion is not present in L. To do this we proceed with a proof by contradiction:

1. Assume L is regular
2. Use pumping lemma to "guarantee" that all strings $s \in L$ that are "long enough" (i.e., $|s| \geqslant p$) can be pumped
3. Find some string $s \in L$ where $|s| \geqslant p$ and show that it **cannot** be pumped.

Western Engineering

Showing a String Cannot Be Pumped

In order to show that a string $s \in L$ *cannot* be pumped, we must consider all possible ways that s can be divided into parts xyz, and for each of then find some $i \geqslant 0$ such that xy^iz is string that **is not** in the language to produce our contradiction.

Recall our string s, the one we use to find a contradiction. We don't actually pick a specific string s, but leave s abstract and focus on its general structure/form:

- We don't assume we know what pumping length p is. We simply define our s such that $|s| \geqslant p$
- We don't examine all possible ways to partition string s into 3 parts, rather since s is of a general form, we look at all the possible *cases* that s could be partitioned.

Western Engineering

Pumping Lemma: An Example

Recall our language $L = \{0^n 1^n : n \geqslant 0\}$. Prove L is non-regular.

Proof:

1. Assume L is regular
2. Let s be a string of the form $0^p 1^p$. Clearly $s \in L$ and $|s| \geqslant p$
3. Let $s = xyz$. If L is regular (as we have assumed), then as per the pumping lemma it is also the case that $xy^i z \in L$ for $i \geqslant 0$
4. Show s **cannot be pumped.** (see next slide)

Western Engineering

Show s cannot be pumped

Now we examine how we can prove $s = 0^p 1^p$ cannot be pumped. To do this we consider all the possible (relevant) cases of partitioning s in to xyz. Ok, let $s = xyz$, and by our definition s contains an equal amount of 0's and 1's. Recall from our pumping lemma definition we require $|y| > 0$ and $|xy| \leqslant p$

Western Engineering

Show s cannot be pumped

Case 1: y contains all 0's: What happens if we pump y? Let $s' = xy^iz$. If $i > 1$, and since $|y| > 0$, then $s' = xy \ldots yz$, i.e., the process of pumping *adds* some non-zero amount of 0's to s'. This means that now s' contains more 0's than $1's$, meaning $s' \notin L$. Similarly if we consider $i = 0$, we have $s' = xz$. We have *taken away* 0's, and thus s' contains fewer

Western Engineering

Show s cannot be pumped

Case 2: y contains all 1's: Similar to Case 1: if we have $s' = xy^iz$ and $i > 1$ then we are *adding* 1's to the string. If we have $i = 0$, we are *removing* 1's from the string. Either way, s' will not have an equal number of 0's and 1's, meaning $s' \notin L$.

Western Engineering

Show s cannot be pumped

Case 3: y contains both 0's and 1's: There are two sub-cases to consider here:

- **Case 3a: y contains an unequal amount of 0's and 1's.** The argument here is similar again to the previous two cases: pumping y will either add (or remove) and unequal amount of 0's and 1's, giving us the same result, i.e., $s' \notin L$

Western Engineering

Show s cannot be pumped

Cont'd:

- **Case 3b: y contains an equal amount of 0's and 1's.** This case is different from Cases 1, 2, and 3a, since pumping y in this case would add (or remove) and equal amount of 0's and 1's to the language. For this case, suppose $y = 0^x1^x$ for some $x > 0$. Let $i > 1$. Then $y^i = \underbrace{yy\ldots y}_{\text{i times}} = \underbrace{0^x1^x0^x1^x\ldots0^x1^x}_{\text{i times}}$.

So even though s' contains the correct number of 0's and 1's, they're out of order! So once again we have $s' \notin L$.

Western Engineering

Pumping Lemma Example (Cont'd)

Our goal was to prove $L = \{0^n 1^n : n \geqslant 0\}$ is non-regular. To that end, we started by assuming L *is* regular. Then we chose a string $s = 0^p 1^p$ satisfying $s \in L$ and $|s| \geqslant p$, and sought to show it cannot be pumped.

From there we considered all possible ways to partition $s = xyz$ and, for each case, showed the existence of some $i \geqslant 0$ for which $s = xy^i z \notin L$.

Western Engineering

Pumping Lemma Example (Cont'd)

What does that all mean? Well, if L was regular, then as per the pumping lemma *all* strings in L must have this "pumpable" y portion.

Yet by showing that our sting s cannot not be pumped *no matter how we slice it*, we have proved s does not have the requisite property to be in L. Yet clearly s is in L.

This is a contradiction. Therefore L **is not regular**. □

Western Engineering

Pumping Lemma FAQ

As you study the pumping lemma it is important for you to understand exactly what it proves (or does not prove).

Q: I found a string in my language and showed that, for all possible ways of partitioning the string into 3 parts, I could pump it such that the resulting string was not in the language. What have I proved?

A: By way of a contradiction, you have proved the language is non-regular.

Western Engineering

Pumping Lemma FAQ Cont'd

Q: I found a string in my language and showed it could be pumped for some (or even all) ways of partitioning it. Does that prove anything?

A: No, because it could mean one of 3 possibilities:

1. The language is regular, in which case all strings are pump-able,
2. The language is non-regular, and there exist strings in the language that cannot be pumped, but you just haven't found one yet, or,
3. All the strings in the language can be pumped, yet the language is non-regular. Indeed such languages do exist, and you would require a different proof technique, e.g., the Myhill–Nerode theorem. This theorem, however, is outside the scope of this course.

Western Engineering

Pumping Lemma Summary

In summary, the pumping lemma for regular languages is the fact that:

> *All sufficiently long strings in a regular language contain a substring that can be repeated arbitrarily many times, (and is) usually used to prove (by way of contradiction) that certain languages are not regular.*

Finally, if the pumping lemma feels complicated to you (and it kind of is), you may enjoy reading this blog post: I hate the Pumping Lemma.

Western Engineering