3.

REGULAR EXPRESSIONS

Overview of today's lecture

- Review – from last lecture
- Regular expressions
- From DFAs to regular expressions
- Algebraic laws
- By the way …
- Reading material and exercises

REVIEW – FROM LAST LECTURES

Alphabet, strings and languages

- A language L is a set of strings
- A string w is a sequence of symbols from an alphabet $w = a_1\ a_2 \dots a_k$
- The empty string is written ε
- An alphabet Σ is a set of symbols (or letters)

- The concatenation of two strings w and w' is written $w\ w'$

Sets of strings

- Σ^k: the set of strings of length k
- Σ^*: The set of all strings over Σ
  - $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- Σ^+: The set of non-empty strings over Σ
  - $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots$

Operations on languages

- L^k: k copies of L concatenated
  - $L^0 = \{\varepsilon\}$
  - $L^1 = L$
  - $L^2 = L\,L = \{w\,w' \mid w \text{ and } w' \text{ are in } L\}$
  - …
- L^*: the Kleene closure of L
  - $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$
- L^+: the positive closure of L
  - $L^+ = L^1 \cup L^2 \cup \ldots$

What are

- $\emptyset^0$
- $\emptyset^k$
- $\emptyset^*$
- $\{\varepsilon\}^0$
- $\{\varepsilon\}^k$
- $\{\varepsilon\}^*$

REGULAR EXPRESSIONS

Why study regular expressions?

- DFAs (NFAs) are often used as machine for recognizing languages
  - does *w* belong to *L* where *L* is given by the DFA *A*?
- Regular expressions are used for specifying languages
- They are often the input to systems processing strings
  - When searching for strings (as e.g. UNIX grep)
  - For specifying lexical analysers in compilers

Primitive regular expressions

- ∅
  - The empty set of strings
- ε
  - The set containing only the empty string
- a
  - The set containing only the string a from the alphabet Σ

Constructed regular expressions

- E+F
  - The union of the sets of strings described by E and F
- E F
  - The concatenation of the sets of strings described by E and F
- E*
  - The Kleene closure of the set of strings described by E
- We can use parentheses as in (E)
- We can use abbreviations for regular expressions

Examples

- The set of all strings over {0,1} having an even number of 0's
- The set of all strings over {0,1} not having 111 as a substring
- All strings in {0,1} with no more than three occurrences of 0's

Examples

- (a + b c)*
  - Describes the language { ε, a, bc, aa, abc, bca, ...}
- (a+b)*(a+bb)
  - Describes the language ...
- (aa)* (bb)* b
  - Describes the language ...
- (a+b)* aa (a+b)
  - Describes the language ...

Examples from lexical analysis

- Keywords for a programming language
  - `if + then + else + while + do`
- Identifier in a programming language
  - *alpha* (*alpha* + *digit*)*

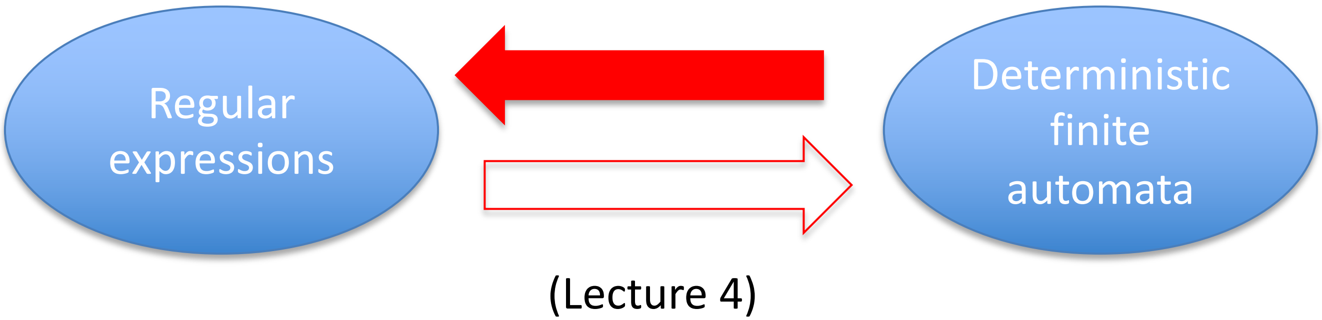
  where

  - *alpha* is an abbreviation for the alphabetic characters [A..Z a..z]
  - *digit* is an abbreviation for the digits [0..9]

The language defined by regular expressions

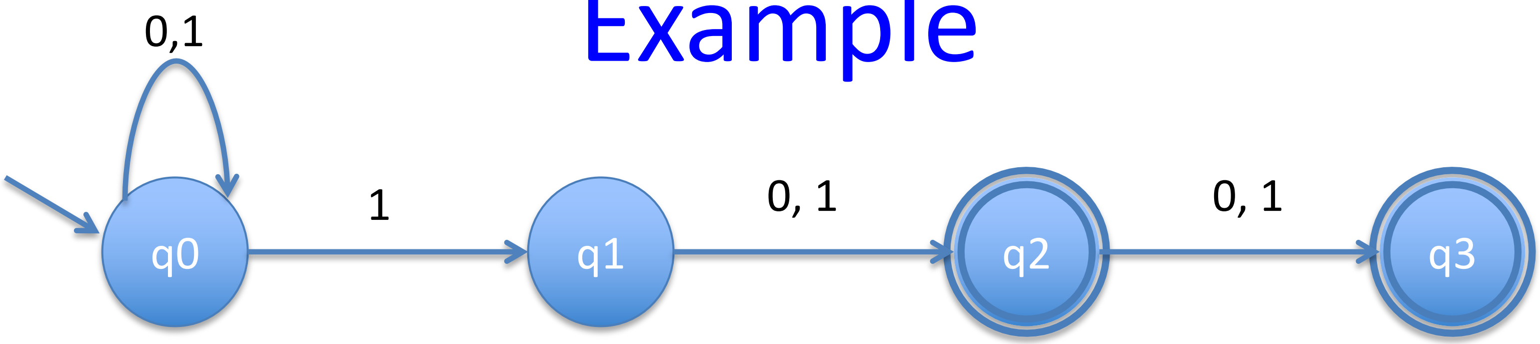
A regular expression E defines a subset of Σ^* denoted $L(E)$ and called the language of E:

- $L(\emptyset) \quad = \emptyset$
- $L(\varepsilon) \quad = \{\varepsilon\}$
- $L(a) \quad = \{a\}$
- $L(E+F) \quad = L(E) \cup L(F)$
- $L(E\ F) \quad = L(E)\ L(F)$
- $L(E^*) \quad = L(E)^*$

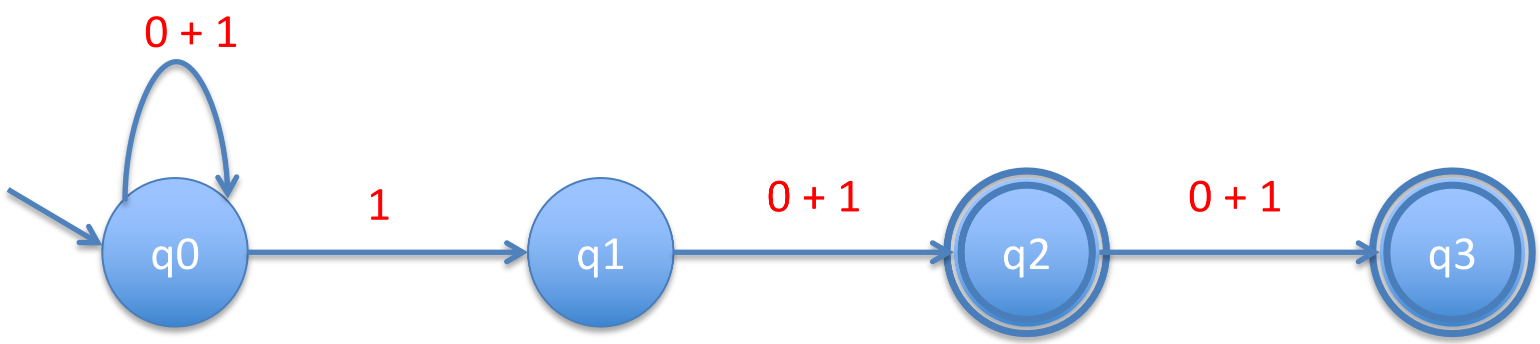


FROM FINITE AUTOMATA TO REGULAR EXPRESSIONS

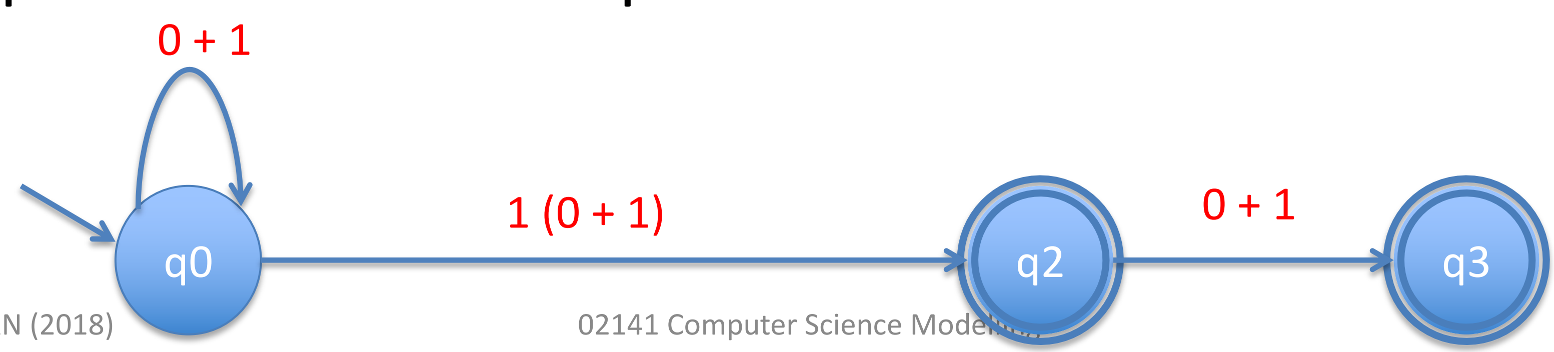
Example



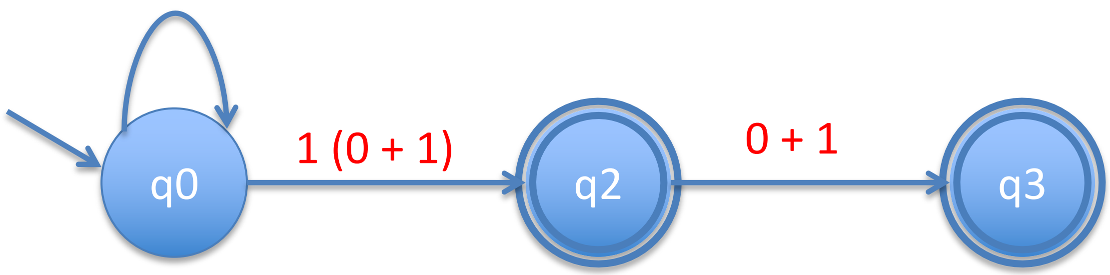
Step 1: We write regular expressions on the edges



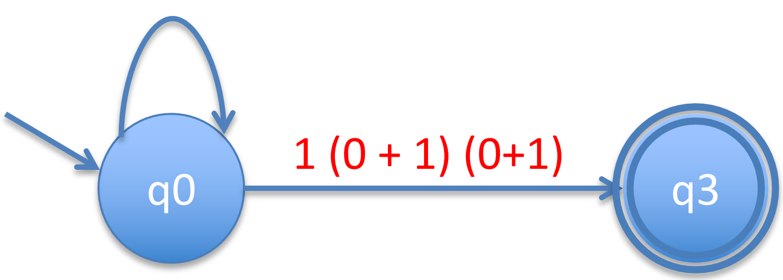
Step 2: Eliminate state q1



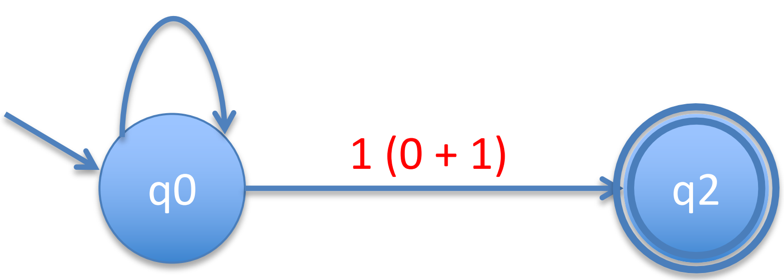
Example



Step 3a: eliminate q2



Step 3b: eliminate q3



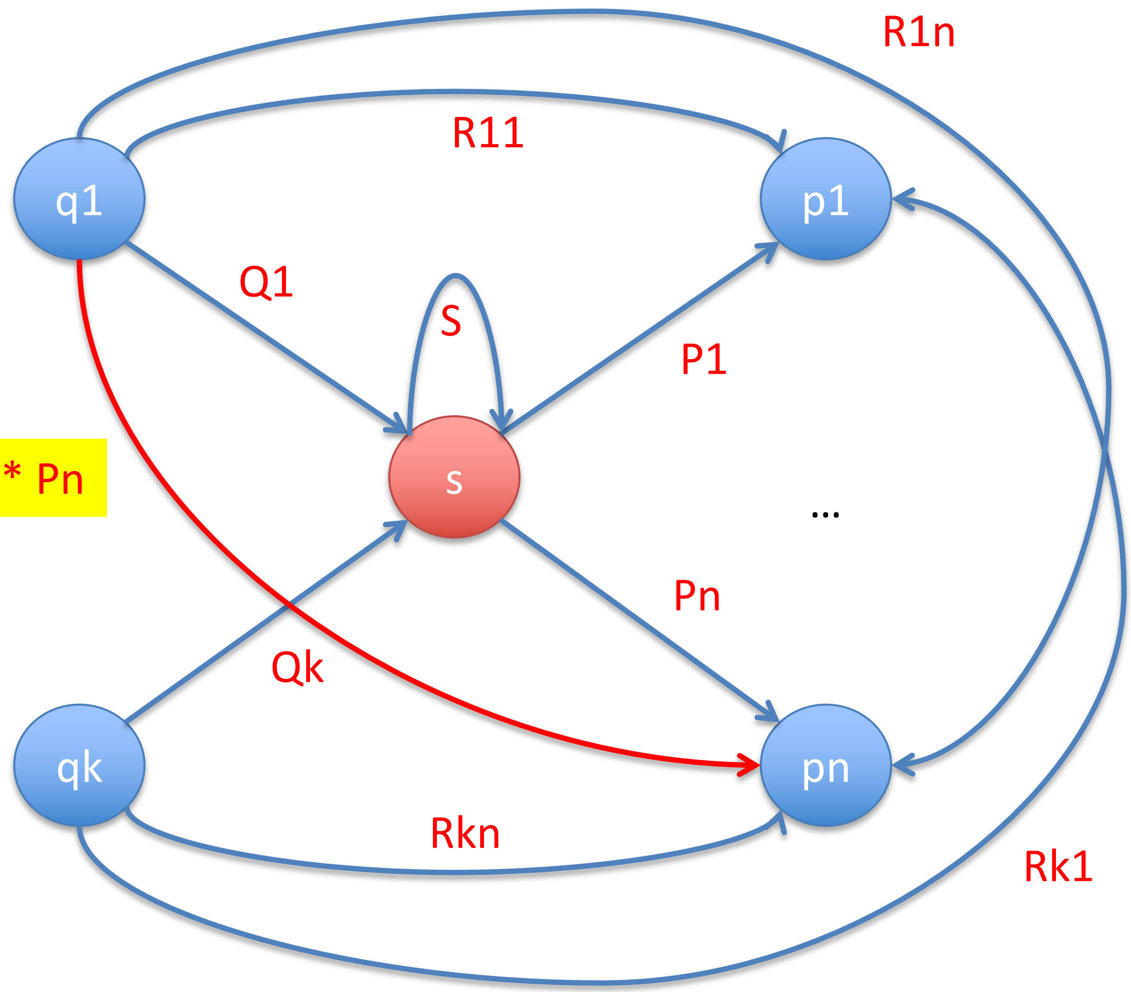
The resulting regular expression:

$(0 + 1)^* \, 1 \, (0 + 1) \, (0 + 1) \quad + \quad (0 + 1)^* \, 1 \, (0 + 1)$

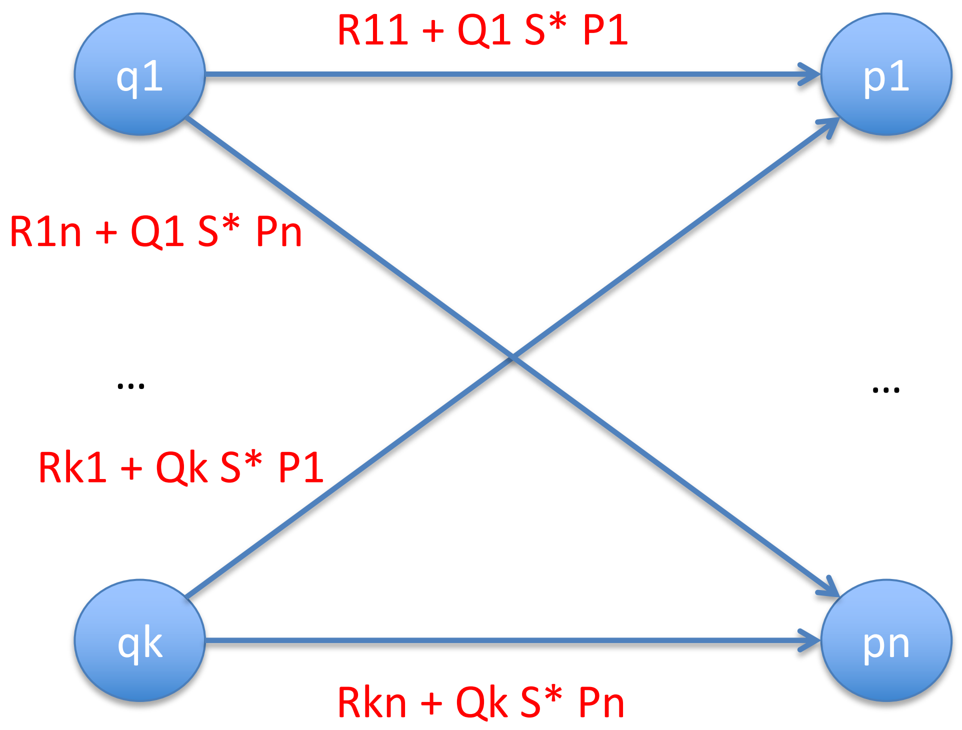
The general algorithm: Elimination of states

We want to eliminate s

R1n + Q1 S* Pn



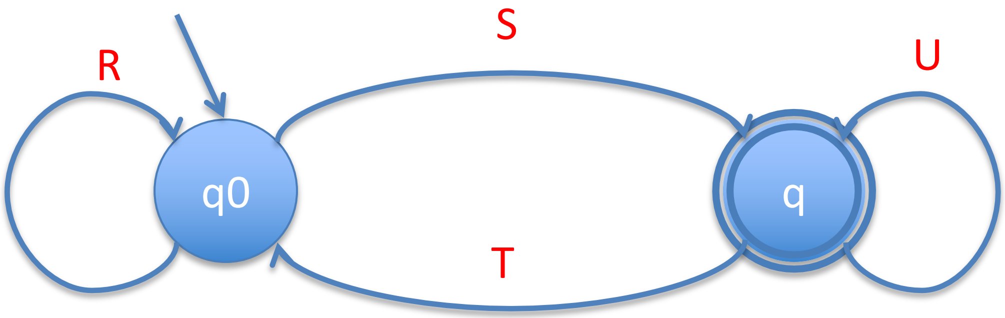
The general algorithm: Elimination of states



The general algorithm

Given a finite automaton we shall

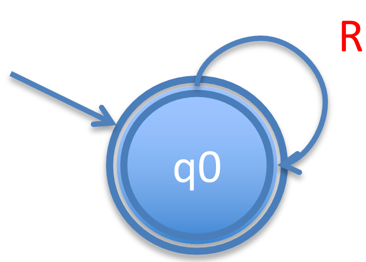
- For each of its final states q we eliminate all states except for the initial state q0 and the final state q
- the resulting automaton has the generic form



The resulting regular expression is
(R + S U* T)* S U*
if q0 and q are distinct states

The general algorithm

- If q0 = q then the state elimination algorithm will result in an automata of the generic form



The resulting regular expression is simply R*

Complexity: $O(n^3 4^n)$

Let's do it!

Exercise 3.2.3

- Convert the following DFA to a regular expression:

	0	1
p	s	p
q	p	s
r	r	q
s	q	r

p is the initial as well as the final state

ALGEBRAIC PROPERTIES

Commutative and associative laws

- Commutative law for union
  - $L+M = M+L$
- Associative law for union
  - $(L + M) + N = L + (M + N)$
- Associative law for concatenation:
  - $(L\ M)\ N = L\ (M\ N)$
- Commutative law for concatenation does *not* hold:
  - $L\ M \neq M\ L$

- Commutative law for addition
  - $x+y = y + x$
- Associative law for addition
  - $(x + y) + z = x + (y + z)$
- Associative law for multiplication
  - $(x * y) * z = x * (y * z)$
- Commutative law for multiplication
  - $x * y = y * x$

Identity and annihilator laws

- Identity for union
  - $\emptyset + L = L + \emptyset = L$
- Identity for concatenation:
  - $\varepsilon L = L \varepsilon = L$
- Annihilator for concatenation
  - $\emptyset L = L \emptyset = \emptyset$

- Identity for addition
  - $0 + x = x + 0 = x$
- Identity for multiplication
  - $1 * x = x * 1 = x$
- Annihilator for multiplication
  - $0 * x = x * 0 = 0$

Distributive and idempotent laws

- Left distributive law
  - $L (M + N) = L M + L N$
- Right distributive law
  - $(M + N) L = M L + N L$
- Idempotent law for union:
  - $L + L = L$

- Distributive law
  - $x * (y + z) = x * y + x * z$
- There is no need to distinguish between left and right distributivity for arithmetic
- We do *not* have idempotent laws for addition:
  - $x + x \neq x$

Laws for closure operator

- $(L^*)^* = L^*$
- $\emptyset^* = \varepsilon$
- $\varepsilon^* = \varepsilon$
- $L^+ = L\,L^* = L^*\,L$
- $L^* = L^+ + \varepsilon$
- $L^? = L + \varepsilon$

Let's do it!

Exercise 3.4.2

- Prove or disprove the following statements
  a) $(R+S)^* = R^*+S^*$
  b) $(RS+R)^*R = R(SR+R)^*$

- (We will do the rest of exercise 3.4.2 later)

BY THE WAY …
REGULAR EXPRESSIONS IN TOOLS

Regular expressions in Tools

- Regular expressions are used for manipulating text:
  - Unix tools and scripting: grep, awk, perl, …
  - Programming languages: Java, C, Python, …
  - Text editors: emacs, vim, …
  - Databases: MySQL, Oracle, PostgreSQL, …
- All these tools have a similar syntax for regular expressions
  - Concatenation: ab ➔ `ab`
  - Alternation: a + b ➔ `a|b`
  - Kleene star: a* ➔ `a*`
- But there are also many derived operators to make life easier!

Extended regular expressions

- Extended syntax allows us to match:
  - Any given character: `[abc]` `[0-9]`
  - Any character except given: `[^abc]` `[^0-9]`
  - One or more instances: `a+` `(abc)+`
  - Zero or one instances: `a?` `(abc)?`
  - Exact number of times: `(abc){3}`
  - Between 3 and 5 instances: `(abc){3,5}`
  - The start/end of the string: `^` `$`
  - Any character: `.`
  - A special character: `\*` `\?` `\\`

READING MATERIAL AND EXERCISES

Reading material and exercises

- Covered in the lecture today:
  - HMU chapter 3: pages 85-91, 98-102 and 115-121
- Topic of the next lecture on Regular Languages:

  Equivalence results for regular languages

  to be based on HMU section 2.3, 2.5 and 3.2
- Exercises for today:
  - Writing and understanding regular expressions: HMU 3.1.1 (a,b), 3.1.4 (a,b)
  - From DFA to regular expressions: HMU 3.2.3
  - Algebraic laws: HMU 3.4.2

The big picture