CSE 105
Theory of Computation

Professor Jeanne Ferrante
Today’s Agenda

• $\delta^*$ for DFA’s: Useful Notation
• Regular Expressions
• Non-Regular Languages
  • Pigeon-Hole Principle
  • Pumping Lemma
Definition of $\delta^*$ for DFA’s

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, with $
\delta: Q \times \Sigma \rightarrow Q$.

We define $\delta^*: Q \times \Sigma^+ \rightarrow Q$ as follows:

If $a \in \Sigma$, then
$$\delta^*(q, a) = \delta(q, a)$$

If $s = aw$, with $a \in \Sigma$, and $|w| \geq 1$, then
$$\delta^*(q, s) = \delta^*(\delta(q, a), w)$$
Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Then a string $w$ is accepted by $M$ IFF $\delta^*(q_0, w) \in F$. This statement is:

A. TRUE
B. FALSE
C. Don’t Know

- Note that this is a useful way of describing acceptance of a string in a DFA without referring to the intermediate states.
- Can be used in proofs on HW and Exams.
Extremely useful

Regular Expressions

Another way to describe Languages
A Regular Expression is one of the following:

• Any symbol \( a \) from a finite set \( \Sigma \) (the alphabet)
• The “empty string” \( \varepsilon \)
• The empty set symbol \( \emptyset \)
• \((R1 \cup R2)\)  
  • where \( R1 \) and \( R2 \) are regular expressions
• \((R1 \circ R2)\) (or shorthand, \( R1 \rightarrow R2 \))  
  • where \( R1 \) and \( R2 \) are regular expressions
• \((R1)^*\)  
  • where \( R1 \) is a regular expression
Whoa!! How can we define a regular expression in terms of regular expressions? Isn’t that a contradiction?

• It’s okay!
• The reason is, the definition gives us a “roadmap” we can use for forming larger regular expressions out of smaller regular expressions
• This is an example of a definition by induction (or recursion)
• Many computational problems are defined this way
A Regular Expression $R$ is “shorthand” for a Language $L$

<table>
<thead>
<tr>
<th>Expression $R$</th>
<th>Language $L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>${\varepsilon}$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$ (the empty set)</td>
</tr>
<tr>
<td>$(R_1 \cup R_2)$</td>
<td>$L(R_1) \cup L(R_2)$</td>
</tr>
<tr>
<td>$(R_1 \circ R_2)$</td>
<td>$L(R_1) \circ L(R_2)$</td>
</tr>
<tr>
<td>$(R_1)^*$</td>
<td>$(L(R_1))^*$</td>
</tr>
<tr>
<td></td>
<td>(note that this contains $\varepsilon$, even if $L(R_1)$ doesn’t)</td>
</tr>
</tbody>
</table>

To find $L$ from $R$, work from inner to outer parentheses.
Examples

Regular Expression: Language:

\[(a \cup b)\] \{a,b\}

\[(a \cup b)^*\] \{w | w is 0 or more a’s and b’s\}

\[(a \cup \varepsilon)^* \circ b\] \{w | w is 0 or more a’s, ending in one b\}

Cautions:

• Parentheses are often omitted. When in doubt, do star (*) first, then concatenation (\(\circ\)), then union (\(\cup\))
• The concatenation symbol is often dropped in practice, so ab is used instead of a \(\circ\) b
• \(\varepsilon\) and \(\emptyset\) are different types, and have different effects: 1\(\varepsilon\) is \{1\}, but 1 \(\emptyset\) is \(\emptyset\)
• The notation + is often used to denote “1 or more”
• The alphabet symbol \(\Sigma\) is often used (e.g. \(\Sigma^*\))
• + is often used for U (see HW 3)
Which of the following is NOT a regular expression $(\Sigma = \{0, 1\})$?

A. $(\Sigma \Sigma \Sigma)$ *
B. $\varepsilon \varepsilon$
C. $\Sigma + 1$
D. $1^* \emptyset$ *
E. None or More than one of the above
• Let L be the language of this regular expression: $1^*0$
• Which of the following is NOT in L?
A. 10
B. 100
C. 110
D. More than 1 or none are NOT in L
Regular Expressions

Let $L$ be the language of this regular expression:

$((a \cup \emptyset) \ b^*)^*$

Which of the following is NOT true of $L$?

A. Some strings in $L$ have equal numbers of $a$’s and $b$’s
B. All strings in $L$ have more $b$’s than $a$’s
C. $L$ contains “aaaaaaaa”
D. $a$’s never follow $b$’s in any string in $L$
E. More than one of the above
Regular Expressions Describe Regular Languages!

**THM 1.54:** A language \( L \) is Regular IFF some regular expression describes \( L \).

**Lemma 1.55:** If a language \( L \) is described by a regular expression, then \( L \) is regular.

**Proof:** Let \( R \) be the regular expression describing \( L \). We show how to convert \( R \) into an NFA that recognizes \( L \), following the inductive definition of regular expressions.

**Case 1:** \( R = a \): Then \( L \) is \( \{a\} \), and the NFA to recognize \( L \) is

![NFA for a]

**Case 2:** \( R = \varepsilon \): \( L \) is \( \{\varepsilon\} \), and the NFA to recognize \( L \) is

![NFA for \( \varepsilon \)]

**Case 3:** \( R = \emptyset \): Then \( L \) is the empty set, and the NFA is

![NFA for \( \emptyset \)]
Regular Expressions to NFA’s, Continued

Case 4: $R = (R_1 U R_2)$: Let $N_1$ be NFA corresponding to $R_1$, and $N_2$ be NFA corresponding to $R_2$. We get a new NFA $N$ describing $L_1 U L_2$ using Th. 1.25 Sipser (Closure of Regular languages under $U$)

Case 5: $(R_1 \circ R_2)$: *Follows from Th. 1.47 Sipser (Closure of Regular languages under $\circ$)*

Case 6: $R_1^*$: *Follows from Th. 1.49 Sipser (Closure of Regular languages under $*$)*
Example: Regular Expression to NFA

Given the regular expression (01*), use Th. 1.54 to convert to NFA:
Lemma 1.60: If a language is regular, then it is described by a regular expression.

- Need to know above fact, but **not proof**
- Proof uses a *Generalized NFA (GNFA)*
  - GNFA allows *regular expressions* on edges, and
  - Reads a string of symbols, and
  - Can take the edge if the string is described by the regular expression on the edge
  - Accepts input string if after reading all input, ends in final state
- Proof converts DFA to GNFA’s with fewer and fewer states to get single regular expression
- We won’t be using GNFA’s
One More Regular Expression

Let L be the language of this regular expression: \((1 \ (0 \cup 1^*))^*\)

Which of the following is true of L?

A. \(\varepsilon\) is in L
B. L contains “00”
C. 1’s are never followed by 0’s in any string in L
D. More than one or none of the above
Regular Expressions in Software Tools

- First phase of Compiler: Transform Strings to Tokens

- Tokens can include
  - Keywords: if then end
  - Operators: < > =
  - Identifiers: x v variablename
  - Literals: decimal integer string character

- One Regular Expression for Each Token type

- Flex, Ragel
  - Example tools for creating a lexical analyzer
  - Based on regular expressions

- Variants of RE used in lots of other software tools
  - Perl, Python, Java, Ruby...
On beyond Regular! Or is that all there is?

NON-REGULAR LANGUAGES
Def: A Non-Regular Language is a set of strings that is *not* the language of any DFA.

To show a language L over a finite alphabet is Non-Regular we can:
A. Try a few DFA’s and prove none of them recognize L
B. Show there is an NFA that recognizes L
C. Try a few regular expressions and prove none of them describe L
D. Prove L is infinite
E. None of the Above
We know Regular Languages are given by Regular Expressions (or NFA’s or DFA’s)

Ex: A Regular Expression over \{a, b\} for the set \{w \mid w \text{ starts with 5}a\text{’s and ends with 5}b\text{’s}\} is aaaaa (a \cup b)^* bbbbb.

What about \{w \mid w = a^nb^n \text{ for } n \geq 0\}?

Discuss with your neighbors whether you think there is a Regular Expression (or a DFA or NFA) that can describe this set, and summarize your conclusion.

A. YES
B. NO
C. Don’t know.

NO! DFA’s can only “remember” bounded amount of info!
For the birds
PIGEONHOLE PRINCIPLE
What is the length of the longest string this DFA can accept without visiting any state more than once?

A. 1
B. 3
C. 4
D. 5
E. None of the above
Generalizing:

Given a string $s$ in $L$, and a DFA $M$ that recognizes $L$:

If $|s| \geq |Q|$, then when $M$ reads $s$, the sequence of states visited must be of length at least $|Q| + 1$

In this sequence of states, one (or more) state(s) must appear more than once \textit{(by pigeon hole principle)}

Let’s consider one state that is being visited twice
Generalizing (continued):

Let’s consider one state that is being visited twice. We can write \( s = xyz \)

Questions: What happens in the DFA if you
Input \( xz \)
Input \( xyyyyz \)
Discuss with your neighbors
Any Regular Language must satisfy!

THE PUMPING LEMMA
Th. 1.70: Pumping Lemma for Regular Languages

For each regular language $L$
There is a pumping length $p$ for $L$, such that
For every string $s$ in $L$ of length $\geq p$:
1. $s$ may be divided into three pieces, $x,y,z$, $s = xyz$
2. $|y| > 0$ and $|xy| \leq p$
3. For each $i \geq 0$, $xy^iz$ is in $L$.

(For regular languages $L$, you can “pump out” elements of $L$: $xy^iz$, $i \geq 0$)

Question: For Reg Ex 0(101)*1: What are $x$, $y$ and $z$?
On beyond Regular!

**USING THE PUMPING LEMMA**

**PROVING A GIVEN LANGUAGE IS NOT REGULAR**
Review: Proof of S by Contradiction

• We start by assuming Statement S is FALSE
  • This is the same as NOT(S) is TRUE
• From our assumption NOT(S), we correctly derive a logically false consequence
• This false consequence is the Contradiction
• Conclusion: Our assumption, NOT(S) is TRUE is wrong, so S must be TRUE
We’ll use the Pumping Lemma to show a given language $L$ is **not** Regular

For each regular language $L$

[We assume $L$ regular and hope to get a contradiction]

There is a pumping length $p$ for $L$

[Pumping lemma gives you a number $p$ for $L]

For every string $s$ in $L$ of length $\geq p$

[You wisely choose a string $s$ in $L$ at least as long as $p$]

There are strings $x, y, z$ with $s = xyz$, $|y| > 0$, $|xy| \leq p$

[Given by the pumping lemma]

For each $i \geq 0$, $xy^i z$ is in $L$.

[You choose an $i$ that leads to contradiction!
Therefore the assumption was false, $L$ is NOT regular]
Your Script
“|s| ≥ p. I think you’ll really like it.”

“Excellent. I’m giving you this string s that I made using the pumping length p you gave me. It is in L and |s| ≥ p. I think you’ll really like it.”

“Hm. I followed your directions for xyz, but when I [copy y N times or delete y], the new string is NOT in L! What happened to your 100% Warranty??!”

Pumping Lemma’s Script
“|y| > 0 and |xy| ≤ p. Also, I make you this 100% Lifetime Warranty: you can remove y, or copy it as many times as you like, and the new string will still be in L, I promise!”

“Great string, thanks. I’ve cut s up into parts xyz for you. I won’t tell you what they are exactly, but I will say this: |y| > 0 and |xy| ≤ p. Also, I make you this 100% Lifetime Warranty: you can remove y, or copy it as many times as you like, and the new string will still be in L, I promise!”

“Well, then L wasn’t a Regular Language. Since you lied, the Warranty was void. Thanks for playing.”

“Is L Regular? In this shop I only work on Regular Languages.”

“Good. For the Regular language L that you’ve given me, I pick this nice pumping length I call p.”

“Is L Regular? In this shop I only work on Regular Languages.”
Your Script
“`I’m giving you a language L.”`

“`Uh…let’s just say it’s Regular.”`

“`Excellent. I’m giving you this string s that I made using the pumping length p you gave me. It is in L and |s| ≥ p. I think you’ll really like it.”`

“`Hm. I followed your directions for xyz, but when I [copy y N times or delete y], the new string is NOT is L! What happened to your 100% Warranty??!”`

Pumping Lemma’s Script
“`Is L Regular? In this shop I only work on Regular Languages.”`

“`Thanks. For the language L that you’ve given me, I pick this nice pumping length I call p.”`

“`Great string, thanks. I’ve cut s up into parts xyz for you. I won’t tell you what they are exactly, but I will say this: |y| > 0 and |xy| ≤ p. Also, I make you this 100% Lifetime Warranty: you can remove y, or copy it as many times as you like, and the new string will still be in L, I promise!”`

“`Well, then L wasn’t a Regular Language. Since you lied, the Warranty was void. Thanks for playing.”`
How to use the Pumping Lemma Script to write a Pumping Lemma Proof (1)

Let’s look at the parts of your script:

**Picking language L:**
Done—given to you in the homework/exam problem

**Start the proof:**
Proof by contradiction: Assume L is regular

**Picking s:**
Here you need to get creative
Try several things, remember:

- s must be in L
- |s| must be ≥ p (often do this by making some pattern repeat p times, e.g., s = “0^p1^p” is clearly of length >= p)
How to use the Pumping Lemma Script to write a Pumping Lemma Proof (2)

Let’s look at the parts of your script:

Picking $i$ (the number of times to copy part $y$):

- For many problems, two main ways to go:
  - $i = \text{big}$: (say, 3 or $p$), or
  - $i = \text{small}$: 0 (delete $y$)
- Try both and see if either “breaks” $s$ (makes a string not in $L$)
- If several tries don’t work, you may need to design a different $s$

Once you find an $i/s$ pair that “breaks” the warranty, this is a contradiction, and so the assumption is false, and $L$ is not Regular. QED.
L = \{w \mid w = a^n b^n \text{ for } n \geq 0\}

Is L regular or non-regular?

To prove L regular: Show there is a DFA (or NFA or regular expression) that recognizes/describes L. We would construct one!

To prove L non-regular: Show there is no DFA (or NFA or regular expression) that recognizes L.

Proof by contradiction: Suppose L is regular, i.e., there is a DFA that accepts L. Then the Pumping Lemma applies....
\[ L = \{ w \mid w = a^n b^n \text{ for } n \geq 0 \} \text{ is not regular.} \]

Assume \( L \) is regular. Then the pumping lemma holds for \( L \).

Let \( p \) be the pumping length for \( L \) given by the pumping lemma.

Consider the string \( s = a^p b^p \) in \( L \) of length \( > p \).

Because \( s \in L \) and is of length \( > p \), by the pumping lemma, \( s \) can be split into 3 pieces, \( s = xyz \), with:

1. \( |xy| \leq p \) and
2. \( |y| > 0 \)
3. for \( i \geq 0 \), the string \( xy^iz \in L \).
$L = \{w \mid w = a^n b^n \text{ for } n \geq 0\}$ not regular, Cont’d.

But since $s = a^p b^p = xyz$ and $|xy| \leq p$, it must be that $xy$ consists only of a’s, and since $|y| > 0$, $y$ consists of one or more a’s.

Let $i = 0$, obtaining the string $xz$. By condition 3 of the pumping lemma, $xz \in L$.

But string $xz$ has fewer a’s than $s = xyz$, but has the same number of b’s. This is because $y$ has at least 1 a, and $y$ does not occur in $xz$, but $xz$ has the same number of b’s as $xyz$. Therefore $xz$ is not in $L$, contradicting our assumption that $L$ was regular.

We conclude that $L$ is not regular.
Non-Regular Languages

• Pumping Lemma gives a method for moving outside the boundary of Regular languages

• Closure properties of Regular Languages keep us *in* the boundaries of the class of Regular Language.
  • Union
  • Intersection
  • Concatenation
  • Star
  • Complement
  • Reversing