Text Search and Closure Properties

CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN

Chinese University of Hong Kong

Fall 2015

Text Search

grep program

`grep -E` *`regexp`* `file.txt`

Searches for an occurrence of patterns matching a regular expression

`cat|12`	{cat, 12}	union
`[abc]`	{a, b, c}	shorthand for a|b|c
`[ab][12]`	{a1, a2, b1, b2}	concatenation
`(ab)*`	{ε, ab, abab, …}	star
`[ab]?`	{ε, a, b}	zero or one
`(cat)+`	{cat, catcat, …}	one or more
`[ab]{2}`	{aa, ab, ba, bb}	n 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 '([aeiouy].*){10}' words
```

Words without any vowels?

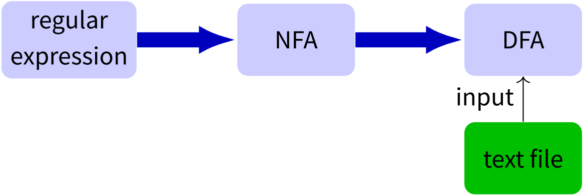
```
grep -iE '^[^aeiouy]*$' words
```

`[^R]` means "does not contain"

Words with exactly ten vowels?

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

How grep (could) work



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

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

Implementation of grep

How do you handle expressions like

`[ab]?`	$\rightarrow$ `()|[ab]`	zero or more	$R? \rightarrow \varepsilon \vert R$
`(cat)+`	$\rightarrow$ `(cat)(cat)*`	one or more	$R+ \rightarrow RR^*$
`a{3}`	$\rightarrow$ `aaa`	n copies	$R\{n\} \rightarrow \underbrace{RR \ldots R}_{n \text{ times}}$
`[^aeiouy]`	?	not containing	

Closure properties

Example

The language L of strings that end in 101 is regular

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

How about the language $\overline{L}$ 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 $\overline{L}$ 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 $\overline{L}$ 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 $\overline{L}$ of a language L contains those strings that are not in L

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

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

$$L_1 = \text{all strings that end in 101}$$

$$\overline{L_1} = \text{all strings that do not end in 101}$$

$$= \text{all strings that end in 000, ..., 111 (but not 101)}$$

$$\text{or have length 0, 1, or 2}$$

$$L_2 = 1^* = \{\varepsilon, 1, 11, 111, \dots\}$$

$$\overline{L_2} = \text{all strings that contain at least one 0}$$

$$= \text{language of the regular expression } (0+1)^*0(0+1)^*$$

Example

The language L of strings that contain 101 is regular

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

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

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

Closure under complement

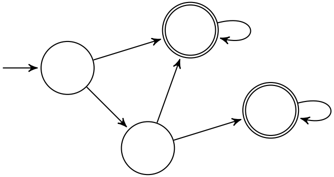
If L is a regular language, so is $\overline{L}$

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 $\overline{L}$ also has a DFA

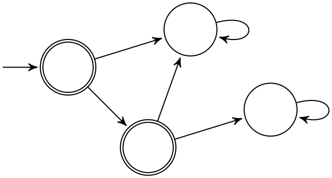
Arguing closure under complement

Suppose L is regular, then it has a DFA M



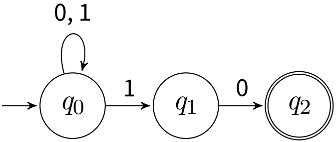
accepts L

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

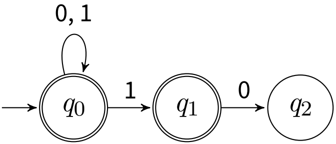


accepts strings not in L

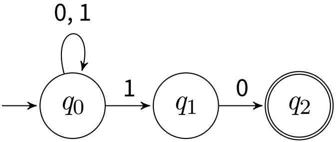
Can we do the same with an NFA?



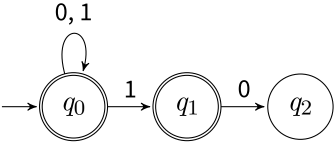
$(0 + 1)^*10$



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 $L \cap L'$

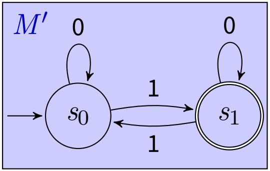
To argue this, we can use any of the equivalent definitions of regular languages

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

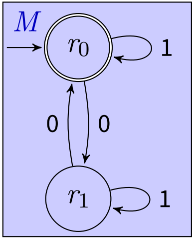
Goal: construct a DFA (or NFA) for $L \cap L'$

Example

$L' =$ odd number of 1s



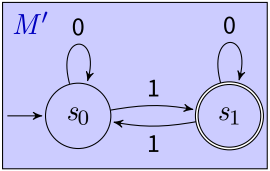
$L =$ even number of 0s



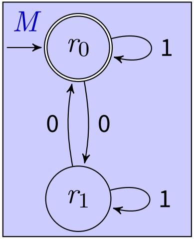
$L \cap L' =$ even number of 0s and odd number of 1s

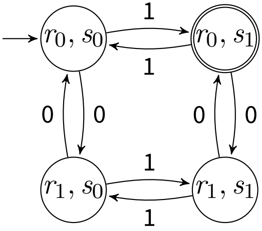
Example

$L' =$ odd number of 1s



$L =$ even number of 0s





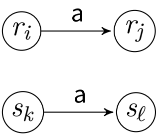
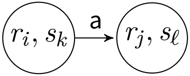
$L \cap L' =$ even number of 0s and odd number of 1s

Closure under intersection

	M and M'	DFA for $L \cap L'$
states	$Q = \{r_1, \ldots, r_s\}$ $Q' = \{s_1, \ldots, s_m\}$	$Q \times Q' = \{(r_1, s_1), (r_1, s_2),$ $\ldots, (r_2, s_1), \ldots, (r_n, s_m)\}$
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		

Reversal

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

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

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

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

Reversal of regular languages

$L =$ all strings that end in 001 is regular

$(0 + 1)^*001$

How about L^R?

This is the language of all strings beginning in 100

It is regular and represented by

$100(0 + 1)^*$

Closure under reversal

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

How do we argue?

Arguing closure under reversal

Take a regular expression E for L

We will show how to reverse E

A regular expression can be of the following types:

- special symbol $\emptyset$ and ε
- alphabet symbols like a and b
- union, concatenation, or star of simpler expressions

Proof of closure under reversal

Regular expression E	reversal E^R
$\emptyset$	$\emptyset$
ε	ε
a	a
$E_1 + E_2$	$E_1^R + E_2^R$
$E_1 E_2$	$E_2^R E_1^R$
E_1^*	$(E_1^R)^*$

Duplication?

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

Example:
$L = \{\text{cat}, \text{dog}\}$
$L^{\mathsf{DUP}} = \{\text{catcat}, \text{dogdog}\}$

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

Attempts

Let's try regular expression

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

Attempts

Let's try regular expression

~~$L^{\mathsf{DUP}} \stackrel{?}{=} L^2$~~

$$L = \{\mathsf{a}, \mathsf{b}\}$$
$$L^{\mathsf{DUP}} = \{\mathsf{aa}, \mathsf{bb}\}$$
$$LL = \{\mathsf{aa}, \mathsf{ab}, \mathsf{ba}, \mathsf{bb}\}$$

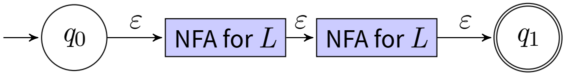
Attempts

Let's try regular expression

~~$L^{\mathsf{DUP}} \stackrel{?}{=} L^2$~~

$$L = \{\mathsf{a}, \mathsf{b}\}$$
$$L^{\mathsf{DUP}} = \{\mathsf{aa}, \mathsf{bb}\}$$
$$LL = \{\mathsf{aa}, \mathsf{ab}, \mathsf{ba}, \mathsf{bb}\}$$

Let's try NFA



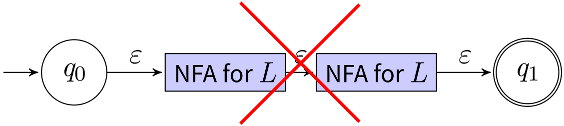
Attempts

Let's try regular expression

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

$$L = \{\mathsf{a}, \mathsf{b}\}$$
$$L^{\mathsf{DUP}} = \{\mathsf{aa}, \mathsf{bb}\}$$
$$LL = \{\mathsf{aa}, \mathsf{ab}, \mathsf{ba}, \mathsf{bb}\}$$

Let's try NFA



An example

$$L = \text{language of } 0^*1 \qquad (L \text{ is regular})$$

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

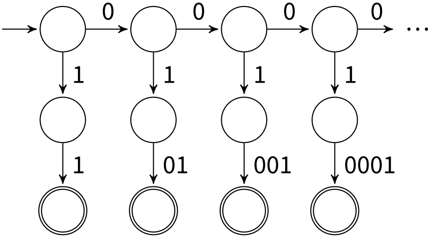
$$L^{\mathsf{DUP}} = \{11, 0101, 001001, 00010001, \ldots\}$$

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

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

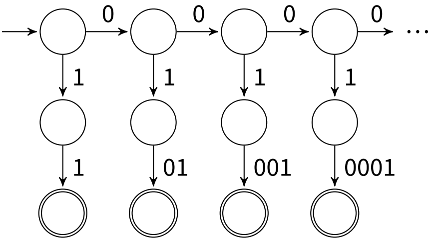
An example

$$L^{\mathsf{DUP}} = \{11, 0101, 001001, 00010001, \ldots\}$$
$$= \{0^n 1 0^n 1 \mid n \geqslant 0\}$$



An example

$$\begin{aligned} L^{\mathsf{DUP}} &= \{11, 0101, 001001, 00010001, \ldots\} \\ &= \{0^n 1 0^n 1 \mid n \geqslant 0\} \end{aligned}$$



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)!