MA/CSSE 474

Theory of Computation

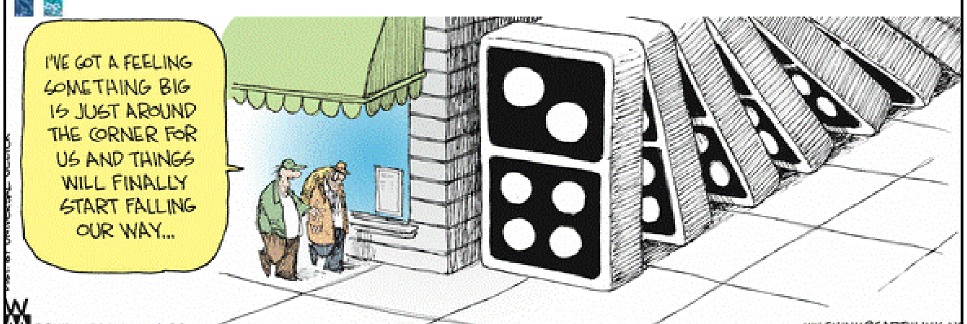
DFSM to RE, Part 2

Closures

Pumping Theorem Intro

Your Questions?

- Previous class days' material
- Reading Assignments
- HW 6 or 7 problems
- Exam 1 questions
- Anything else



Recap: Kleene's Theorem

Finite state machines and regular expressions define the same class of languages.

To prove this, we showed:

***Theorem**:* Any language that can be defined by a regular expression can be accepted by some FSM and so is regular. **Done Day 11.**

Theorem: Every regular language (i.e., every language that can be accepted by some DFSM) can be defined with a regular expression. **Done Day 12**

Recap: DFSM→Reg. Exp.

- R_{ijk} is the set of all strings that take M from q_i to q_j without passing through any intermediate states numbered higher than k.

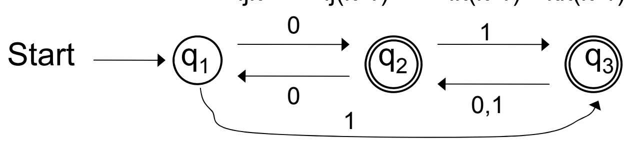
  It can be computed recursively:
- Base cases (k = 0):
  - If $i \neq j$, $R_{ij0} = \{a \in \Sigma : \delta(q_i, a) = q_j\}$
  - If $i = j$, $R_{ii0} = \{a \in \Sigma : \delta(q_i, a) = q_i\} \cup \{\varepsilon\}$
- Recursive case (k > 0):

  R_{ijk} is $R_{ij(k-1)} \cup R_{ik(k-1)}(R_{kk(k-1)})^*R_{kj(k-1)}$
- We showed by induction that each R_{ijk} is defined by some regular expression r_{ijk}.

DFA→Reg. Exp. Proof pt. 3

- We showed by induction that each R_{ijk} is defined by some regular expression r_{ijk}.
- In particular, for all $q_j \in A$, there is a regular expression r_{1jn} that defines R_{1jn}.
- Then $L(M) = L(r_{1j_1n} \cup \ldots \cup r_{1j_pn})$, where $A = \{q_{j_1}, \ldots, q_{j_p}\}$

An Example (r_{ijk} is $r_{ij(k-1)} \cup r_{ik(k-1)}(r_{kk(k-1)})^* r_{kj(k-1)}$)



	k=0	k=1	k=2
r_{11k}	ε	ε	(00)*
r_{12k}	0	0	0(00)*
r_{13k}	1	1	0*1
r_{21k}	0	0	0(00)*
r_{22k}	ε	ε ∪ 00	(00)*
r_{23k}	1	1 ∪ 01	0*1
r_{31k}	∅	∅	(0 ∪ 1)(00)*0
r_{32k}	0 ∪ 1	0 ∪ 1	(0 ∪ 1)(00)*
r_{33k}	ε	ε	ε ∪(0 ∪ 1)0*1

Aside: Regular Expressions in Perl

Syntax	Name	Description
abc	Concatenation	Matches a, then b, then c, where a, b, and c are any regexs
$a \mid b \mid c$	Union (Or)	Matches a or b or c, where a, b, and c are any regexs
a*	Kleene star	Matches 0 or more a's, where a is any regex
a+	At least one	Matches 1 or more a's, where a is any regex
a?		Matches 0 or 1 a's, where a is any regex
a{n, m}	Replication	Matches at least n but no more than m a's, where a is any regex
a*?	Parsimonious	Turns off greedy matching so the shortest match is selected
a+?	"	"
.	Wild card	Matches any character except newline
^	Left anchor	Anchors the match to the beginning of a line or string
$	Right anchor	Anchors the match to the end of a line or string
[a-z]		Assuming a collating sequence, matches any single character in range
[^a-z]		Assuming a collating sequence, matches any single character not in range
\d	Digit	Matches any single digit, i.e., string in [0-9]
\D	Nondigit	Matches any single nondigit character, i.e., [^0-9]
\w	Alphanumeric	Matches any single "word" character, i.e., [a-zA-Z0-9]
\W	Nonalphanumeric	Matches any character in [^a-zA-Z0-9]
\s	White space	Matches any character in [space, tab, newline, etc.]

Regular Expressions in Perl

Syntax	Name	Description
\S	Nonwhite space	Matches any character not matched by \s
\n	Newline	Matches newline
\r	Return	Matches return
\t	Tab	Matches tab
\f	Formfeed	Matches formfeed
\b	Backspace	Matches backspace inside []
\b	Word boundary	Matches a word boundary outside []
\B	Nonword boundary	Matches a non-word boundary
\0	Null	Matches a null character
\nnn	Octal	Matches an ASCII character with octal value nnn
\xnn	Hexadecimal	Matches an ASCII character with hexadecimal value nn
\cX	Control	Matches an ASCII control character
\char	Quote	Matches char; used to quote symbols such as . and \
(a)	Store	Matches a, where a is any regex, and stores the matched string in the next variable
\1	Variable	Matches whatever the first parenthesized expression matched
\2		Matches whatever the second parenthesized expression matched
…		For all remaining variables

Examples

Email addresses

```
\b[A-Za-z0-9_%-]+@[A-Za-z0-9_%-]+(\.[A-Za-z]+){1,4}\b
```

WW

```
^([ab]*)\1$
```

Duplicate words

Find them

```
\b([A-Za-z]+)\s+\1\b
```

Delete them

```
$text =~ s/\b([A-Za-z]+)\s+\1\b/\1/g;
```

How Many Regular Languages?

- Given an alphabet, Σ, how many different languages over Σ? How many of those languages are regular?
- Background: since
  - Σ is finite,
  - each string in Σ* is finite, and
  - there is no limit to the length of the strings in Σ*,

  the number of different strings in Σ* is countably infinite (think about how to enumerate them).
- Is the set of subsets of Σ* countable?
- It suffices to work with Σ = {a}, a single-symbol alphabet.

How Many Regular Languages?

Theorem: The number of regular languages over any nonempty alphabet Σ is countably infinite .

Proof:

- Upper bound on number of regular languages: number of DFSMs (or regular expressions).

- Lower bound on number of regular languages:

  {a},{aa},{aaa},{aaaa},{aaaaa},{aaaaaa},…

  are all regular. That set is countably infinite.

Are Regular or Nonregular Languages More Common?

There is a countably infinite number of regular languages.

There is an uncountably infinite number of languages over any nonempty alphabet Σ.

So there are *many* more nonregular languages than regular ones.

Languages: Regular or Not?

Recall our intuition:

a*b* is regular. $A^nB^n = \{a^nb^n : n \geq 0\}$ is not.

$\{w \in \{a, b\}^*$: every a is immediately followed by b$\}$ is regular.

$\{w \in \{a, b\}^*$: every a has a matching b somewhere$\}$ is not regular.

How do we

- show that a language is regular?
- show that a language is not regular?

Showing that a Language is Regular

Theorem: Every finite language L is regular.

Proof: If L is the empty set, then it is defined by the regular expression $\varnothing$ and so is regular.

If L is a nonempty finite language, composed of the strings $s_1, s_2, \ldots s_n$ for some positive integer n, then it is defined by the regular expression:

$s_1 \cup s_2 \cup \ldots \cup s_n$

Finiteness - Theoretical vs. Practical

Every finite language is regular.
The size of the language doesn't matter.

Parity Checking ⟷ Soc. Sec. # Checking

But, from an implementation point of view, it matters!.

When is an FSM a good way to encode the facts about a language?

FSM's are good at looking for repeating patterns. They don't help much when the language is just a set of unrelated strings.

To Show that a Language L is Regular

We can do any of the following:

Construct a DFSM that accepts L.

Construct a NDFSM that accepts L.

Construct a regular expression that defines L.

Construct a regular grammar that generates L.

Show that there are finitely many equivalence classes under $\approx_L$.

Show that L is finite.

Use one or more of the closure properties.

Closure Properties of Regular Languages

- Union
- Concatenation
- Kleene Star
- Complement
- Intersection
- Difference
- Reverse
- Letter Substitution

The first three are easy: definition of regular expressions.

We will give the ideas of how to do Complement and Reverse.

Intersection: HW5, or ...

You should read about Letter Substitution.

Don't Try to Use Closure Backwards

One Closure Theorem:

If L_1 and L_2 are regular, then so is

$L = L_1 \cap L_2$

But if $L_1 \cap L_2$ is regular, what can we say about L_1 and L_2?

$\underline{L} = L_1 \cap L_2$

ab = ab $\cap$ (a $\cup$ b)* (L$_1$ and L$_2$ are regular)

ab = ab $\cap$ {a^{n}b^n, $n \geq 0$} (may not be regular)

Don't Try to Use Closure Backwards

Another Closure Theorem:

If L_1 and L_2 are regular, then so is

$L = L_1\ L_2$

But if L_2 is not regular, what can we say about L?

$L = \quad L_1\ \ L_2$

$\{aba^n b^n : n \geq 0\} = \{ab\}\ \{a^n b^n : n \geq 0\}$

$L(aaa^*) = \{a\}^*\ \{a^p : p \text{ is prime}\}$

Showing that a Language is Not Regular

Every regular language can be accepted by some FSM M.

M can only use a finite amount of memory to record essential properties.

Example:
$A^nB^n = \{a^n b^n, n \geq 0\}$ is not regular

Showing that a Language is Not Regular

The only way to generate/accept an infinite language with a finite description is to use:

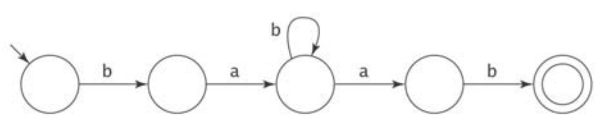
- Kleene star (in regular expressions), or
- cycles (in automata).

This forces a simple repetitive cycle within the strings.

Example:
ab*a generates aba, abba, abbba, abbbba, etc.

Example:
$\{a^n : n \geq 1$ is a prime number$\}$ is not regular.

Exploiting the Repetitive Property



If an FSM with n states accepts at least one string of length $\geq n$, how many strings does it accept?

L = bab*ab

$\underbrace{\text{b a}}_{x}\ \underbrace{\text{b}}_{y}\ \underbrace{\text{b b b a b}}_{z}$

xy^*z must be in L.

So L includes: baab, babab, babbab, babbbbbbbbbbab

Theorem – Long Strings

Theorem: Let $M = (K, \Sigma, \delta, s, A)$ be any DFSM. If M accepts any string of length $|K|$ or greater, then that string will force M to visit some state more than once (thus traversing at least one loop).

Proof: M must start in one of its states.
Each time it reads an input character, it visits some state. So, in processing a string of length n, M does a total of $n + 1$ state visits.

If $n+1 > |K|$, then, by the pigeonhole principle, some state must get more than one visit.

So, if $n \geq |K|$, then M must visit at least one state more than once.

The Pumping Theorem* for Regular Languages

If L is regular, then every long string in L is "pumpable".
Formally, if L is regular, then

$\exists k \geq 1$ such that
 $(\forall$ strings $w \in L$,
  $(|w| \geq k \rightarrow$
   $(\exists\ x, y, z\ (w = xyz,$
    $|xy| \leq k,$
    $y \neq \varepsilon$, and
    $\forall q \geq 0\ (xy^q z$ is in L)))))

Write this in contrapositive form

- a.k.a. **"the pumping lemma"**.
  We will use the terms interchangeably.
- What if L has *no* strings whose lengths are greater than k?

Use The Pumping Theorem to show that L is not Regular:

We use the contrapositive of the theorem:

If some long enough string in L is not "pumpable", then L is not regular.

What we need to show in order to show L non-regular:

$(\forall k \geq 1$

$(\exists$ a string $w \in L$

$(|w| \geq k$ *and*

$(\forall x, y, z\ ((w = xyz \wedge |xy| \leq k \wedge y \neq \varepsilon) \rightarrow$

$\exists q \geq 0\ (xy^qz \notin L)))))$

$\rightarrow$ L is not regular .

Before our next class meeting:
Be sure that you are convinced that this really is the contrapositive of the pumping theorem.

A way to think of it: adversary argument (following J.E. Hopcroft and J.D.Ullman)

1. Choose the language L you want to prove non-regular.
2. The "adversary" picks k, the constant mentioned in the theorem. We must be prepared for any positive integer to be picked, but once it is chosen, the adversary cannot change it.
3. We select a string $w \in L$ (whose length is at least k) that cannot be "pumped".
4. The adversary breaks w into w=xyz, subject to the constraints $|xy| \leq k$ *and* $y \neq \varepsilon$. Our choice of w must take into account that any such x and y can be chosen.
5. All we must do is produce a single number $q \geq 0$ such that $xy^qz \notin L$.

Note carefully what we get to choose and what we do not get to choose.

Example: $\{a^n b^n : n \geq 0\}$ is not Regular

k is the number from the Pumping Theorem.
We don't get to choose it.

Choose *w* to be $a^{\lceil k/2 \rceil} b^{\lceil k/2 \rceil}$ ("long enough").

```
        1                   2
a a a a a … a a a a a | b b b b  … b b b b b b
          x              y            z
```

Adversary chooses *x*, *y*, *z* with the required properties:

$|xy| \leq k$,

$y \neq \varepsilon$,

We must show $\exists\, q \geq 0\ (xy^q z \notin L)$.

Three cases to consider:

- *y* entirely in region 1:
- *y* partly in region 1, partly in 2:
- *y* entirely in region 2:

For each case, we must find at least one value of q that takes $xy^q z$ outside the language L.
The most common q values to use are q=0 and q=2.

A Complete Proof

We prove that $L = \{a^n b^n : n \geq 0\}$ is not regular

If *L* were regular, then there would exist some *k* such that any string *w* where $|w| \geq k$ must satisfy the conditions of the theorem. Let $w = a^{\lceil k/2 \rceil} b^{\lceil k/2 \rceil}$. Since $|w| \geq k$, *w* must satisfy the conditions of the pumping theorem. So, for some *x*, *y*, and *z*, $w = xyz$, $|xy| \leq k$, $y \neq \varepsilon$, and $\forall q \geq 0$, $xy^q z$ is in *L*. We show that no such *x*, *y*, and *z* exist. There are 3 cases for where *y* could occur: We divide *w* into two regions:

```
aaaaa…..aaaaaa    | bbbbb…..bbbbbb
             1    |        2
```

So *y* can fall in:

- (1): $y = a^p$ for some *p*. Since $y \neq \varepsilon$, *p* must be greater than 0. Let $q = 2$. The resulting string is $a^{k+p} b^k$. But this string is not in *L*, since it has more a's than b's.
- (2): $y = b^p$ for some *p*. Since $y \neq \varepsilon$, *p* must be greater than 0. Let $q = 2$. The resulting string is $a^k b^{k+p}$. But this string is not in *L*, since it has more b's than a's.
- (1, 2): $y = a^p b^r$ for some non-zero *p* and *r*. Let $q = 2$. The resulting string will have interleaved a's and b's, and so is not in *L*.

There exists one long string in *L* for which no pumpable *x*, *y*, *z* exist. So *L* is not regular.

What You Should Write (read these details later)

We prove that $L = \{a^n b^n : n \geq 0\}$ is not regular

Let $w = a^{\lceil k/2 \rceil} b^{\lceil k/2 \rceil}$. (If not completely obvious, as in this case, show that *w* is in fact in *L*.)

aaaaa.....aaaaaa| bbbbb.....bbbbbb
1 | 2

There *are three possibilities for y*:

- (1): $y = a^p$ for some p. Since $y \neq \varepsilon$, p must be greater than 0. Let $q = 2$. The resulting string is $a^{k+p}b^k$. But this string is not in *L*, since it has more a's than b's. .
- (2): $y = b^p$ for some p. Since $y \neq \varepsilon$, p must be greater than 0. Let $q = 2$. The resulting string is $a^k b^{k+p}$. But this string is not in *L*, since it has more b's than a's.
- (1, 2): $y = a^p b^r$ for some non-zero p and r. Let $q = 2$. The resulting string will have interleaved a's and b's, and so is not in *L*.

Thus *L* is not regular.

A better choice for w

Second try. A choice of w that makes it easier:

Choose *w* to be $a^k b^k$
(We get to choose any *w* whose length is *at least k*).

$$\overbrace{\underbrace{a\,a\,a\,a\,a \ldots a\,a}_{x}\,\underbrace{a\,a\,a}_{y}}^{1}\;|\;\overbrace{b\,b\,b\,b \ldots b\,b\,b\,b\,b\,b}^{2}$$
z

We show that there is no *x*, *y*, *z* with the required properties:

$|xy| \leq k$,
$y \neq \varepsilon$,
$\forall\, q \geq 0\ (xy^q z \text{ is in } L)$.

Since $|xy| \leq k$, *y* must be in region 1. So $y = a^p$ for some $p \geq 1$. Let $q = 2$, producing:

$$a^{k+p}b^k$$

which $\notin L$, since it has more a's than b's.

We only have to find **one** q that takes us outside of L.

Recap: Using the Pumping Theorem

If L is regular, then every “long” string in L is pumpable.

To show that L is not regular, we find one string that isn’t.

To use the Pumping Theorem to show that a language L is not regular, we must:

1. Choose a string w where $|w| \geq k$. Since we do not know what k is, we must describe w in terms of k.
2. Divide the possibilities for y into a set of equivalence classes that can be considered together.
3. For each such class of possible y values where $|xy| \leq k$ and $y \neq \varepsilon$:

   Choose a value for q such that xy^qz is not in L.