Regular Expressions

11 March 2021
Fun:

Assignment 2 corrections due today!
Assignment 3 out today!
Where are we?
A language $L$ is a \textit{regular language} if there is a DFA $D$ such that $L(D) = L$. 
THEOREM. The following are equivalent:

- $L$ is a regular language.
- There is a DFA for $L$.
- There is an NFA for $L$. 
If $w \in \Sigma^*$ and $x \in \Sigma^*$, then $wx$ is the \textit{concatenation} of $w$ and $x$.

If $L_1$ and $L_2$ are languages over $\Sigma$, the \textit{concatenation} of $L_1$ and $L_2$ is the language $L_1L_2$, defined as

$$L_1L_2 = \{wx \mid w \in L_1 \text{ and } x \in L_2\}$$

For example, if $L_1 = \{a, ba, bb\}$ and $L_2 = \{aa, bb\}$, then

$$L_1L_2 = \{aaa, abb, baaa, babb, bbaa, bbbb\}$$
Lots of concatenation

Consider the language $L = \{ \text{aa}, \text{b} \}$

$L L$ is the set of strings formed by concatenating pairs of strings in $L$:

{aaaa, aab, baa, bb}

$LLL$ is the set of strings formed by concatenating triples of strings in $L$:

{aaaaaa, aaaaab, aabaa, aabb, baaaa, baab, bbaa, bbb}

$LLLL$ is the set of strings formed by concatenating quadruples of strings in $L$…
Language exponentiation

We can define what it means to “exponentiate” a language as follows:

\[ L^0 = \{ \varepsilon \} \]

**Base case:** Any string formed by concatenating zero strings together is just the empty string.

\[ L^{n+1} = L \cdot L^n \]

**Recursive case:** Concatenating \( n + 1 \) strings together works by concatenating \( n \) strings, then concatenating one more.
Kleene (star) closure

An important operation on languages is the Kleene closure, which is defined as

$$L^* = \{ w \in \Sigma^* \mid \exists n \in \mathbb{N}_0 . w \in L^n \}$$

A word is in $L^*$ iff it’s in one of the languages $L^0$, $L^1$, $L^2$, ...

That is, $L^*$ consists of all the possible ways of concatenating zero or more strings in $L$. 
If \( L = \{a, bb\} \), then \( L^* = \{
\begin{align*}
\varepsilon, \\
a, bb, \\
aa, abb, bba, bbbb, \\
aaa, aabb, abba, abbbb, bbba, bbabb, bbbba, \\
bbbbbb, \\
\ldots
\end{align*}
\}
Last class, we saw that the class of regular languages is *closed* under the following operations:

- Complement
- Union
- Intersection
- Concatenation
- Kleene star
Regular expressions
We’ve seen we can show a language is regular by
constructing a DFA for it
constructing an NFA for it (with or without $\varepsilon$-transitions)

We can also show a language is regular by constructing it out of other regular languages using closure properties.
Regular expressions are a concise notation for describing how to assemble a larger language out of smaller pieces.
This is a bottom-up approach to the regular languages:

- Start with a small set of simple languages we know to be regular
- Use closure properties to combine these to form more elaborate languages

Regular expressions provide a string representation for describing a language in this way.
Atomic regular expressions

Regular expressions start with three simple building blocks:

For any symbol $\alpha \in \Sigma$, the regular expression $\alpha$ represents the language $\{\alpha\}$

The symbol $\varepsilon$ is a regular expression representing the language $\{\varepsilon\}$

The symbol $\emptyset$ is a regular expression for the empty language $\emptyset$
Compound regular expressions

New regular expressions are built out of existing ones using symbols for the regular operations:

- union,
- concatenation, and
- Kleene star.
Operation: Union

If $R_1$ and $R_2$ are regular expressions, 

$$(R_1 \cup R_2)$$

is a regular expression for the union of the languages of $R_1$ and $R_2$:

$$L((R_1 \cup R_2)) = L(R_1) \cup L(R_2).$$
Operation: Concatenation

If $R_1$ and $R_2$ are regular expressions,

$$(R_1 \circ R_2) \text{ or } (R_1 R_2)$$

is a regular expression for the concatenation of the languages of $R_1$ and $R_2$. 
Operation: Kleene star

If $R$ is a regular expression, $(R^*)$ is a regular expression for the Kleene closure of the language of $R$. 
DEFINITION. $R$ is a regular expression if $R$ is

1. $\alpha$ for some $\alpha \in \Sigma$
2. $\varepsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions
5. $(R_1 \circ R_1)$, where $R_1$ and $R_1$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression

Basis

Recursive cases

Every regular expression arises by a finite number of applications of these six rules.
Order of operations

We can omit parentheses to make regular expressions more compact, but this makes them ambiguous unless we define precedence:

1. Parentheses – \((R)\)

2. Kleene star – \(R^*\)

3. Concatenation – \(R_1 \circ R_2\) or \(R_1 R_2\)

4. Union – \(R_1 \cup R_2\)
Empty strings, empty sets

Do not confuse the regular expressions:

\( \varepsilon \) – the language containing only the empty string

\( \emptyset \) – the language containing no strings

Identities:

\[ R \cup \emptyset = R \]

\[ R \circ \varepsilon = R \]
R is a regular expression if R is
1 $\alpha$ for some $\alpha \in \Sigma$
2 $\varepsilon$
3 $\emptyset$
4 $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions
5 $(R_1 \circ R_1)$, where $R_1$ and $R_1$ are regular expressions
6 $(R_1^*)$, where $R_1$ is a regular expression

EXAMPLE. To prove $((a(b^*)) \cup a)$ is a regular expression
over $\Sigma = \{a, b\}$, show it can be constructed according to
the rules:

1 $b$ is regular by Rule 1
2 $(b^*)$ is regular by Rule 6
3 $a$ is regular by Rule 1
4 $(a(b^*))$ is regular by Rule 5
5 $((a(b^*)) \cup a)$ is regular by Rule 4 applied to expressions (4) and (3)
Examples

$L(\text{hi}) = \{\text{hi}\}$

$L(\text{hi} \cup \text{heyy}^*) = \{\text{hi}, \text{hey}, \text{heyy}, \text{heyyy}, \ldots\}$

$L(((0(0 \cup 1))^*) = \text{the set of strings of } 0\text{s and } 1\text{s, of even length, such that every odd position has a } 0$

\[0 [0 \text{ or } 1] 0 [0 \text{ or } 1] 0 [0 \text{ or } 1] \ldots\]
A few more examples...

\[
ab^*a
\]
\[
a^*b^*
\]
\[
(ab)^*
\]
Is this the same as \(a^*b^*\)?
\[
a^*b^*a^*
\]
Is \(baa\) in this?

\[
L = \{x^{\text{odd}}\}
\]
\[
= x(xx)^* \text{ or } (xx)^*x
\]
(but not \(x^{*xx^*}\))

All strings of \(a\)s and \(b\)s of exactly length 3

\[
L = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}
\]
\[
\text{or (}a \cup b\text{)} \ (a \cup b) \ (a \cup b)
\]
\[
\text{or (}a \cup b\text{)}^3
\]
The *language of a regular expression* is the language described by that regular expression. Formally,

\[
\begin{align*}
L(\varepsilon) &= \{\varepsilon\} \\
L(\emptyset) &= \emptyset \\
L(a) &= \{a\} \\
L(R_1 R_2) &= L(R_1) L(R_2) \\
L(R_1 \cup R_2) &= L(R_1) \cup L(R_2) \\
L(R^*) &= L(R)^* \\
L((R)) &= L(R)
\end{align*}
\]
Designing regular expressions
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid w \text{ contains } aa \text{ as a substring}\}$
Designing regular expressions

Let \( \Sigma = \{a, b\} \)

Let \( L = \{w \in \Sigma^* \mid w \text{ contains } aa \text{ as a substring}\} \)

\[(a \cup b)^*aa(a \cup b)^*\]
Designing regular expressions

Let \( \Sigma = \{a, b\} \)

Let \( L = \{w \in \Sigma^* \mid w \text{ contains } aa \text{ as a substring}\} \)

\[ (a \cup b)^*aa(a \cup b)^* \]
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid w \text{ contains } aa \text{ as a substring}\}$

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

$bbabbbbaabab$

$aaaa$

$bbbbbbbbbaabbbb$
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid w \text{ contains } aa \text{ as a substring}\}$

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

$bbabbbbaabab$

$aaaa$

$bbbbbabbbbbaabbbbb$
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid w \text{ contains } aa \text{ as a substring}\}$

$$\Sigma^*aa\Sigma^*$$

A convenient shorthand

bbabbbbaabab

aaaa

bbbbbabbbbaabbbbb
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid |w| = 4}\}$
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid |w| = 4\}$

*Recall: $|w|$ denotes the length of string $w$.*/
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid |w| = 4\}$
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid |w| = 4\}$
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid |w| = 4\}$
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid |w| = 4\}$
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid |w| = 4\}$

$\Sigma^4$

aaaa
baba
bbbb
baaa

Another shorthand
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid w \text{ contains at most one } a\}$

Here are some candidate regular expressions for $L$. Which are correct?

- $\Sigma^* a \Sigma^*$
- $b^* a b^* u b^*$
- $b^* (a u \varepsilon) b^*$
- $b^* a^* b u b^*$
- $b^* (a^* u \varepsilon) b^*$
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid w \text{ contains at most one } a\}$

$b^* (a \cup \varepsilon) b^*$
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid w \text{ contains at most one } a\}$

$b^* (a \cup \epsilon) b^*$
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid w$ contains at most one $a\}$

\[b^*(au\varepsilon)b^*\]

$$bbbabb$$
$$bbbbbb$$
$$abbb$$
$$a$$
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid w \text{ contains at most one } a\}$

\[ b^* (a \cup \varepsilon) b^* \]

- bbbbbabbb
- bbbbbb
- abbb
- a
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid w \text{ contains at most one } a\}$

\[
\begin{align*}
\text{b*}(a?b*) \\
\text{bbbbabbb} \\
\text{bbbbbb} \\
\text{abbb} \\
\text{a}
\end{align*}
\]
Designing regular expressions

Let $\Sigma = \{a, b\}$

Let $L = \{w \in \Sigma^* \mid w \text{ contains at most one } a\}$

Another shorthand
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

Let's make a regular expression for email addresses.

$$aa^*$$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents "any letter".

Let’s make a regular expression for email addresses.

$aa^*$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”. Let’s make a regular expression for email addresses.

$$aa^*(.aa^*)^*$$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

$$aa^*(.aa^*)^*$$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

$$aa*(.aa*)*@$$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

$$aa*(.aa*)*@$$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

$$aa^*(.aa^*)*@a^*.aa^*$$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

$$aa^*(.aa^*)*@aa^*.aa^*$$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

$$aa*(.aa*)*@aa*.aa*(.aa*)*$$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

\[
aa^*(.aa^*)@aa^*.aa^*(.aa^*)^*
\]

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

$$a^+(.aa*)*@aa*.aa*(.aa*)*$$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

You guessed it – another shorthand

```
a+(.aa*)*@aa*.aa*(.aa*)*
```

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

$$a^+ (.a^+)*@a^+.a^+ (.a^+)*$$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
A more elaborate design

Let $\Sigma = \{a, ., @\}$, where $a$ represents “any letter”.

Let’s make a regular expression for email addresses.

$$a^+(a^+*)@a^+(a^+*)^+$$

mvassar@vassar.edu
matthew.vassar@vassarbrewery.com
matt@cs.vassar.edu
For comparison

\[ a^+ (a^+) @ a^+ (a^+)^+ \]
Shorthands summary

$\Sigma$ is a shorthand for “any character in $\Sigma$”

$R^n$ is a shorthand for $RR\ldots R$ ($n$ times)

$R?$ is shorthand for $(Ru\varepsilon)$ – that is, zero or one copies of $R$.

$R^+$ is a shorthand for $RR^*$ – that is, one or more copies of $R$. 
How to design regular expressions

Write out some sample strings in the language and look for patterns:

Can I separate out the strings into two (or more) categories?
   Find the pattern for each category, then union together.

Can I break this problem down into solving smaller subproblems?
   Find the pattern for each piece/subproblem then concatenate together.

Is there some sort of repeating structure?
   Find the smallest repeating unit, then use Kleene star on that pattern.
BANANA GRAMMAR

BANANA  BABA  ANANNANA  NANABA  ABBA

B  AN  NAB  ANNA BANANNA  BBBBBBAN

AAAAAAA  NBA  BANANA?  NANANANANANANANA  BANBAAA
Banana languages

Let $\Sigma = \{a, n\}$.

Design a regular expression for the language

$L = \{w \in \Sigma^+ \mid \text{the characters of } w \text{ alternate between } a \text{ and } n\}$
Banana languages

Let $\Sigma = \{a, n\}$.

Design a regular expression for the language

$L = \{w \in \Sigma^+ \mid \text{the characters of } w \text{ alternate between } a \text{ and } n\}$

$n \in L$

$anana \in L$

$nanananan \in L$
Banana languages

Let $\Sigma = \{a, n\}$.

Design a regular expression for the language $L = \{w \in \Sigma^+ |$ the characters of $w$ alternate between $a$ and $n\}$

- $n \in L$
- $aan \not\in L$
- $anana \in L$
- $nnnnn \not\in L$
- $nanananan \in L$
- $anaana \not\in L$
\[ L = \{ w \in \Sigma^+ \mid \text{the characters of } w \text{ alternate between } a \text{ and } n \} \]

Write out some sample strings in the language and look for patterns:

- Can I separate out the strings into two (or more) categories?
  - Find the pattern for each category, then union together.
- Can I break this problem down into solving smaller subproblems?
  - Find the pattern for each piece/subproblem then concatenate together.
- Is there some sort of repeating structure?
  - Find the smallest repeating unit, then use Kleene star on that pattern.
$$L = \{ w \in \Sigma^+ \mid \text{the characters of } w \text{ alternate between } a \text{ and } n \}$$

Write out some sample strings in the language and look for patterns:

- Can I separate out the strings into two (or more) categories?
  - Find the pattern for each category, then union together.
- Can I break this problem down into solving smaller subproblems?
  - Find the pattern for each piece/subproblem then concatenate together.
- Is there some sort of repeating structure?
  - Find the smallest repeating unit, then use Kleene star on that pattern.
$L = \{w \in \Sigma^+ | \text{the characters of } w \text{ alternate between } a \text{ and } n\}$

Write out some sample strings in the language and look for patterns:

- Can I separate out the strings into two (or more) categories?
  - Find the pattern for each category, then union together.
- Can I break this problem down into solving smaller subproblems?
  - Find the pattern for each piece/subproblem then concatenate together.
- Is there some sort of repeating structure?
  - Find the smallest repeating unit, then use Kleene star on that pattern.
\[ L = \{ w \in \Sigma^+ \mid \text{the characters of } w \text{ alternate between } a \text{ and } n \} \]

<table>
<thead>
<tr>
<th>Starts with ( a )</th>
<th>Starts with ( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( n )</td>
</tr>
<tr>
<td>( an )</td>
<td>( na )</td>
</tr>
<tr>
<td>( ana )</td>
<td>( nan )</td>
</tr>
<tr>
<td>( anan )</td>
<td>( nana )</td>
</tr>
<tr>
<td>( anana )</td>
<td>( nanan )</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Write out some sample strings in the language and look for patterns:

- Can I separate out the strings into two (or more) categories?
  - Find the pattern for each category, then **union** together.
- Can I break this problem down into solving smaller subproblems?
  - Find the pattern for each piece/subproblem then **concatenate** together.
- Is there some sort of repeating structure?
  - Find the smallest repeating unit, then use **Kleene star** on that pattern.
$L = \{w \in \Sigma^+ \mid \text{the characters of } w \text{ alternate between } a \text{ and } n\}$

Write out some sample strings in the language and look for patterns:

<table>
<thead>
<tr>
<th>Starts with $a$</th>
<th>Starts with $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$n$</td>
</tr>
<tr>
<td>$an$</td>
<td>$na$</td>
</tr>
<tr>
<td>$ana$</td>
<td>$nan$</td>
</tr>
<tr>
<td>$anan$</td>
<td>$nana$</td>
</tr>
<tr>
<td>$anana$</td>
<td>$nanan$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Can I separate out the strings into two (or more) categories?

Find the pattern for each category, then union together.

Can I break this problem down into solving smaller subproblems?

Find the pattern for each piece/subproblem then concatenate together.

Is there some sort of repeating structure?

Find the smallest repeating unit, then use Kleene star on that pattern.
\[ L = \{ w \in \Sigma^+ \mid \text{the characters of } w \text{ alternate between } a \text{ and } n \} \]

<table>
<thead>
<tr>
<th>Starts with a</th>
<th>Starts with n</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>n</td>
</tr>
<tr>
<td>an</td>
<td>na</td>
</tr>
<tr>
<td>ana</td>
<td>nan</td>
</tr>
<tr>
<td>anan</td>
<td>nana</td>
</tr>
<tr>
<td>anana</td>
<td>nanan</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Write out some sample strings in the language and look for patterns:

Can I separate out the strings into two (or more) categories?
- Find the pattern for each category, then \textit{union} together.

Can I break this problem down into solving smaller subproblems?
- Find the pattern for each piece/subproblem then \textit{concatenate} together.

Is there some sort of repeating structure?
- Find the smallest repeating unit, then use \textit{Kleene star} on that pattern.
$L = \{ w \in \Sigma^+ \mid \text{the characters of } w \text{ alternate between } a \text{ and } n \}$

Write out some sample strings in the language and look for patterns:

Can I separate out the strings into two (or more) categories?

Find the pattern for each category, then union together.

Can I break this problem down into solving smaller subproblems?

Find the pattern for each piece/subproblem then concatenate together.

Is there some sort of repeating structure?

Find the smallest repeating unit, then use Kleene star on that pattern.

*a*(sequence of *nas*) *(possibly another *n*)
\[ L = \{ w \in \Sigma^+ \mid \text{the characters of } w \text{ alternate between } a \text{ and } n \} \]

Write out some sample strings in the language and look for patterns:

<table>
<thead>
<tr>
<th>Starts with a</th>
<th>Starts with n</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>n</td>
</tr>
<tr>
<td>an</td>
<td>na</td>
</tr>
<tr>
<td>ana</td>
<td>nan</td>
</tr>
<tr>
<td>anan</td>
<td>nana</td>
</tr>
<tr>
<td>anana</td>
<td>nanan</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Can I separate out the strings into two (or more) categories?

Find the pattern for each category, then **union** together.

Can I break this problem down into solving smaller subproblems?

Find the pattern for each piece/subproblem then **concatenate** together.

Is there some sort of repeating structure?

Find the smallest repeating unit, then use **Kleene star** on that pattern.

\[ a(\text{sequence of } n\!a\!s)(\text{possibly another } n) \]
\[ L = \{ w \in \Sigma^+ \mid \text{the characters of } w \text{ alternate between } a \text{ and } n \} \]

Write out some sample strings in the language and look for patterns:

Can I separate out the strings into two (or more) categories?

Find the pattern for each category, then \textit{union} together.

Can I break this problem down into solving smaller subproblems?

Find the pattern for each piece/subproblem then \textit{concatenate} together.

Is there some sort of repeating structure?

Find the smallest repeating unit, then use \textit{Kleene star} on that pattern.

\[ a(na)^*n? \]
\[ L = \{ w \in \Sigma^+ \mid \text{the characters of } w \text{ alternate between } a \text{ and } n \} \]

<table>
<thead>
<tr>
<th>Starts with ( a )</th>
<th>Starts with ( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( n )</td>
</tr>
<tr>
<td>( an )</td>
<td>( na )</td>
</tr>
<tr>
<td>( ana )</td>
<td>( nan )</td>
</tr>
<tr>
<td>( anan )</td>
<td>( nana )</td>
</tr>
<tr>
<td>( anana )</td>
<td>( nanan )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Write out some sample strings in the language and look for patterns:

- Can I separate out the strings into two (or more) categories?
  - Find the pattern for each category, then \textit{union} together.
- Can I break this problem down into solving smaller subproblems?
  - Find the pattern for each piece/subproblem then \textit{concatenate} together.
- Is there some sort of repeating structure?
  - Find the smallest repeating unit, then use \textit{Kleene star} on that pattern.

\[ a(na)^*n? \cup n(an)^*a? \]
Regular expressions in the real world
UNIX regular expressions

From the beginning (of time), UNIX has used regular expressions in many places, including the `grep` command.

`grep` = global (search for a) regular expression and print

Many UNIX commands use an extended RE notation, but it still expresses only the regular languages.
UNIX RE notation

\([a_1 a_2 \ldots a_n]\) is shorthand for \(a_1 \cup a_2 \cup \cdots \cup a_n\).

Ranges are indicated by first-dash-last and brackets, using ASCII character order, e.g.,

\([a\texttt{--}z]\) = any lowercase letter

\([a\texttt{--}zA\texttt{--}Z]\) = any letter

Dot (\(\cdot\)) = any character (like our shorthand \(\Sigma\))
UNIX RE notation, *continued*

Since characters like brackets, dashes, and dots have special meaning, if you want to match them, you need to quote with backslash (`\`).

Union operator is represented with a bar (`|`)

Includes our `+` shorthand for “one or more”, e.g.,

```
[a-z]+ = one or more lowercase letter
```
Perl, Python, Emacs, …

Include additional extensions, notably character classes like \b for word boundary characters, \w for word characters, etc.

With each implementation of regular expressions, they become less standard, so what you write for one language or application won’t work in another.
| **grep** |
|------------------|------------------|
| grep lets you search files for text |
| `$ grep bananas foo.txt` |
| Here are some of my favourite grep command line arguments! |
| `-E` aka `egrep` |
| Use if you want regexps like `.*+` to work. Otherwise you need to use `".*+"` |
| `-v` |
| invert match: find all lines that don't match |
| `-i` case insensitive |
| Show context for your search. |
| `$ grep -A 3 foo` will show 3 lines of context after a match |
| `-a` |
| only show the filenames of the files that matched |
| `-o` |
| only print the matching part of the line (not the whole line) |
| `-A` |
| grep alternatives |
| `-F` don't treat the matching string as a regex /
| eg `$ grep -F ...` |
| `-r` Recursive! Search all the files in a directory. |
| `-c` search binaries: treat binary data like it's text instead of ignoring it! |

https://twitter.com/b0rk/status/991880504805871616
Lexical analysis

The first thing a compiler does is break a program into *tokens*, which are substrings that together represent a unit, e.g.,

- identifiers
- reserved words like “if”
- meaningful single characters like “;” or “+”
- multi-character operators like “<=”
Lexical analysis, continued

There are tools like **lex** or **flex** that let you write a regular expression for each kind of token.

E.g., in UNIX notation, identifiers are something like 

```
[A-Za-z][A-Za-z0-9_]*
```

Each RE has an associated action like returning a code for the token found or adding it to a symbol table.
Acknowledgments

This lecture incorporates material from:

Nancy Ide, Vassar College
Keith Schwarz, Stanford University
Michael Sipser, *Introduction to the Theory of Computation*
Jeffrey Ullman, Stanford University