

Text Search and Closure Properties

CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN

Fall 2019

Chinese University of Hong Kong

Text Search

```
grep -E regex file.txt
```

Searches for an occurrence of patterns matching a regular expression

regex	language	meaning
<code>cat 12</code>	<code>{cat, 12}</code>	union
<code>[abc]</code>	<code>{a, b, c}</code>	shorthand for <code>a b c</code>
<code>[ab][12]</code>	<code>{a1, a2, b1, b2}</code>	concatenation
<code>(ab)*</code>	<code>{ϵ, ab, abab, ...}</code>	star
<code>[ab]?</code>	<code>{ϵ, a, b}</code>	zero or one
<code>(cat)+</code>	<code>{cat, catcat, ...}</code>	one or more
<code>[ab]{2}</code>	<code>{aa, ab, ba, bb}</code>	<i>n</i> copies

Searching with grep

Words containing
savor or savour

```
cd /usr/share/dict/  
grep -E 'savou?r' words
```

```
savor  
savor's  
savored  
savorier  
savories  
savoriest  
savoring  
savors  
savory  
savory's  
unsavory
```

Searching with grep

Words containing
savor or savour

```
cd /usr/share/dict/  
grep -E 'savou?r' words
```

```
savor  
savor's  
savored  
savorier  
savories  
savoriest  
savoring  
savors  
savory  
savory's  
unsavory
```

Words with 5 consecutive a or b
grep -E '[abAB]{5}' words

```
Babbage
```

More grep commands

.	any symbol
[a-d]	anything in a range
^	beginning of line
\$	end of line

```
grep -E '^a.pl.$' words
```



How do you look for

Words that start in **go** and have another **go**

```
grep -E '^go.*go' words
```

Words with **at least** ten vowels?

```
grep -iE '([aeiou].*){10}' words
```

Words **without** any vowels?

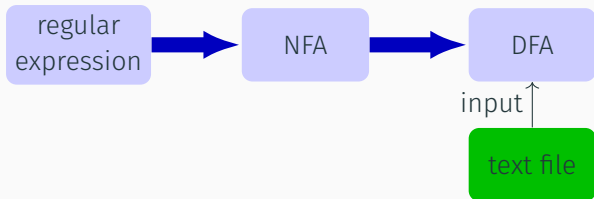
```
grep -iE '^[^aeiou]*$' words
```

[^R] means “does not contain”

Words with **exactly** ten vowels?

```
grep -iE '^[^aeiou]*([aeiou][^aeiou]*){10}$' words
```

How grep (could) work



differences	in class	in grep
<code>[ab]?, a+, (cat){3}</code>	not allowed	allowed
input handling	matches whole	looks for substring
output	accept/reject	finds substring

Regular expression also supported in modern languages (C, Java, Python, etc)

Implementation of grep

How do you handle expressions like

<code>[ab]?</code>	$\rightarrow () [ab]$	zero or more	$R? \rightarrow \epsilon R$
<code>(cat)+</code>	$\rightarrow (cat)(cat)^*$	one or more	$R+ \rightarrow RR^*$
<code>a{3}</code>	$\rightarrow aaa$	n copies	$R\{n\} \rightarrow \underbrace{RR \dots R}_{n \text{ times}}$
<code>[^aeiouy]</code>	$?$	not containing	

Closure properties

Example

The language L of strings that end in 101 is regular

$$(0 + 1)^*101$$

How about the language \bar{L} of strings that do not end in 101?

Example

The language L of strings that end in 101 is regular

$$(0 + 1)^*101$$

How about the language \bar{L} of strings that do not end in 101?

Hint: a string does not end in 101 if and only if it ends in

000, 001, 010, 011, 100, 110 or 111

or has length 0, 1, or 2

So \bar{L} can be described by the regular expression

$$(0 + 1)^*(000 + 001 + 010 + 011 + 100 + 110 + 111) + \varepsilon + (0 + 1) + (0 + 1)(0 + 1)$$

Complement

The **complement** \bar{L} of a language L contains those strings that are not in L

$$\bar{L} = \{w \in \Sigma^* \mid w \notin L\}$$

Examples $(\Sigma = \{0, 1\})$

L_1 = lang. of all strings that end in 101

\bar{L}_1 = lang. of all strings that **do not end** in 101

= lang. of all strings that end in 000, ..., 111 (but not 101)
or have length 0, 1, or 2

L_2 = lang. of $1^* = \{\epsilon, 1, 11, 111, \dots\}$

\bar{L}_2 = lang. of all strings that contain at least one 0

= lang. of the regular expression $(0 + 1)^*0(0 + 1)^*$

Example

The language L of strings that contain 101 is regular

$$(0 + 1)^*101(0 + 1)^*$$

How about the language \bar{L} of strings that **do not contain** 101?

You can write a regular expression, but it is a lot of work!

Closure under complement

If L is a regular language, so is \bar{L}

To argue this, we can use any of the **equivalent definitions** of regular languages

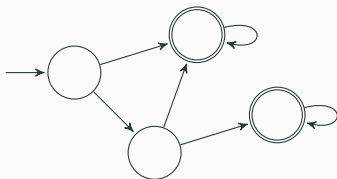


The **DFA definition** will be the most convenient here

We assume L has a DFA, and show \bar{L} also has a DFA

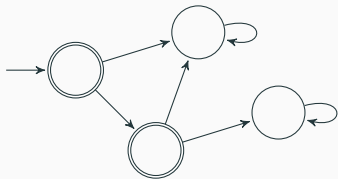
Arguing closure under complement

Suppose L is regular, then it has a DFA M



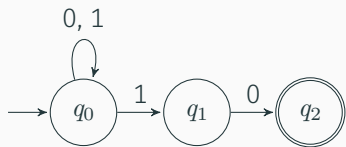
accepts L

Now consider the DFA M' with the accepting and rejecting states of M reversed

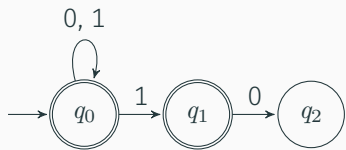


accepts strings not in L

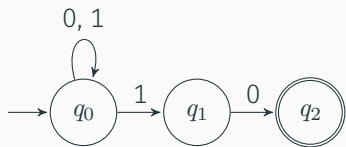
Can we do the same with an NFA?



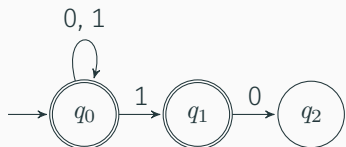
$(0 + 1)^*10$



Can we do the same with an NFA?



$(0 + 1)^*10$



$(0 + 1)^*$

Not the complement!

Intersection

The **intersection** $L \cap L'$ is the set of strings that are in both L and L'

Examples:

L	L'	$L \cap L'$
$(0 + 1)^*11$	1^*	1^*11

L	L'	$L \cap L'$
$(0 + 1)^*10$	1^*	\emptyset

If L and L' are regular, is $L \cap L'$ also regular?

Closure under intersection

If L and L' are regular languages, so is $L \cap L'$

To argue this, we can use any of the [equivalent definitions](#) of regular languages

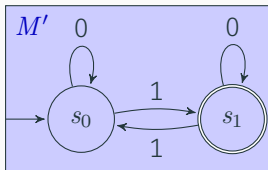


Suppose L and L' have DFAs, call them M and M'

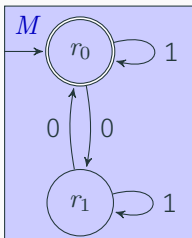
Goal: construct a DFA (or NFA) for $L \cap L'$

Example

L' (odd number of 1s)



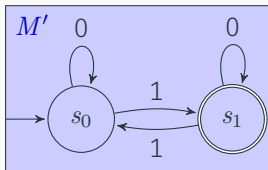
L (even number of 0s)



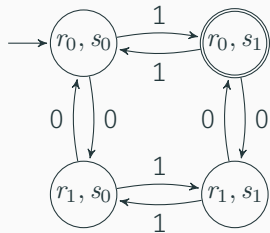
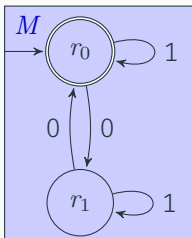
$L \cap L' = \text{lang. of even number of 0s and odd number of 1s}$

Example

L' (odd number of 1s)



L (even number of 0s)



$L \cap L' = \text{lang. of even number of 0s and odd number of 1s}$

Closure under intersection

	M and M'	DFA for $L \cap L'$
states	$Q = \{r_1, \dots, r_n\}$ $Q' = \{s_1, \dots, s_m\}$	$Q \times Q' = \{(r_1, s_1), (r_1, s_2), \dots, (r_2, s_1), \dots, (r_n, s_m)\}$
start states	r_i for M s_j for M'	(r_i, s_j)
accepting states	F for M F' for M'	$F \times F' = \{(r_i, s_j) \mid r_i \in F, s_j \in F'\}$

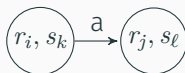
Whenever M is in state r_i and M' is in state s_j , the DFA for $L \cap L'$ will be in state (r_i, s_j)

Closure under intersection

M and M'

DFA for $L \cap L'$

transitions



The reversal w^R of a string w is w written backwards

$$w = \text{dog} \quad w^R = \text{god}$$

The reversal L^R of a **language** L is the language obtained by reversing all its strings

$$L = \{\text{dog}, \text{war}, \text{level}\} \quad L^R = \{\text{god}, \text{raw}, \text{level}\}$$

Reversal of regular languages

L = language of all strings that end in 01

L is regular and has regex

$$(0 + 1)^*01$$

How about L^R ?

This is the language of all strings **beginning** in 10

It is regular and represented by

$$10(0 + 1)^*$$

Closure under reversal

If L is a regular language, so is L^R

How do we argue?



Arguing closure under reversal

Take any regular language L

Will show that L^R is union/concatenation/star of “atomic” regular languages

A regular language can be of the following types:

- \emptyset and $\{\varepsilon\}$
- alphabet symbols e.g. $\{0\}$, $\{1\}$
- **union**, **concatenation**, or **star** of simpler regular languages

Inductive proof of closure under reversal

Regular language L	reversal L^R
\emptyset	\emptyset
$\{\varepsilon\}$	$\{\varepsilon\}$
$\{x\} \quad (x \in \Sigma)$	$\{x\}$
$L_1 \cup L_2$	$L_1^R \cup L_2^R$
$L_1 L_2$	$L_2^R L_1^R$
L_1^*	$(L_1^R)^*$

Duplication?

$$L^{\text{DUP}} = \{ww \mid w \in L\}$$

Example:

$$L = \{\text{cat}, \text{dog}\}$$

$$L^{\text{DUP}} = \{\text{catcat}, \text{dogdog}\}$$

If L is regular, is L^{DUP} also regular?

Let's try regular expression

$$L^{\text{DUP}} \stackrel{?}{=} L^2$$

Attempts

Let's try regular expression

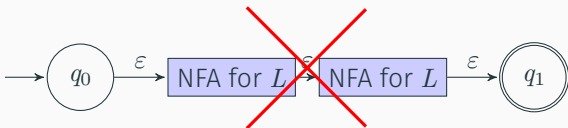
$$\cancel{L^{\text{DUP}} = L^2}$$

$$L = \{a, b\}$$

$$L^{\text{DUP}} = \{aa, bb\}$$

$$LL = \{aa, ab, ba, bb\}$$

Let's try NFA



An example

$L =$ language of 0^*1 (L is regular)

$L = \{1, 01, 001, 0001, \dots\}$

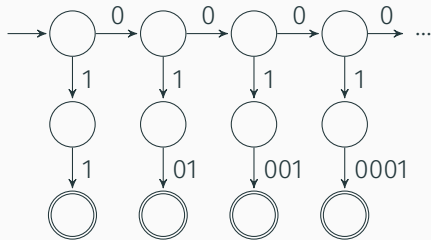
$L^{\text{DUP}} = \{11, 0101, 001001, 00010001, \dots\}$

$= \{0^n 1 0^n 1 \mid n \geq 0\}$

Let's design an NFA for L^{DUP}

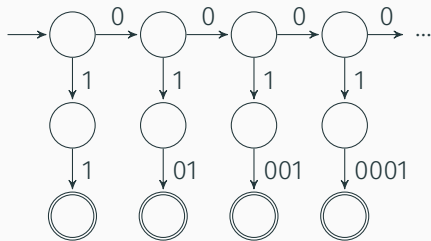
An example

$$\begin{aligned}L^{\text{DUP}} &= \{11, 0101, 001001, 00010001, \dots\} \\ &= \{0^n 1 0^n 1 \mid n \geq 0\}\end{aligned}$$



An example

$$\begin{aligned}L^{\text{DUP}} &= \{11, 0101, 001001, 00010001, \dots\} \\ &= \{0^n 10^{n+1} \mid n \geq 0\}\end{aligned}$$



Seems to require infinitely many states!

Next lecture: will show that languages like L^{DUP} are not regular

Backreferences in grep

Advanced feature in grep and other “regular expression” libraries

```
grep -E '^(.*)\1$' words
```

the special expression `\1` refers to the substring specified by `(.*)`

`(.*)\1` looks for a repeated substring, e.g. `mama`

`^(.*)\1$` accepts the language L^{DUP}

Standard “regular expression” libraries can accept irregular languages (as defined in this course)!