

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force, SPACE and Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

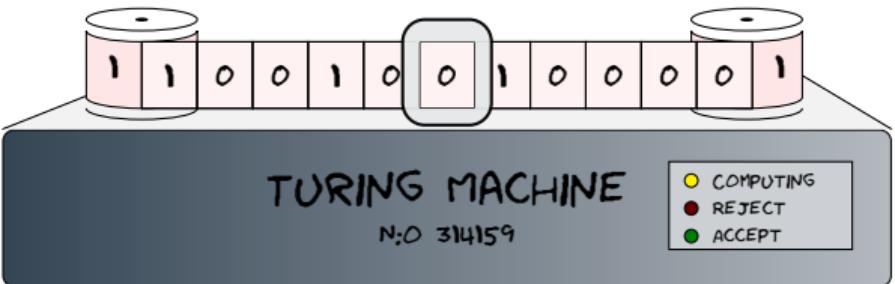
TÖL301G

Formal Languages and Computability

Fall 2021

Esa Hyytiä

University of Iceland



- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Lecture 0

Course Introduction

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Course Topics:

The course deals with various basic topics in computer science with the aim of better understanding what tasks can be solved with computers.

1. Finite state machines, regular languages and grammars
 2. Push-down automata, context-free languages and grammars
 3. Turing machines, general languages and grammars, and their basic properties
- ▶ Connections to *decision problems* and proving unsolvability of such problems
 - ▶ Complexity classes P and NP, and NP-complete

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Learning Outcomes:

After completing the course, the student should

1. Know how to classify formal languages with reference to grammars and state machines, and to be able to make formal deductions about their properties
2. Be able to argue formally
 - ▶ why some computational tasks are inherently more complex than others, and
 - ▶ why some are unsolvable
3. Know about the main models for computability, their inherent limitations, and how one model may be simulated by another

► **Required:**

TÖL104G Mathematical Structures for Computer Science

Exemptions will be made for i) students who have completed a BSc degree in another subject and are completing a degree in computer science and ii) students who have a very strong foundation in mathematics.

► **Recommended:**

- ▶ STÆ101G Mathematical Analysis IA or
- ▶ STÆ104G Mathematical Analysis I or
- ▶ STÆ105G Practical Mathematical Analysis

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Lecturer:

Esa Hyytiä

room xyz at Gróska, 3rd floor

✉ esa@hi.is

(☎ 525 4603)

Teaching assistants:

(to be announced) ✉ username@hi.si

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

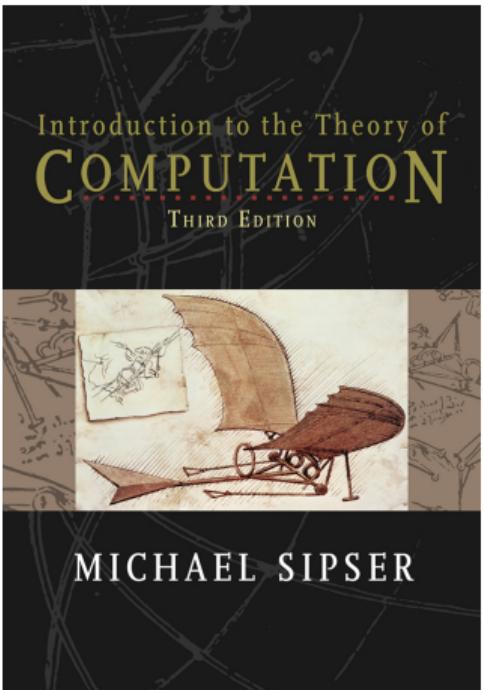
11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Textbook:

Introduction to the Theory of Computation by Michael Sipser, 3rd Edition,
Cengage, 2012.



Tentative Course Schedule

Week	Topics	(Book)	
1	24.8 Course introduction, preliminaries finite automata	(0) (1.1)	0 Introduction 1 DFA
2	31.8 Nondeterminism	(1.2)	2 NFA
3	7.9 Regular expressions	(1.3)	3 Regexp
4	14.9 Nonregular languages	(1.4)	4 Nonregular L
5	21.9 Context-Free grammar	(2.1)	5 Context-Free
6	28.9 Pushdown automata	(2.2)	6 Pushdown
7	5.10 Turing machines	(3.1 - 3.3)	7 Turing
8	12.10 Decidability 19.10 -"-	(4.1) (4.2)	8 Decidability
9	26.10 Reducibility	(5.1,5.3)	9 Reducibility
10	2.11 Time complexity (measuring, P and NP)	(7.1-7.3)	10 Time
11	9.11 NP-completeness	(7.4)	11 Complete
12	16.11 SAT, PSPACE and Probabilistic TMs	(8.1-8.2, 10.2)	12 Brute Force, SPACE and Probabilities
13	23.11 Review		13 Review

Figures in brackets refer to the textbook.

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Other resources:

You may find the following lectures helpful:

https://www.youtube.com/playlist?list=PLbtzT1TYeoMjNOGEiaRmm_vMIwUAidnQz



Course Organization:

	Mon	Tue	Wed	Thu	Fri
8:20-9:00					
9:10-9:50					
10:00-10:40					
10:50-11:30					
11:40-12:20					
12:30-13:10					
13:20-14:00					
14:10-14:50					
15:00-15:40					
15:50-16:30		Lect. (zoom) -			
16:40-17:20					
17:30-18:10					

Lectures:

Tue 15:50 - 18:10 on Zoom?

Tutorials:

- - online on Zoom?

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Weekly routine (this might change!)

1. Lectures:

- ▶ On Tuesday afternoon in Zoom / classroom
- ▶ Will be followed by **short quizzes** that students will return during the lecture

2. Tutorial classes:

- ▶ Once a week (4 groups?), each 2×40 minutes
- ▶ Online using Zoom or in classroom?
- ▶ Example problems will be provided for students to be solved (with the help of teachers)
- ▶ Students **can work** together to solve these tasks
See collaboration later

3. Homework assignments:

- ▶ Will be (typically) posted on Tuesdays
- ▶ Return assignments via Gradescope
- ▶ The deadline will be on the Tuesday one week later
 - ▶ Except the first round DL will be delayed by one week!

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Evaluation:

1. **Final exam** counts as 50%

2. **Homework assignments** count as 50%:

- ▶ Weekly assignments are graded in 0-5 (scaled)
- ▶ The 10 highest weekly grades are counted

3. **Quizzes during the lectures:**

- ▶ 5 points \times 12 weeks, in total 60 points available
- ▶ Quizzes increase your final grade by up to 10%

$$\text{bonus} = \left\lfloor \frac{\max\{\text{pts}, 40\}}{4} \right\rfloor$$

Other:

- ▶ A student must get at least grade 5 in the final exam to pass the course
- ▶ No help material is allowed in the exam
(however a “cheat sheet” will be provided)

Forum:

- ▶ We use Ed for course related discussions:
<https://edstem.org/us/courses/12015>

Collaboration:

- ▶ Students are encouraged to discuss problems together
- ▶ Students who work together on homework assignments still need to submit individual solutions. **They also need to specify who they worked with.**
(e.g. provide email addresses)
- ▶ **It is not allowed to simply share, copy, duplicate or steal solutions!** If teachers observe such, they will lower the grades for the relevant assignments. Contact the lecturer if you are unsure about what is allowed.

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

On Homework Assignments

We will use (mainly?) Gradescope for homework assignments.

Solutions to homework assignments should be readable and include the name of the student, student id number and email address. Late delivery will not be accepted.

If a student thinks that a grading has an error, he/she may request a reassessment. In such cases, a brief description of what is considered to be incorrectly graded is needed. The deadline for making comments is one week from the date when assignments were returned.

Note: Some problems will be auto-graded.

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Preliminaries



“Is this (rasp)berry edible?” (a problem)

Our approach:

1. Formalize the question:
 - encode all relevant berries into strings
2. Devise a computational model or grammar rules that answer the question
 - ▶ Yes, it is edible!
 - ▶ No, it is not

The first step is often assumed to be given/feasible

The second step, whether such a model or grammar exists, is the primary question!

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Computational models are *abstract models of computers*:

- ▶ A finite numbers of states
- ▶ Some transition rules that depend on the input string
- ▶ Goal is to provide a “yes-no” answer at the end

The big question:

*“What are the fundamental capabilities
and limitations of computers?”*

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Two closely related theories:

- ▶ *Complexity theory*:
 classify problems as easy and hard ones
- ▶ *Computability theory*:
 classify problems as solvable and unsolvable

Automata theory: definitions and properties of mathematical models of computation

- ▶ Finite automaton:
 text processing, compilers, and hardware design
- ▶ Context-free grammar:
 programming languages and AI

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Recap:

Mathematical notions and terminology

Section 0.2 of the book

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Statements

- ▶ Let Q and P denote some (mathematical) statements

- ▶ Notation:

$Q \Rightarrow P$ “If Q is true, then P is true”
“ Q implies P ”

$Q \Leftrightarrow P$ Both $Q \Rightarrow P$ and $P \Rightarrow Q$

- ▶ The latter is often written as

“ Q if and only if P ”, or simply “ Q iff P ”

and must be proven both in *forward* and *reverse* direction

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Sets

Defining sets:

$$A = \{1, 2, 4, 8\} \quad (\text{a finite set})$$

$$B = \{x : \text{<condition>}\} \quad (\text{by condition})$$

Then we write

 $x \in A \quad \text{if } x \text{ is from set } A \text{ (similarly, } x \notin A)$
 $A \subseteq B \quad \text{if } A \text{ is a subset of } B \text{ when } x \in A \Rightarrow x \in B$
 $A \subset B \quad \text{if } A \text{ is a proper subset of } B \text{ (i.e., } A \neq B)$
Cartesian product:

$$A = X \times Y = \{(x, y) : x \in X \text{ and } y \in Y\}$$

Sets of numbers:

Natural numbers: $\mathcal{N} = \{1, 2, 3, \dots\}$

Integer numbers: $\mathcal{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$

Binary numbers: $\{0, 1\}$

Real numbers: \mathcal{R}

Standard Set Operations

Union	$A \cup B = \{x : x \in A \text{ or } x \in B\}$
Intersection	$A \cap B = \{x : x \in A \text{ and } x \in B\}$
Complement	$A^c = \{x : x \notin A\}$
Set minus	$A \setminus B = \{x : x \in A \text{ and } x \notin B\}$

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

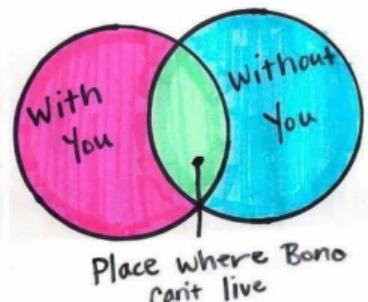
10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Venn diagram:



Example: Consider events in probability theory (e.g., a dice)

Boolean logic

Two values:

- ▶ True and False,
- ▶ 1 and 0
- ▶ ...

Negation	(NOT)	$\neg x$
Conjunction	(AND)	$x \wedge y$
Disjunction	(OR)	$x \vee y$
Exclusive or	(XOR)	$x \oplus y$

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Integers, division and remainder

Division and remainder: (cf. programming)

Let x and y be some non-negative integer numbers.

$$\frac{x}{y} = q + \frac{r}{y}, \quad \text{where } q = \lfloor x/y \rfloor \quad (\text{quotient})$$

$$r = q \bmod y \quad (\text{remainder})$$

Thus, $0 \leq r < y$, and $x = qy + r$.

Integer division C programming language

```
unsigned long x,y,q,r;
...
q = x / y;      /* the quotient of x/y */
r = x % y;      /* and the remainder */
```

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Congruence classes and modulo notations

Congruence classes: (cf. mathematics)

Two integers $x, y \in \mathbb{Z}$ are *equivalent modulo b*

$$x \equiv y \pmod{b} \iff x = y + kb \text{ for some } k \in \mathbb{Z}$$

Example: $1 \equiv 1011 \pmod{5}$

Note that mod is also used as binary operator:

$a \bmod b$: remainder when a is divided by b

Modular arithmetics:

Def. $\mathbb{Z}_p = \{0, 1, 2, \dots, p-1\}$

(integers modulo p)

All standard operations (modulo p):

Example: \mathbb{Z}_4

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

\times	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Sequence is an ordered list

- ▶ In math, sequences are often infinite

$$a_1, a_2, a_3, \dots$$

- ▶ In this course, typically finite but arbitrarily long

Example sums

Consider the two sums where $a_i \in \mathbb{Z}$,

$$\lim_{n \rightarrow \infty} \sum_{i=1}^n a_i \quad \text{vs.} \quad \sum_{i=1}^n a_i$$

The former can be infinite, whereas the latter is never!

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

[0 Introduction](#)[1 DFA](#)[2 NFA](#)[3 Regexps](#)[4 Nonregular \$L\$](#) [5 Context-Free](#)[6 Pushdown](#)[7 Turing](#)[8 Decidability](#)[9 Reducibility](#)[10 Time](#)[11 Complete](#)[12 Brute Force,
SPACE and
Probabilities](#)[13 Review](#)

Tuples

n-tuple is an ordered list of n elements

$$X = (a_1, a_2, \dots, a_n)$$

- ▶ Informally, a “vector” or a “record”
- ▶ Each element may have unique “type”
- ▶ For example, “ $\{name, sex, age, email\}$ ”

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Functions and relations

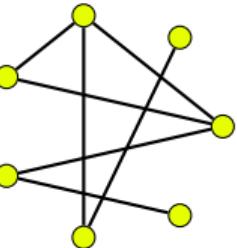
- ▶ Function $f(x) = y$ maps x to y
- ▶ Notation: $f : D \rightarrow R$ (from domain to codomain)

Injection ("one-to-one"):	at most one element $x \in D$ leads to each $y \in R$
Surjection ("onto"):	for every $y \in R$, there exists $x \in D$ s.t. $f(x) = y$
Bijection:	both injection and surjection ($f^{-1}(y)$ exists for $y \in R$)

Equivalence relation:

1. Reflexive, xRx
2. Symmetric, $xRy \Rightarrow yRx$
3. Transitive, xRy and $yRz \Rightarrow xRz$

Divides a set to *equivalence classes* (partitioning)



- ▶ $G = (V, E)$, where V denotes vertices and E edges
- ▶ Node degree = number of edges
- ▶ G is connected if a path exists between every node pair
- ▶ Different types of graphs:
 - ▶ Directed vs. undirected
 - ▶ Weighted graphs
 - ▶ Trees
- ▶ Some examples:
 - ▶ Directory structure, road networks, communication networks
 - ▶ Shortest path algorithms and TSP
 - ▶ Node coloring problems
 - ▶ Percolation models

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Basic proof techniques

1. Proof by construction:

- ▶ “Does an object with certain properties exist?”
- ▶ Show existence simply by constructing a desired object

2. Proof by induction:

- ▶ Show that a statement Q is true for $n = 1$
- ▶ Assuming that Q holds for all $i = 1, \dots, n$, show that then it holds also for $n + 1$ (induction step)
- ▶ Conclude that Q holds for every positive (finite) n

3. Proof by contradiction:

- ▶ Assume Q is true
- ▶ Use that “fact” to derive a contradiction
- ▶ Conclude that the initial assumption must be false

Also, “*a counterexample*” shows that a statement is false

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Alphabet and Strings

Definition 1 (Alphabet)

Alphabet Σ is a finite set of symbols.

For example:

Binary numbers: $\Sigma = \{0, 1\}$

Hexadecimal numbers: $\Sigma = \{0, 1, \dots, 9, a, b, \dots, f\}$

Definition 2 (String)

String w over alphabet Σ is a finite sequence of symbols

$$w = a_1 a_2 \dots a_n, \quad \text{where } a_i \in \Sigma$$

An empty string is denoted with ϵ

Definition 3

Length of string w is the number of symbols it contains

$$|w| := n \quad \text{if} \quad w = a_1 \dots a_n$$

The length of the empty string is zero: $|\epsilon| = 0$

Example: Let $\Sigma = \{0, 1\}$ be the alphabet. Then

- ▶ 0
- ▶ 1011
- ▶ 11110000

are all strings. However, “000000 . . .” is **not a string** (but an infinite sequence).

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Working with strings

Basic operations:

- ▶ Reverse string, w^R

$$(a_1 \dots a_n)^R = a_n \dots a_1$$

- ▶ Concatenation of two strings: xy

$$xy = \overbrace{a_1 \dots a_n}^{=x} \overbrace{b_1 \dots b_m}^{=y}$$

- ▶ Concatenation with itself: x^k (k times)

$$x^k = \overbrace{xx \cdots x}^{k \text{ times}}$$

Lexicographic order: (for strings)

- ▶ Alphabet Σ has some (often natural) order
- ▶ $x < y$ if the first unequal symbol is “smaller” in x than in y
 - ▶ If needed, a shorter string is “padded” with virtual symbols that are smaller than any actual symbol
 - ▶ For example, $00 < 000$

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Definition 4 (Language)

Language L is a set of strings.

Examples: prefix condition: $\{w \mid w \text{ starts with } 0\}$

all strings:

finite set:

$$\{w = a_1 \dots a_n \mid a_i \in \Sigma \text{ for all } i\}$$
$$\{01, 10\}$$

Hence

- ▶ Normal set operations can be applied (union, intersection, set minus)
- ▶ Language is either (i) a finite or (ii) countably infinite set
 - ▶ Never an uncountable set (why?)

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

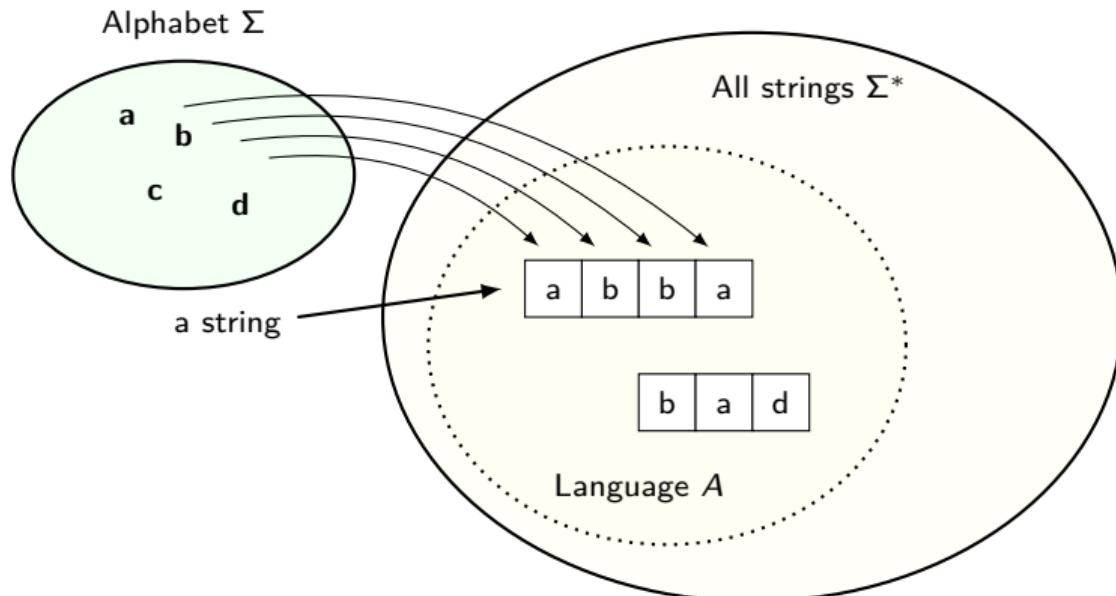
13 Review

Alphabet, Strings and Languages

TÖL301G

E. Hyttiä

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review



- ▶ Alphabet Σ : a finite set of symbols
- ▶ String w : a finite sequence of symbols, $w = a_1 a_2 \cdots a_n$
- ▶ Language A : set of strings (either finite or countably infinite)

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Lecture 1

Deterministic Finite Automata (DFA)

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

1.1 Finite Automata

Deterministic Finite Automata (DFA):

"Set of states and transitions between them"

- ▶ Described by a state diagram (e.g., Fig. 1.4)
 - ▶ Circle vs. double circle
 - ▶ Start state and transitions
- ▶ cf. Markov chains

Definition 5 (DFA (Def. 1.5))

Deterministic finite automaton is defined by 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- ▶ Q is a finite set of states
- ▶ Σ is a finite alphabet
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ is the transition function
- ▶ q_0 is the start state
- ▶ $F \subseteq Q$ is the set of accept states

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example

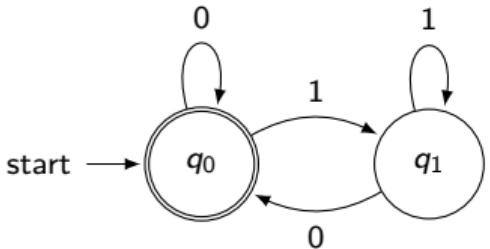


Figure: DFA to check whether a binary number is even.

What are

- ▶ Finite set of states Q ?
- ▶ Alphabet Σ ?
- ▶ Transition function $\delta : Q \times \Sigma \rightarrow Q$?
- ▶ Start state q_0 ?
- ▶ Set of accept states F ?

Computation with DFA

Definition 6

DFA M accepts w if a sequence of states r_0, \dots, r_n exists such that

1. $r_0 = q_0$ (start state)
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n - 1$, (transitions)
3. $r_n \in F$ (accepted state)

Definition 7

M recognizes language A if $A = \{w \mid M \text{ accepts } w\}$.

Definition 8 (Def. 1.6)

Language is regular if some finite automaton recognizes it.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0	Introduction
1	DFA
2	NFA
3	Regexps
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Regular operations

Let A and B denote two languages.

The regular operations are the following:

Union: $A \cup B := \{x \mid x \in A \text{ or } x \in B\}$

Concatenation: $A \circ B := \{xy \mid x \in A \text{ and } y \in B\}$

Star: $A^* := \{x_1 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Two theoretical results

Theorem 1 (Thm 1.25)

Regular languages are closed under union.

Proof:

See the book. □

Theorem 2 (Thm 1.26)

Regular languages are closed under concatenation.

Proof:

Later (see the book). □

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

- ▶ Today's topics:
 1. Course organization and some preliminaries
 2. DFA
- ▶ Answer to Quiz 1 as soon as possible :)
- ▶ Homework 1 will be available in **Canvas**
 - ▶ Return by xyz (submit the necessary files to gradescope!)
- ▶ Questions? Email to esa@hi.is or use our discussion forum at Ed

Welcome to the course!

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Lecture 2

Nondeterministic finite automaton (NFA)

Recap of Lecture 1

In the previous lecture, we

- ▶ first refreshed the mathematical notation,
 - ▶ Sets (union, intersection, ...)
 - ▶ Graphs
 - ▶ Functions
 - ▶ Equivalence relation
 - ▶ Boolean logic
- ▶ Then discussed DFA (defined by a 5-tuple),
- ▶ Regular languages
 - ▶ (DFA exists that recognizes the language)
- ▶ ... and regular operations
 - ▶ Union
 - ▶ Concatenation
 - ▶ Star
- ▶ that all were *closed* in the class of regular languages, i.e., the result is also a regular language

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

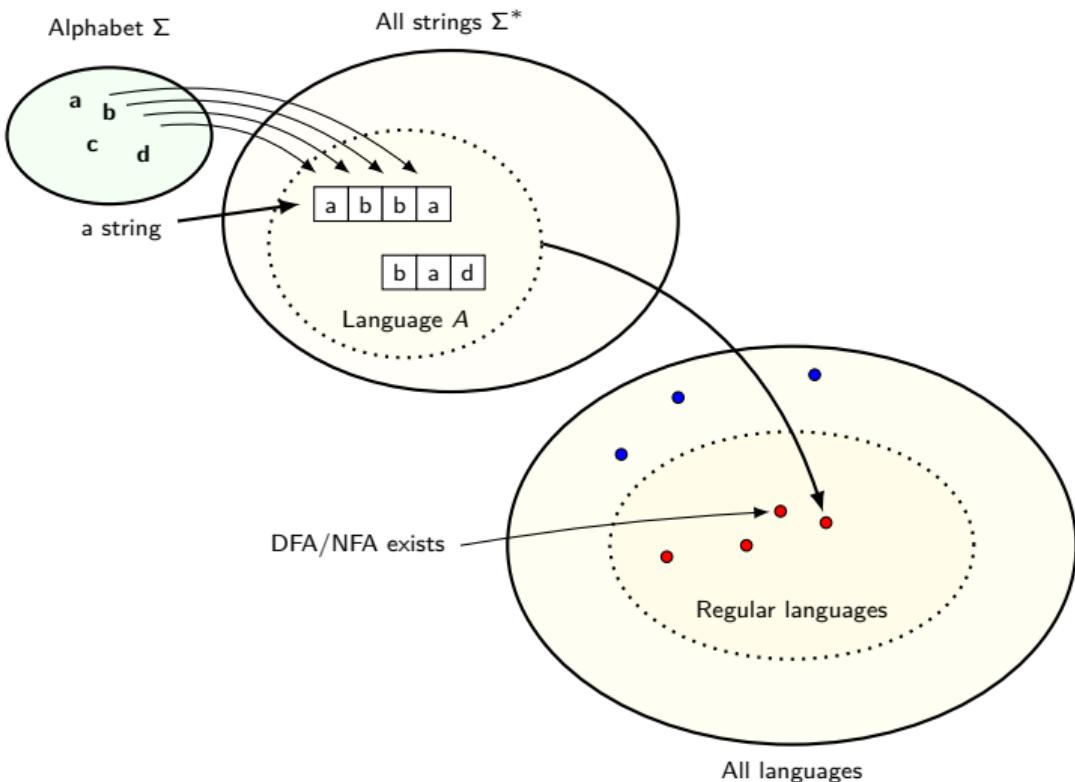
10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Landscape where we are



0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Intersection of two regular languages

Definition 9

Define the intersection of two regular languages A and B as

$$A \cap B \triangleq \{x \mid x \in A \text{ and } x \in B\}.$$

(that is, the standard intersection of two sets)

Theorem 3

The class of regular languages is closed under intersection.

Proof:

(Sketch:) Construct a DFA with state space $Q = Q_1 \times Q_2$, i.e., $q = (q_1, q_2) \in Q$ if $q_1 \in Q_1$ and $q_2 \in Q_2$.

Transitions are as with A and B ,

$$\delta((q_1, q_2), x) = (\delta_1(q_1, x), \delta_2(q_2, x)).$$

Start state is $(q_1^{(0)}, q_2^{(0)})$.

The set of accept states is those where $q_1 \in F_1$ and $q_2 \in F_2$. □

Deterministic Finite Automata (DFA)

- ▶ Finite set of states
- ▶ Deterministic transitions according to input string
 - ▶ Each symbol induces a fixed transition
- ▶ String is *accepted* if the final state is an accept state

Nondeterministic Finite Automata (NFA)

- ▶ Multiple (parallel) transitions possible
 - ▶ Each symbol may lead to different states
 - ▶ Q: how to choose the “right” way?
- ▶ String is *accepted* if . . . ?

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

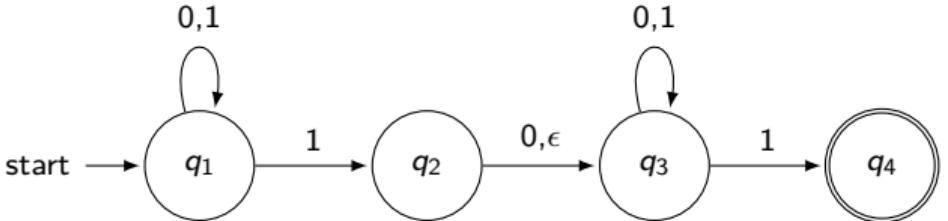
11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Example NFA (Fig. 1.27)



- ▶ DFA: exactly one transition per symbol
- ▶ NFA: zero, one, or many transitions per symbol!
- ▶ NFA: also arrows with symbol ϵ
 - ▶ NFA has multiple ways to proceed!

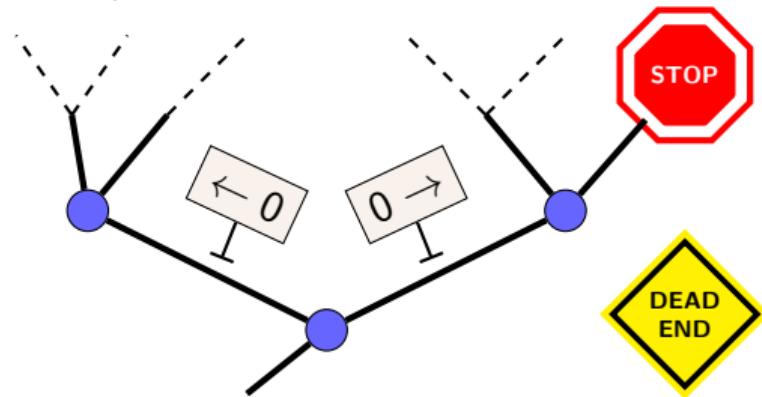
0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Computing with NFA

- ▶ Multiple ways . . . should we

1. Choose one of them?
2. **Or all of them?**
3. (Or something else?)

(probabilistic Markov models)
 ⇐ This one!



- ▶ Split the machine to multiple copies and follow all “tracks” in parallel
- ▶ If no transition for given symbol, machine dies
 - ▶ ϵ : duplicate machine for **every** such transition
- ▶ Accept, if **any copy** is in an accept state

“NFA finds a path to an accept state if possible”

Deterministic vs. nondeterministic computation

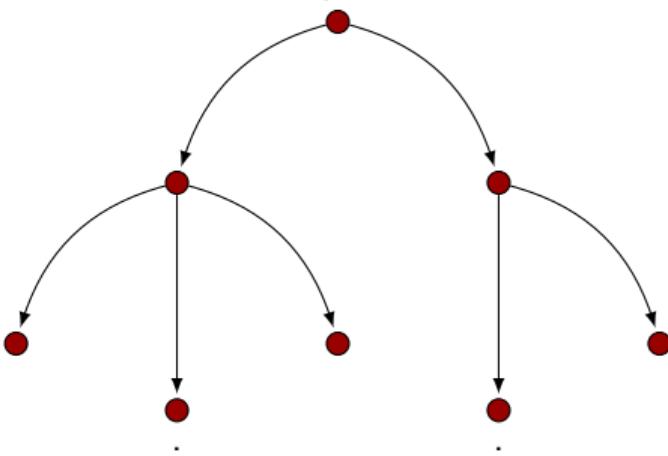
TÖL301G

E. Hyttiä

Deterministic
computation



Nondeter-
ministic
computation



0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Why NFAs

- ▶ Every NFA can be converted into an equivalent DFA
- ▶ ... while construction of NFA is sometimes easier
- ▶ ... and NFA can be significantly smaller

Example 4

Define language A as “all bit strings having 1 as the third last bit”.

1. Design a NFA that recognizes A
2. Design a DFA that recognizes A

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example: (fig. 1.31)

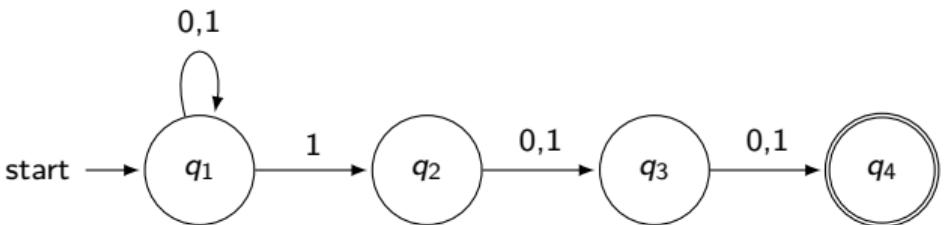


Figure: The NFA N_2 recognizing A .

Note the role of q_1

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

Example: (fig. 1.32)

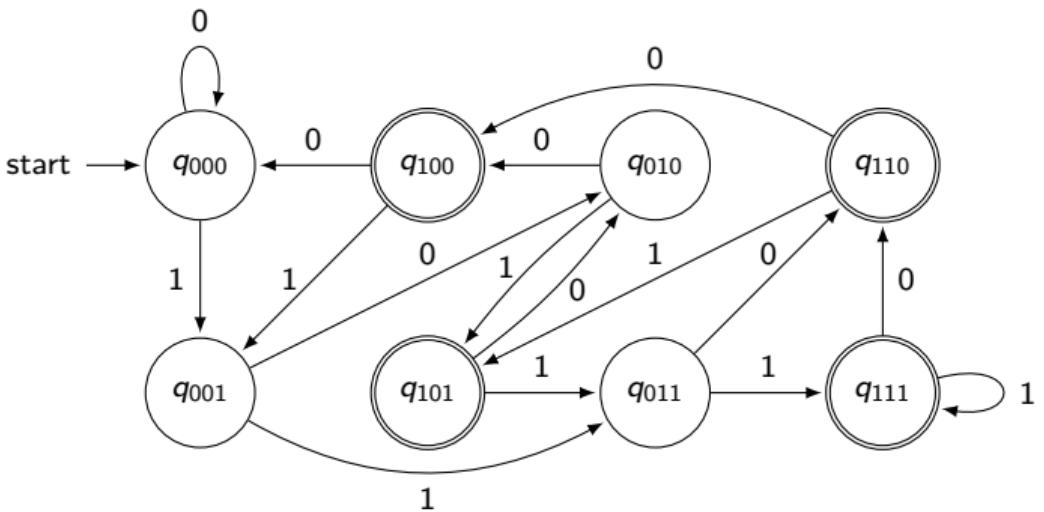


Figure: The DFA M_2 recognizing A .

DFA has more states and it is harder to grasp

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

NFA: (Def. 1.37)

Notation:

- ▶ $\Sigma_\epsilon \triangleq \Sigma \cup \{\epsilon\}$
- ▶ $\mathcal{P}(Q)$ denotes the **power set** of Q

Definition 10 (NFA (Def. 1.37))

A nondeterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- ▶ Q is a finite set of states
- ▶ Σ is a finite alphabet
- ▶ $\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$ is the transition function
- ▶ $q_0 \in Q$ is the start state
- ▶ $F \subseteq Q$ is the set of accept states.

Formally, only the transition function δ changes.

This allows simultaneous transitions to different states!

Definition 13 (Computation with NFA)

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over Σ . N accepts w if we can write w as $w = y_1y_2\dots y_m$, where each y_i is a member of Σ and a sequence of states r_0, \dots, r_m exists in Q with the following conditions:

1. $r_0 = q_0$,
2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, \dots, m - 1$,
3. $r_m \in F$.

“NFA accepts if it can move from the start state to an accept state with a given input string”

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorems!

1. Every NFA has an equivalent DFA
 - ▶ ... and every DFA is also NFA
2. A language is regular iff some NFA recognizes it
 - ▶ ... as with DFA(!)
3. Closure under the regular operations
 - ▶ Union
 - ▶ Concatenation
 - ▶ Star

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Every NFA has an equivalent DFA

Definition 14

Two NFAs are equivalent if they recognize the same language.

Remark: Every DFA is also an NFA.

Theorem 5 (Thm. 1.39)

Every NFA has an equivalent DFA.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Proof:

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A . We construct DFA $M = (Q', \Sigma, \delta', q'_0, F')$ that also recognizes A .

Assume first that N has no ϵ transitions.

1. $Q' = \mathcal{P}(Q)$ (power set)

2. For $R \in Q'$ and $a \in \Sigma$, let

$$\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}.$$

Alternatively,

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

3. $q'_0 = \{q_0\}$

4. $F' = \{R \in Q' \mid R \text{ contains an accept state}\}$

□

For complete proof with ϵ transitions, see book (page 56).

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Closure under the regular operations

Theorem 6 (1.45)

The class of regular languages is closed under union.

Proof:

Let A_1 and A_2 be two regular languages with NFAs

$$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1), \quad \text{recognize } A_1, \text{ and}$$

$$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2), \quad \text{recognize } A_2.$$

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = Q_1 \cup Q_2 \cup \{q_0\}$; all states in Q_1 and Q_2 + new state q_0
2. q_0 is the start state of N
3. The set of accept states is $F = F_1 \cup F_2$
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\epsilon$,

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1, \\ \delta_2(q, a) & q \in Q_2, \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \epsilon, \\ \emptyset & q = q_0 \text{ and } a \neq \epsilon. \end{cases}$$

Theorem 7 (1.47)

The class of regular languages is closed under concatenation.

Proof:

Let A_1 and A_2 be two regular languages and

$$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1), \quad \text{recognize } A_1, \text{ and}$$

$$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2), \quad \text{recognize } A_2.$$

Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$.

1. $Q = Q_1 \cup Q_2$; all states in Q_1 and Q_2
2. The same start state q_1 as with N_1
3. The same set of accept states as with N_2 , $F = F_2$
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\epsilon$,

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \setminus F_1, \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \epsilon, \\ \delta_1(q, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \epsilon, \\ \delta_2(q, a) & q \in Q_2. \end{cases}$$



0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 8 (1.49)

The class of regular languages is closed under star.

Proof:

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 . Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^* .

1. $Q = \{q_0\} \cup Q_1$; same states + new start state
2. The start state is q_0
3. The set of accept states is $F = \{q_0\} \cup F_1$
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\epsilon$,

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \setminus F_1, \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \epsilon, \\ \delta_1(q, a) \cup \{q_1\} & q \in F_1 \text{ and } a = \epsilon, \\ \{q_1\} & q = q_0 \text{ and } a = \epsilon, \\ \emptyset & q = q_0 \text{ and } a \neq \epsilon. \end{cases}$$



- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Steps ...

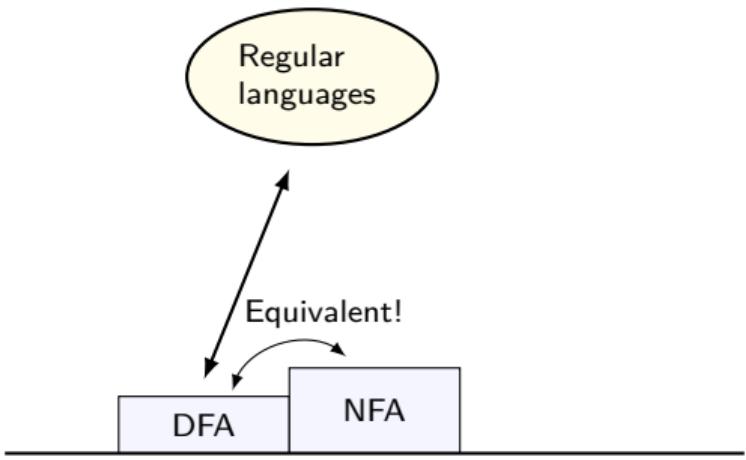


Figure: A stairway to ...

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

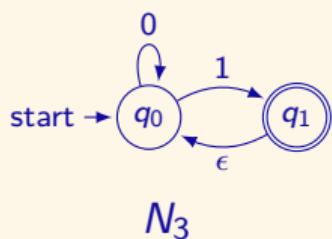
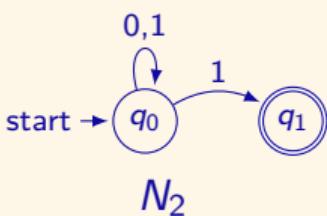
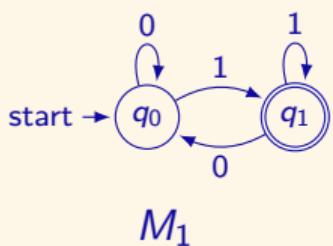
12 Brute Force,
SPACE and
Probabilities

13 Review

Some example automata (for quiz!)

Example 9

Below we have state diagrams of three example automata:



Convention:

- ▶ With DFA, we explicitly state all transitions
- ▶ With NFA, transition missing \rightarrow reject!

- ▶ Both DFA and NFA are *computational models*
 - ▶ Finite set of states
 - ▶ Each input symbol triggers a state change
 - ▶ New state a function of (i) the current state and
(ii) input symbol
- ▶ DFA operates in deterministic fashion (single copy)
- ▶ NFA operates nondeterministically
(multiple copies, parallelism!)
- ▶ **DFAs and NFAs are equivalent**
 - ▶ Every DFA is an NFA
 - ▶ Any NFA can be converted into an equivalent DFA

Thanks!

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Lecture 3

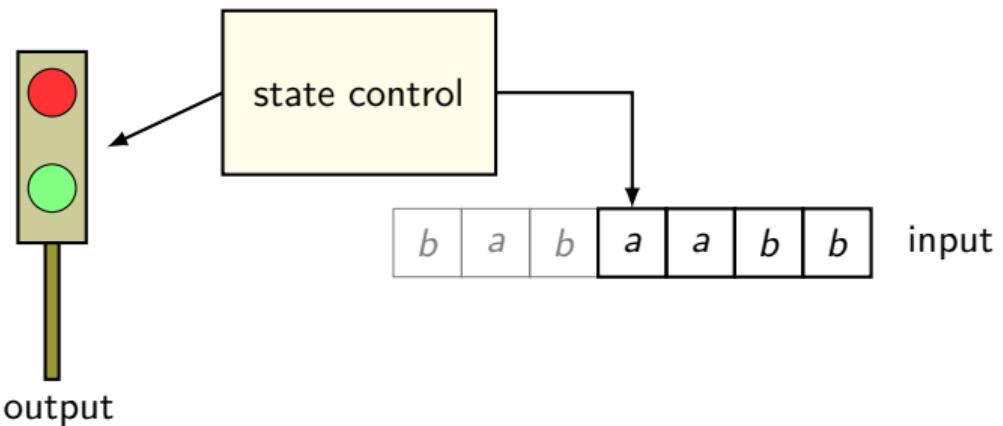
Regular expressions

Last week, we discussed the *nondeterministic finite automaton* (NFA):

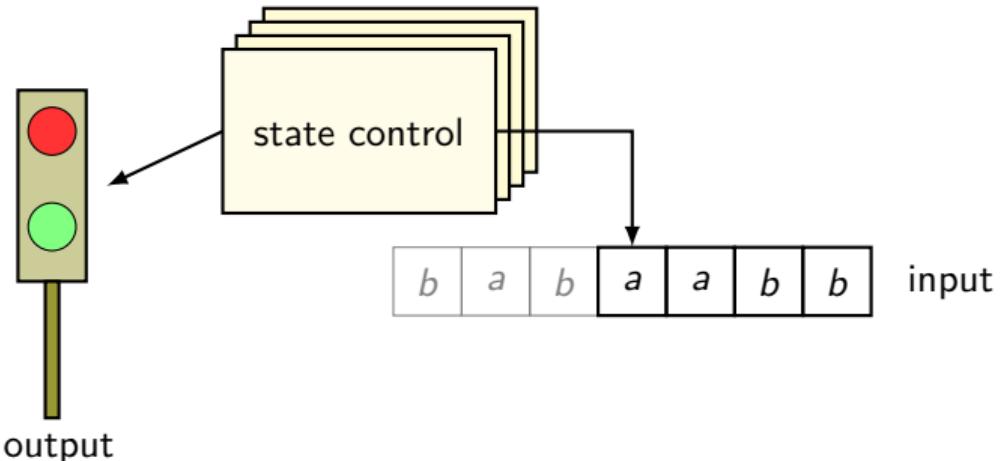
- ▶ NFAs are generalization of DFAs, with
 1. multiple parallel transitions, and
 2. “immediate” transitions by ϵboth triggering multiple copies of the automaton
- ▶ Absence of transition for a given symbol
⇒ delete “this copy of automaton”
- ▶ NFA recognizes string w if *any* copy is in *an accept state*
- ▶ Main result: **DFA_s and NFA_s are equivalent:**
 1. Every DFA is NFA trivially
 2. Every NFA can be converted to an equivalent DFA

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review



- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexp
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review



0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Steps ...

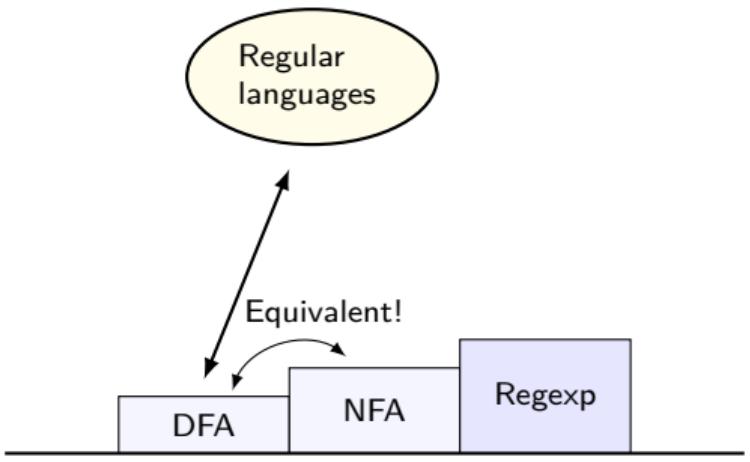


Figure: A stairway to ...

- Arithmetic expressions such as

$$4 \times (5 + 5) + 2$$

yield numerical values

- Boolean expressions (C syntax) such as

$$(x|y) \& 1$$

produce Boolean values

- **Regular expressions** describe languages
(i.e. sets of strings)

Example 10

Regular expression $(0 \cup 1)0^*$ defines a language of strings that

1. start with 0 or 1
2. and are followed by any number of zeroes (incl. none)

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

$$A = (0 \cup 1)0^*$$

- ▶ Symbols 0 and 1 refer to sets $\{0\}$ and $\{1\}$
 - ▶ $(0 \cup 1) \Rightarrow \{0, 1\}$
- ▶ Similarly,

$$0^* \Rightarrow \{0\}^*$$

a language consisting of strings with any number of zeroes

- ▶ Concatenation symbol “dot”, \circ , is often omitted
 - ▶ Similarly as \times from arithmetic expressions
- ▶ Complete/explicit version would thus be

$$A = (\{0\} \cup \{1\}) \circ \{0\}^*$$

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Example uses of REs

Regular expressions are important also in practice:

- ▶ Text search/pattern matching/string modifications
 - ▶ awk, grep, perl, sed, ...
- ▶ Text editors such as vi and vim

Simple examples: (unix command line)

```
egrep "^[a-zA-Z0-9]*@[hi.is$]"    # print email addresses
sed '/^ *#/d' file.sh             # remove comments
ls IMG-[0-9]*.JPG | \
  sed -e 's/IMG-\([0-9]*\)\.JPG/mv IMG-\1.JPG img_\1.jpg/g' | \
  bash
```

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Precedence order

- ▶ In arithmetic,
 1. Exponentiation has precedence over multiplication,
 2. which has precedence over addition/subtraction

$$a + b \cdot c = a + (b \cdot c).$$

- ▶ With regular expressions, the precedence order is
 1. Star operation $(\cdot)^*$ first (closure)
 2. Then concatenation \circ
 3. and finally union \cup

unless specified otherwise with parentheses (...)

Arithmetic	Regexp
a^b	A^*
$a \cdot b$	$A \circ B$
$a + b$	$A \cup B$

Definition of the regular expression (RE)

Definition 15 (RE (Def. 1.52))

R is a regular expression (RE) if

1. *$R = a$ for some a in the alphabet Σ ,*
2. *$R = \epsilon$,*
3. *$R = \emptyset$*
4. *$R = (R_1 \cup R_2)$, where R_1 and R_2 are regular expressions*
5. *$R = (R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, OR*
6. *$R = (R_1^*)$, where R_1 is a regular expression.*

Note:

- ▶ ϵ represents a language with one string; **empty string**
- ▶ \emptyset represents a language with **no strings**
- ▶ Regular expressions get defined recursively!

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example

Example 11

Let $\Sigma = \{a, b\}$ and $R = a \circ (a \cup b)^*$. Is R a regular expression?

Let's check against the definition Def. 1.52

$R = R_1 \circ R_2$, where $R_1 = a$ and $R_2 = (a \cup b)^*$

- R is thus regular if R_1 and R_2 are regular (case 5)

- 1) R_1 is regular expression (Case 1) ✓
 - 2) R_2 is regular expression if $R_3 = a \cup b$ is (case 6)
 - 2.1) R_3 is a regular expression if a and b are (case 4)
Both a and b are regular expressions (case 1) ✓
-

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Note: 4-6 of Def. 1.52, R is defined in terms of R_1 and R_2 :

1. Defining something in terms of itself
 \Rightarrow circular definition? (invalid!)
2. Here R_1 and R_2 are “smaller” than R
3. Therefore, we actually have an *inductive definition*

Examples inductive definitions

► Set of natural numbers, \mathbb{N}

1. Zero, denoted by 0, is a natural number
2. Every natural number has a successor;
If n is a natural number, so is $n + 1$

► Sometimes also term *recursive definition* is used

- Especially with functions, e.g., the factorial

$$f(n) = \begin{cases} 1, & \text{if } n = 0, \\ n \cdot f(n - 1), & \text{otherwise} \end{cases}$$

► Recall also Pascal's triangle and combinations:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{for } 0 < k < n$$

i.e.,

$$f(n, k) = \begin{cases} 1, & \text{if } k = 0 \text{ or } k = n, \\ f(n-1, k-1) + f(n-1, k), & \text{otherwise} \end{cases}$$

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0	Introduction
1	DFA
2	NFA
3	Regexps
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

- ▶ Each regular expression R defines a *language*
- ▶ Often R is implicitly used to denote also the corresponding language
- ▶ When distinction is important, $L(R)$ refers to the language R describes

Def. 1.52 includes concatenation, union and star operators. Also other operators can be useful.

Additional notation:

$$R^+ \triangleq RR^*$$

one or more copies of R

$$R^k \triangleq \underbrace{R \circ R \circ \dots \circ R}_{k \text{ times}}$$

exactly k copies of R

Example:

$$\cup_{i>0} R^i = R^+.$$

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Example 12 (Example 1.53)

Consider the following REs:

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$
3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ has } 001 \text{ as a substring}\}$
4. ...
5. $01 \cup 10 = \{01, 10\}$
6. ...

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example

Recall: Length of string is the number of symbols it contains

See [here](#)!

Example 13

Let the alphabet be $\Sigma = \{0, 1\}$, and suppose language L is defined as

$$L = \{w \mid \text{length of } w \text{ is a multiple of three}\}$$

1. What is the regular expression for L ?
2. Determine a DFA that recognizes L .
3. What would be the minimal NFA for the same purpose?

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Equivalency between Regular Languages and REs

Theorem 14 (1.54)

A language is regular iff some regular expression (RE) describes it.

Recall that iff (if and only if) means that

1. L is regular if some R describes it (if)
2. L is not regular, if no R describes it (only if)

0	Introduction
1	DFA
2	NFA
3	Regexps
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Lemma 15 (1.55)

If a language is described by a regular expression, then it is regular.

Proof:

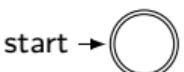
Recall: L is regular if an NFA exists that recognizes it.

Then consider the six parts of the definition of regexps:

1. Case $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$ and the following automaton recognizes it,



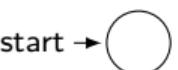
2. Case $R = \epsilon$: Now $L(R) = \{\epsilon\}$ and the following automaton recognizes it,



0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Proof continues

3. Case $R = \emptyset$: Here $L(R) = \emptyset$ and the following automaton recognizes it



4. Cases $R = R_1 \cup R_2$, $R = R_1 \circ R_2$ and $R = R_1^*$: We construct a new NFA N for R using NFAs N_1 and N_2 corresponding to R_1 and R_2 , respectively.

\Rightarrow A corresponding NFA exists for every RE!

Lemma 16 (1.60)

If a language L is regular, then a regular expression describing it exists.

The proof proceeds as follows:

1. L is regular means that a DFA M recognizing it exists.
2. Define a generalized nondeterministic finite automaton (GNFA)
3. Convert M into GNFA with $k + 2$ states (2 new states)
4. Show steps how to remove one state from GNFA without affecting its behavior. (State, other than start and stop states).
5. GNFA with two states gives the regular expression corresponding to M

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Generalized NFA (GNFA)

Idea: *In GNFA, transitions are regular expressions a substring must match with*

Definition 16

GNFA is a 5-tuple $(Q, \Sigma, \delta, q_{start}, q_{accept})$, where

- ▶ Q is a finite set of states,
- ▶ Σ is the input alphabet,
- ▶ $\delta : (Q \setminus \{q_{accept}\}) \times (Q \setminus \{q_{start}\}) \rightarrow \mathcal{R}$ is the transition function and \mathcal{R} the set of all regular expressions,
- ▶ q_{start} is the starting state, and
- ▶ q_{accept} is the (single) accept state.

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

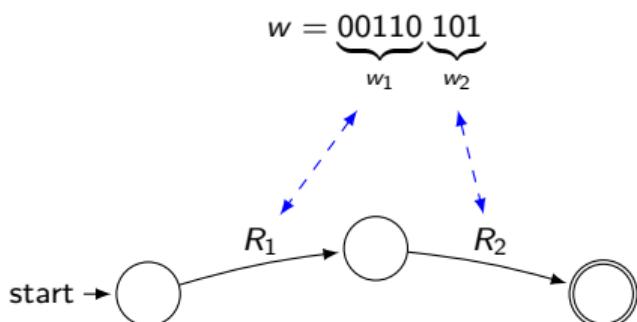
13 Review

Computing with GNFA

Definition 17

A GNFA accepts string w in Σ^* if $w = w_1 w_2 \cdots w_k$, where

1. each $w_i \in \Sigma^*$, and
2. a sequence of states q_0, q_1, \dots, q_k exist such that
 - 2.1 $q_0 = q_{\text{start}}$
 - 2.2 $q_k = q_{\text{accept}}$
 - 2.3 For each i , $w_i \in L(R_i)$ where $R_i = \delta(q_{i-1}, q_i)$
(R_i is the RE on arrow from q_{i-1} to q_i)



0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Note:

- ▶ δ maps a pair of states (q_i, q_j) to a regular expression, i.e., δ can be represented as a matrix of REs
- ▶ There is exactly one accept state, and there is NO transitions away from it
- ▶ Similarly, no transition leading back to start state exists
- ▶ Computation works so that if a *block* in input string matches a regular expression, then the corresponding transition can be followed
- ▶ Automaton is **nondeterministic**

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Step 1: DFA \rightarrow GNFA

1. Add new start state q_{start} and create immediate ϵ transitions from it to the old start state.
2. Add new accept state q_{accept} , and add ϵ transitions from all old accept states to it.
3. If old DFA has multiple transitions from q_i to q_j , combine those with union.
4. If no transition existed from q_i to q_j , add transition with label \emptyset .

As a result, we have all entries for the δ matrix and GNFA with two additional states.

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

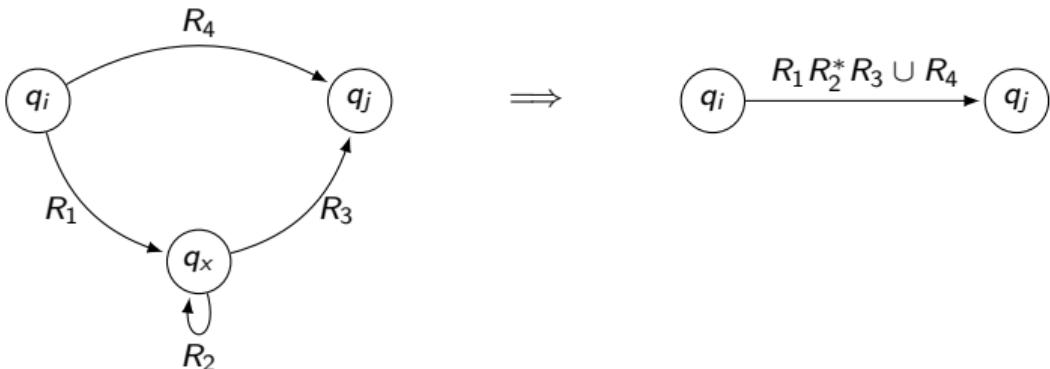
12 Brute Force,
SPACE and
Probabilities

13 Review

Step 2: Remove one state from GNFA

Next suppose GNFA has $k > 2$ states.

1. Choose state, say q_x , that is not start or accept state.
2. Then for all pairs of states (q_i, q_j) , with $q_i \notin \{q_{\text{accept}}, q_x\}$ and $q_j \notin \{q_{\text{start}}, q_x\}$, do the following compensation:



which compensates for the removal of q_x

The new GNFA accepts the same language, but has one state less!

Step 3: Repeat step 2 until $k = 2$ states left

- ▶ Remove states until GNFA has only two states:
 q_{start} and q_{accept}
- ▶ The regular expression on transition from start to accept is equivalent to the original DFA.
- ▶ Steps 1-3 show how regular expression can be constructed for any regular language, thus completing the proof.

Corollary 17

Combining Lemmas 1.55 and 1.60 proves Theorem 1.54.

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Relationships ...

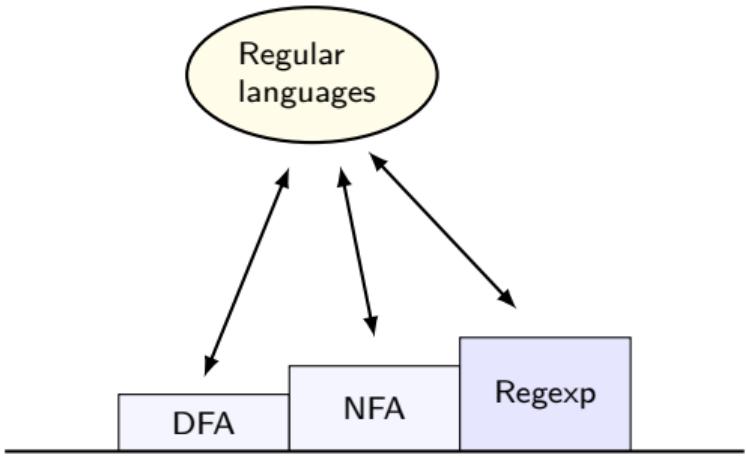


Figure: A stairway to ...

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Number systems

In the **decimal system**, the base is $B = 10$ and the alphabet is $\Sigma = \{0, \dots, 9\}$.

The numbers are represented as

$$x = x_0 + 10x_1 + 10^2x_2 + \dots = \sum_{i=0}^k x_i B^i,$$

where the $x_i \in \Sigma$, i.e., $0 \leq x_i < B$.

In the **binary system**, the base is $B = 2$ and the alphabet is $\Sigma = \{0, 1\}$. The numbers are represented as

$$x = x_0 + 2x_1 + 2^2x_2 + \dots = \sum_{i=0}^k x_i B^i,$$

where the x_i are now binary digits, $x_i \in \{0, 1\}$.

In the **ternary system**, the base is $B = 3$ and the alphabet $\Sigma = \{0, 1, 2\}$.

Basic “blackboard” algorithms for

- ▶ Addition
- ▶ Subtraction
- ▶ Multiplication
- ▶ Division

work the same way in every base!

$$\begin{array}{r} & & & & 1 \\ & 3 & 1 & 4 & 1 & 5 & 9 \\ + & & 2 & 0 & 1 & 1 \\ \hline & & & & 7 & 0 \end{array}$$

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Number systems: divisibility

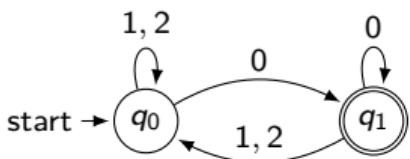
With binary numbers, it is easy to check the divisibility¹ by 2^k :

$$2^k \mid x \Leftrightarrow k \text{ lowest bits are zero.}$$

In general, in base B ,

$$B^k \mid x \Leftrightarrow k \text{ lowest digits are zero.}$$

For example, a DFA checking the divisibility by three with ternary numbers is thus trivial:



Q: What is the corresponding regular expression?

Q: What is the DFA for divisibility by B in base B ?

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

¹The notation $a \mid b$ means a divides b , $b \bmod a = 0$.

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Summary

- ▶ Regular expressions (RE):
 - ▶ Defined recursively
 - ▶ $R_1 \cup R_2$ (union), R^* (star) and $R_1 \circ R_2$ (concat.)
- ▶ DFAs, NFAs and REs are equivalent
 - ▶ Any one of them can be converted to another
- ▶ Language A is *regular* if
 - ▶ DFA recognizing it exists, or
 - ▶ NFA recognizing it exists, or
 - ▶ RE generating it exists
- ▶ DFA and NFA are computational models
 - ▶ RE is a *search pattern* – a string either matches it or not

Thanks!

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Lecture 4

Nonregular Languages

Recap

The previous lecture was about *regular expressions*:

R is a regular expression (RE) if

1. $R = a$ for some a in the alphabet Σ
2. $R = \epsilon$
3. $R = \emptyset$
4. $R = (R_1 \cup R_2)$, where R_1 and R_2 are regular expressions
5. $R = (R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, OR
6. $R = (R_1^*)$, where R_1 is a regular expression

► Last 3 cases were inductive definitions

► Main result:

REs are equivalent to DFAs and NFAs

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- ▶ RE defines a criteria for selecting strings
- ▶ In practice, the steps are as follows:
 1. Let R denote a given RE
 2. Construct a corresponding NFA by parsing R
 - ▶ Similarly as with Lemma 1.55 on slide 84
 - ▶ This involves defining
 - i) States Q
 - ii) Starting state q_0
 - iii) Accept states F
 - iv) Transition function (matrix) δ
(alphabet Σ can be implicit)

Implement a parser in your favorite programming language!

3. Run the NFA for each input string
 - ▶ Simulate the dynamically constructed NFA

Can you implement your own grep program?!

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Example: skeleton code for DFA

TÖL301G

E. Hyttiä

```
static int transition[][]; /* destination states */
static int accept[];      /* true if accept state */
/*
 * state st: state, 1,2,...
 * string str: 1,2,..., null-terminated
 */
int processDFA(int st, const char *str) {
    return *s
    ? processDFA(transition[st][*str], str+1)
    : accept[st];
}
```

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

- ▶ So far we have happily “computed” with DFAs and NFAs
 - ▶ Equivalently, *regular expressions* can be used
 - ▶ Yielding the class of regular languages
- ▶ Clearly for many tasks they are sufficient
 - ▶ We can determine, e.g.
 - ▶ Whether a number is divisible by some (fixed) number
 - ▶ If string has certain substring(s)
 - ▶ If string conforms with some format (cf. cvs files)
 - ▶ and many other similar conditions
 - ▶ **... but how powerful they really are?**
 - ▶ i.e., what kind of languages are not regular?
- ▶ In short,
 1. What questions can be answered with regular languages?
 2. ... and what cannot?

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example

Consider the following languages with $\Sigma = \{0, 1\}$

$$A = \{0^n 1^m \mid n \geq 0, m \geq 0\}$$

$$B = \{(01)^n \mid n \geq 0\}$$

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

Which of them are *regular languages*?

- ▶ From previous lectures: A and B are regular
- ▶ What about C ?

Example problems

Problem	w/ DFA?
Search substrings with wildcards?	
Check if a line from a CSV file has k fields?	
Check if a string includes forbidden words?	
Check if a list is sorted in increasing order?	
Check if a number is less than c	$x < c$
Check if a number is multiple of c	$x \bmod c = 0$
Check if a number is a prime	$x \in \text{PRIMES}$
Verify arithmetic expressions of form	$a + b = c?$
Check if a number corresponds to digits of π	$x \in \{3, 31, \dots\}$
Check if a binary string has correct parity bit?	even # of 1s
Validate the CRC-3-GSM code in binary strings?	-

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

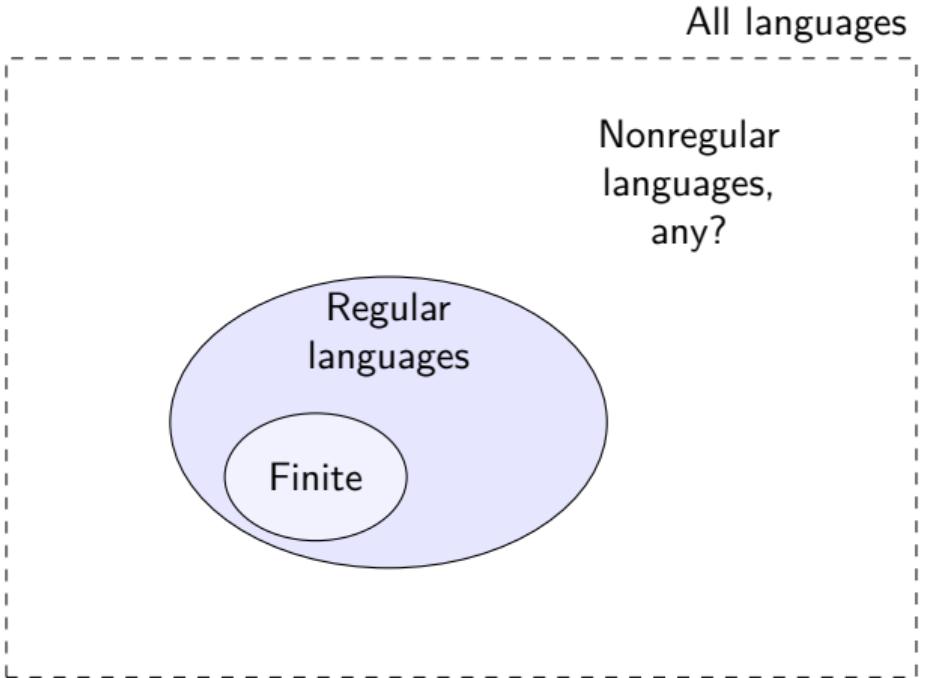
9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review



- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Why some languages are not regular

Let B be some language with alphabet Σ ,

$$B = \{w = a_1 \cdots a_n \mid a_i \in \Sigma\}$$

- ▶ Can we just list all valid strings? ?
- ▶ Language may include infinite number of strings² ?
- ▶ DFAs and NFAs have a finite number of states
- ▶ We can still easily *enumerate* them: ?
- ▶ Alphabet is finite, $m = |\Sigma| < \infty$
- ▶ There is m^i strings of length i , $i = 0, 1, \dots$
- ▶ E.g., with $\Sigma = \{0, 1\}$, we have
 - ▶ 2^i strings of length i ,
 - ▶ $2^{i+1} - 1$ strings of length i or less
- ▶ Show by induction that all languages are regular?! ?

²Cardinality of B is infinite

Proof techniques: Induction (recap)

Basic steps:

1. State a claim: $P(n)$ is true for $n = 1, 2, \dots$
2. Show that the claim holds for the first m instances
3. *Induction hypothesis*: “Assume $P(n)$ holds for all $n \leq m$
4. *Induction step*: Assuming claim is true for $n \leq m$, show that it holds also for $m + 1$
5. It follows that the claim is true for any (finite) n

Note:

- ▶ $P(n)$ is a property, not, e.g., a number
- ▶ By induction, one can show that $P(n)$ is true for any finite n
- ▶ ... but it does not necessarily carry on to limit $n \rightarrow \infty$

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Induction: Example 1

Show that

$$S_2(n) := 1 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(1+2n)}{6} \quad (1)$$

- The claim holds for $n = 1$ as

$$S_2(1) = 1 = \frac{1 \cdot 2 \cdot 3}{6}$$

- Assume that (1) holds for $1, \dots, n$ *(induction hypothesis)*
- Consider the case $n + 1$ *(induction step)*

$$\begin{aligned} 1^2 + 2^2 + \dots + (n+1)^2 &= \frac{n(n+1)(1+2n)}{6} + (n+1)^2 \\ &= \frac{n+1}{6} (n(1+2n) + 6(n+1)) \\ &= \frac{(n+1)(n+2)(1+2(n+1))}{6}. \end{aligned}$$

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- ▶ Let

$$S(n) = \bigcup_{i=1}^n A_i, \quad \text{where} \quad A_i = \left[\frac{1}{i}, 1 \right] \subseteq \mathbb{R}$$

- ▶ Claim $P(n)$ = “ $S(n)$ is a closed set”
- ▶ The claim is clearly true for every n as

$$S(n) = A_n = \left[\frac{1}{n}, 1 \right]$$

- ▶ Induction proof can be also carried out . . .
- ▶ However, $P(\infty)$ is false as

$$\lim_{n \rightarrow \infty} S(n) = (0, 1]$$

which is not a closed set!

Be careful with limits!

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Example: Induction on regularity

Therefore, even though

1. Any finite language is regular
2. Any language B can partitioned into finite sets

$$B_i = \{w \in B \mid |w| = i\}$$

... and therefore, for every n ,

$$A_n \triangleq \bigcup_{i=0}^n B_n$$

is finite and thus regular ... it **does not imply** that

$$B = \lim_{n \rightarrow \infty} A_n = \bigcup_{i=0}^{\infty} B_n$$

is (necessarily) regular!

0	Introduction
1	DFA
2	NFA
3	Regexps
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

To summarize:

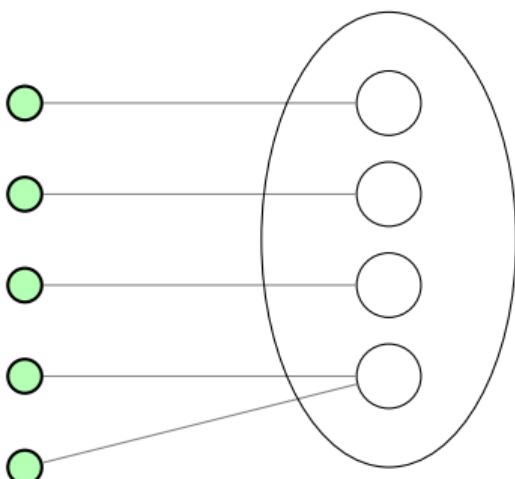
- ▶ Any language with a finite number of strings is regular
 - ▶ Why?
- ▶ Languages with infinite number of strings *may be regular*
 - ▶ For example, $\{w \mid w \text{ is an even number}\}$, etc.
- ▶ But some of them are not regular!
 - ▶ Because no finite automaton can recognize them

Next we will formalize this!

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Pigeonhole Principle

"If n pigeons is placed into m pigeonholes with $n > m$, then at least one pigeonhole has more than one pigeon."



- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexp
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

[0 Introduction](#)[1 DFA](#)[2 NFA](#)[3 Regexp](#)[4 Nonregular \$L\$](#) [5 Context-Free](#)[6 Pushdown](#)[7 Turing](#)[8 Decidability](#)[9 Reducibility](#)[10 Time](#)[11 Complete](#)[12 Brute Force,
SPACE and
Probabilities](#)[13 Review](#)

Pumping Lemma

Proofs that a language is nonregular often use the so-called Pumping Lemma:

Theorem 18 (Pumping Lemma (Def. 1.70))

If A is a regular language, then there is a number p , referred to as the pumping length, where if s is any string in A with $|s| \geq p$, then s may be divided into three parts, $s = xyz$, satisfying the following conditions:

1. For each $i \geq 0$, $xy^i z \in A$
2. $|y| > 0$
3. $|xy| \leq p$

Pumping lemma for finite languages

TÖL301G

E. Hyttiä

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Proof:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes A .

- ▶ Let $p = |Q| = \text{nr. of states in } M$
- ▶ Let $s = a_1 a_2 \cdots a_n \in A$ with $|s| = n \geq p$

Let (r_i) be a sequence of states that M enters when processing s

$$r_1 = q_0, \quad r_{i+1} = \delta(r_i, a_i), \quad i = 1, \dots, n \quad \text{and} \quad r_{n+1} \in F$$

Sequence (r_i) has length $n + 1 \geq p + 1$

- ▶ Among r_1, \dots, r_{p+1} at least one state exists multiple times

Let (r_j, r_k) be one such pair, where $r_j = r_k$ and $j < k \leq p + 1$.

Considering the string s ,

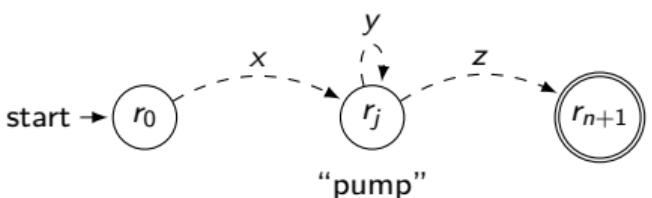
$$s = \overbrace{a_1 \cdots a_{j-1}}^x \overbrace{a_j \cdots a_{k-1}}^y \overbrace{a_k \cdots a_n}^z$$

- ▶ x takes M from q_0 to r_j
- ▶ y from r_j back to r_j
- ▶ z from r_j to an accept state r_{n+1}

Hence, string s can be split into xyz such that

1. M accepts xy^iz with any $i \geq 0$ (the loop from r_j back to it)
2. $|y| > 0$ as $j < k$ ($y = a_j \dots a_{k-1}$ and $|y| = k - j$)
3. $|xy| \leq p$ as $|xy| = k - 1 \leq p$ ($k \leq p + 1$)

□



0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

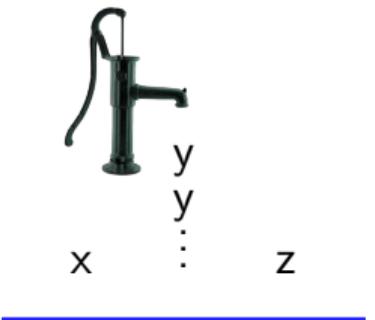


Figure: Pumping lemma: substring y can be “pumped” into given location between x and z without affecting the result of computation.

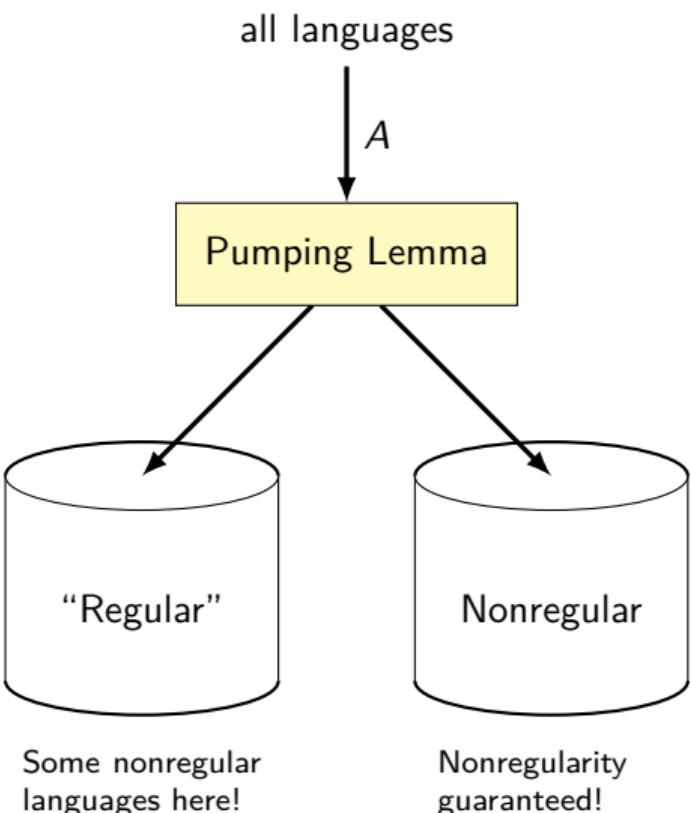
About pumping lemma

- ▶ Due to M. Rabin and D. Scott (1959)
- ▶ The name refers to the fact that every string longer than $p - 1$ has a substring that can be “pumped” to any power
- ▶ The lemma is used to show that a language is nonregular
 - ▶ First one assumes that language is regular
 - ▶ Then the Lemma should hold for every string with length p or more
 - ▶ By choosing a “nasty” string, a contradiction is obtained
 - ▶ i.e., the initial assumption must be wrong
 - ▶ and there is no DFA that could recognize the language
 - ▶ which makes the given language *nonregular*
- ▶ Lemma gives necessary **but not sufficient** conditions:
a language may satisfy all conditions while being nonregular!

Finding a suitable counterexample can be “art”

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review



Relationships ...

TÖL301G

E. Hyttiä

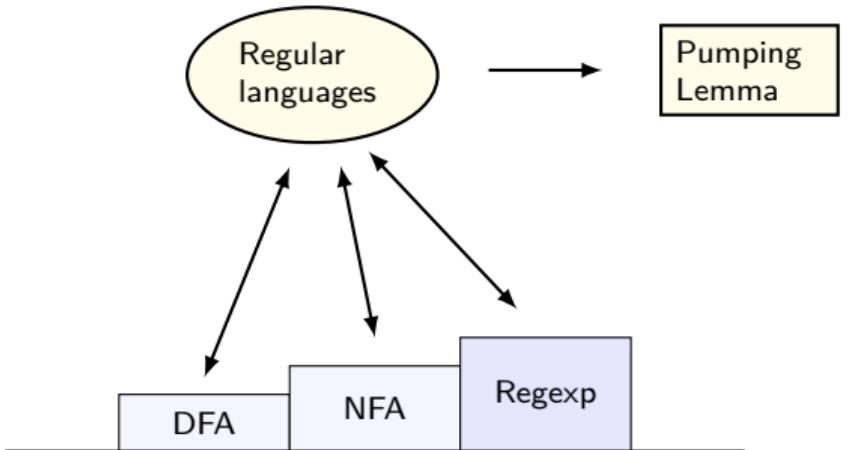


Figure: Stairway to ...

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

0	Introduction
1	DFA
2	NFA
3	Regexps
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

In order to show that a language A is

- ▶ **Regular:** construct a DFA, NFA or Regexp
- ▶ **Nonregular:** assume A is regular and use the pumping lemma to find a contradiction

Example 1.73

Prove that $B = \{0^n 1^n \mid n \geq 0\}$ is nonregular:

- ▶ Assume B is regular \Rightarrow pumping lemma holds for some p
- ▶ Let $s = 0^p 1^p$, so that $|s| = 2p > p$
- ▶ According to pumping lemma, s can be split into 3 pieces, $s = xyz$, such that $xy^i z \in B$ for all $i \geq 0$
- ▶ Then consider 3 possible cases for y :
 1. y **consists solely on 0s**: Then xy^2z has more 0s than 1s, i.e., it does not belong to B , and y cannot be just 0s
 2. y **consists solely on 1s**: Similarly, this is not possible either
 3. y **consists of 0s and 1s**: Then xy^2z has first 0s, then 1s, then 0s again, and finally 1s, and thus it does not belong to B , and consequently y cannot consist of both 0s and 1s either
- ▶ As all three cases lead to a contradiction, the initial assumption, B being regular, must be false

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example 1.74

Prove that $C = \{w \mid w \text{ has equal nr. of } 0\text{s and } 1\text{s}\}$ is nonregular:

Proof:

- ▶ Assume C is regular \Rightarrow pumping lemma holds for some p
- ▶ Let $s = 0^p 1^p$, so that $|s| = 2p > p$
- ▶ According to pumping lemma, s can be split into 3 pieces, $s = xyz$, such that $xy^i z \in C$ for all $i \geq 0$
- ▶ Condition 3: $|xy| \leq p$, i.e., both x and y consists of only 0s
- ▶ Consequently, xy^2z has too many zeroes to be in C , and a contradiction is reached
- ▶ Therefore, the initial assumption of C being regular is false



- ▶ DFAs, NFAs and regular expressions (RE) are equivalent
 - ▶ Any one of them can be converted to another
- ▶ Language A is *regular* if DFA recognizing it exists
 - ▶ ... or NFA
 - ▶ ... or regular expression
- ▶ However, not all languages are regular
 - ▶ If A is not regular, it is *nonregular*
- ▶ To show that A is nonregular, Pumping Lemma is used
- ▶ Proofs by contraction: Assume A is regular, then using the Pumping Lemma a contradiction is reached.

Thanks!

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Lecture 5

Context-Free Languages

The previous lectures discussed *regular languages*:

- ▶ First defined them via *Finite Automata* (DFA and NFA)
- ▶ Then via *Regular Expressions*
- ▶ Equivalent, yield the same class of languages
- ▶ *Pumping Lemma* was used to show *Nonregularity*

Unfortunately, regular languages fail for some simple languages, the prime example being

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

Yet, they work perfectly in many cases, e.g.,

$$\{w \mid w \text{ has an equal number of substrings } 01 \text{ and } 10\}$$

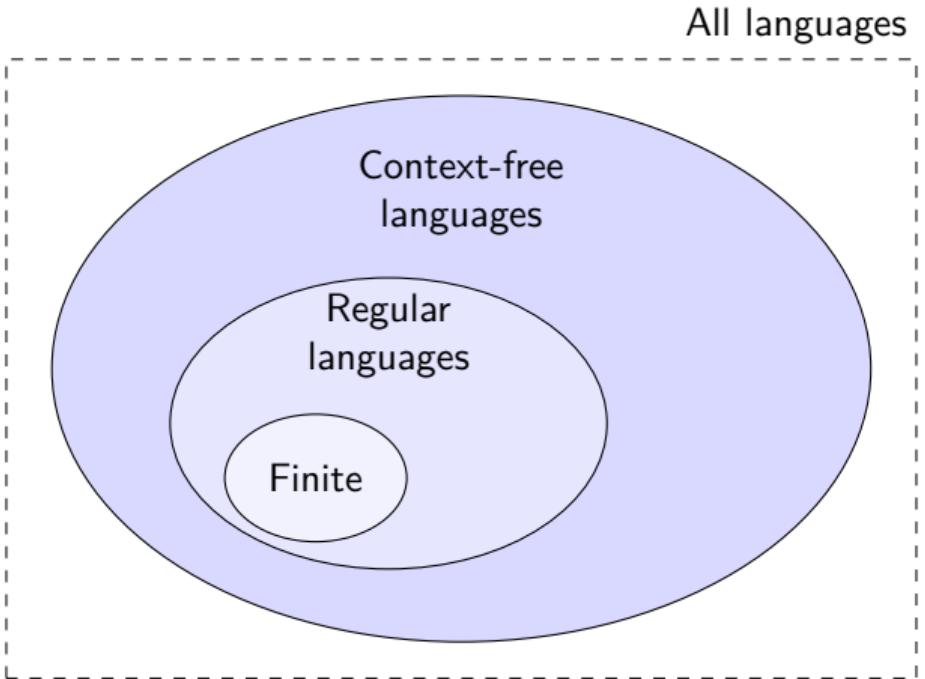
0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0	Introduction
1	DFA
2	NFA
3	Regexps
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

This lecture

Topics for today:

- ▶ Class of *context-free languages* (CFL)
- ▶ CFLs are defined by *context-free grammar* (CFG)
- ▶ CFLs include regular languages (RLs) as a subset
 - ▶ Every regular language can be generated by a CFG
- ▶ CFL are important for *parsing* programming languages



- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

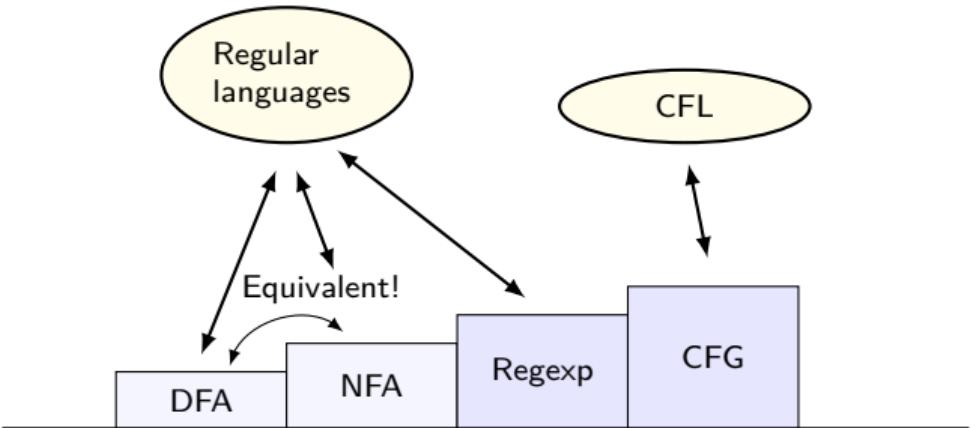


Figure: Stairway to ...

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example: Context-free Grammar

Context-free grammar is defined by a set of recursive substitution rules:

Example 19 (Grammar G1)

$$A \rightarrow 0A1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$

Grammar consists of **substitution rules** (*productions*)

- ▶ Left-hand side has a *variable*
- ▶ Then an arrow
- ▶ Right-hand side is a string
- ▶ String consists of (i) **variables** and (ii) **terminals**

One variable is the so-called **start variable**

- ▶ Convention: start variable is the first rule

The basic steps to *derive* a string:

1. Write down the start variable
2. Repeat until all symbols are terminals:
 - ▶ Find a variable and replace it with an appropriate string

For example, G1 generates strings such as

- ▶ 00#11
- ▶ 000#111
- ▶ L(G) denotes the language of grammar G
 - ▶ In this case, $L(G1) = \{0^n\#1^n \mid n \geq 0\}$.
- ▶ L(G) defined by CFG G is a *context-free language*

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Derivation can be depicted as a parse tree:

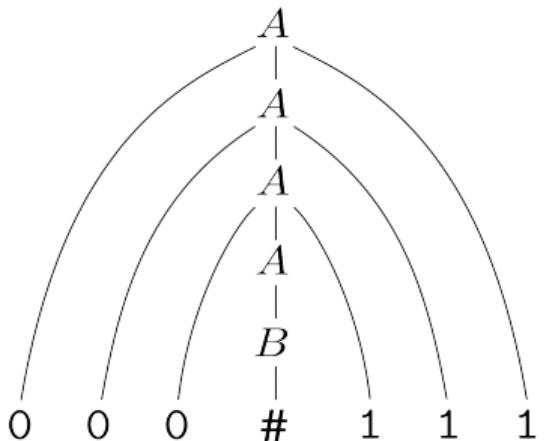


Figure: Figure 2.1 (from book).

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

Example: G2

Grammar G2:

```

<SENTENCE>      → <NOUN-PHRASE><VERB-PHRASE>
<NOUN-PHRASE>   → <CMPLX-NOUN> | <CMPLX-NOUN><PREP-PHRASE>
<VERB-PHRASE>   → <CMPLX-VERB> | <CMPLX-VERB><PREP-PHRASE>
<PREP-PHRASE>   → <PREP><CMPLX-NOUN>
<CMPLX-NOUN>    → <ARTICLE><NOUN>
<CMPLX-VERB>    → <VERB> | <VERB><NOUN-PHRASE>
<ARTICLE>        → a | the
<NOUN>           → boy | girl | flower
<VERB>           → touches | likes | sees
<PREP>           → with

```

- ▶ 10 variables
- ▶ alphabet has 27 symbols (English alphabet + space)
- ▶ 18 rules

a boy sees

the boy sees a flower

a girl with a flower likes the boy

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Definition

Definition 18 (CFG (Def. 2.2))

A **context-free grammar** is a 4-tuple (V, Σ, R, S) , where

- ▶ V is a finite set called the *variables*,
- ▶ Σ is a finite set, disjoint from V , called the *terminals*,
- ▶ R is a finite set of rules, with each rule being a variable and a string of variables and terminals, and
- ▶ $S \in V$ is the *start variable*

Terminology:

- ▶ uAv **yields** uwv , $uAv \Rightarrow uwv$, if $A \rightarrow w$
- ▶ u **derives** v , $u \xrightarrow{*} v$, if
 1. $u = v$, or
 2. a sequence u_1, \dots, u_k exists for $k \geq 0$ s.t.

$$u \Rightarrow u_1 \Rightarrow \dots \Rightarrow u_k \Rightarrow v$$

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example 20 (Example 2.3)

Consider grammar $G_3 = (\{S\}, \{a, b\}, R, S)$ where R is the set of rules

$$S \rightarrow aSb \mid SS \mid \epsilon$$

Questions:

- ▶ What kind of strings G_3 generates?
- ▶ If “ SS ” from the right-hand side is omitted, what would be the strings?

Example 4

Example 21 (Example 2.4)

Consider grammar $G4 = (V, \Sigma, R, \langle \text{EXPR} \rangle)$

- ▶ V is $\{\langle \text{EXPR} \rangle, \langle \text{TERM} \rangle, \langle \text{FACTOR} \rangle\}$
- ▶ Σ is $\{a, +, \times, (,)\}$
- ▶ The rules R are

$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle$$

$$\langle \text{TERM} \rangle \rightarrow \langle \text{TERM} \rangle \times \langle \text{FACTOR} \rangle \mid \langle \text{FACTOR} \rangle$$

$$\langle \text{FACTOR} \rangle \rightarrow (\langle \text{EXPR} \rangle) \mid a$$

- ▶ E.g., strings $a + a \times a$ and $(a + a) \times a$ can be generated with grammar $G4$
- ▶ The parse trees are shown in Figure 2.5

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Example 4 (continued)

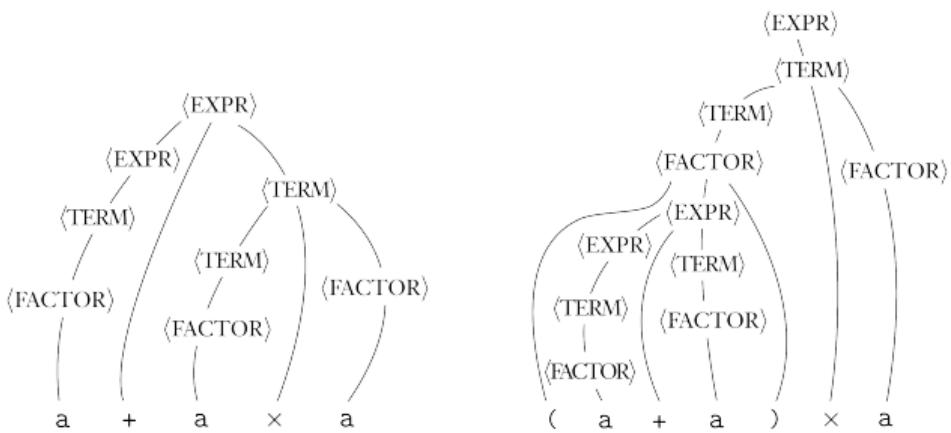


Figure: G4 illustrated (Figure 2.5 from book).

Note:

- ▶ Compilers parse programs, G4 illustrates how arithmetic expressions are parsed
- ▶ Note the precedence rules: multiplication before addition
 - ▶ Unless overwritten with parentheses
- ▶ G4 obeys the standard precedence rules!

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Many CFLs are the union of simpler CFLs

- ▶ Construct CFGs for each piece first (smaller problem)
- ▶ Combine them by
 - ▶ Combine their rules
 - ▶ Add new rule:

$$S \rightarrow S_1 \mid S_2 \mid \dots \mid S_k$$

where the S_i are starting variables of individual cases

Example:

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

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

CFG for a regular language

- ▶ Every regular language has a DFA
 - ▶ DFA can be converted to a CFG (next slide)
- ▶ Any regular language can be expressed by an RE
 - ▶ REs can be converted to a CFG (see later)

CFGs and CFLs are more “powerful” than regular languages

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Designing CFLs: From DFA to CFG

An arbitrary DFA can be converted to an equivalent CFG.
Let 5-tuple $(Q, \Sigma, \delta, q_0, F)$ define the DFA, see [Def. 1.5](#).

-
1. Define variable R_i for each state q_i
 2. Add rule $R_i \rightarrow aR_j$ if $\delta(q_i, a) = q_j$, $a \in \Sigma$
 3. Add rule $R_i \rightarrow \epsilon$ if $q_i \in F$ (accept state)
 4. Define R_0 as the start variable (corresponding to q_0)
-

Why the above procedure yields a CFG that generates the given regular language?

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Designing CFLs: From Regexp to CFG

We can construct a CFG for an arbitrary Regexp as follows:

-
1. If RE is a single operand, $w = \epsilon$ or $w \in \Sigma$, add $\langle \text{RE} \rangle \rightarrow w$
 2. If RE is \emptyset , do nothing
 3. If RE is a union, $R1 \cup R2$, add $\langle \text{RE} \rangle \rightarrow \langle R1 \rangle \mid \langle R2 \rangle$
 4. If RE is a concatenation, $R1 \circ R2$, add $\langle \text{RE} \rangle \rightarrow \langle R1 \rangle \langle R2 \rangle$
 5. If RE is a star, $R1^*$, add $\langle \text{RE} \rangle \rightarrow \langle R1 \rangle \langle \text{RE} \rangle \mid \epsilon$
-

Compare with the definition of the Regexps

Def. 1.52

CFL with two substrings that are linked

- ▶ Prime example $\{0^n 1^n \mid n \geq 0\}$
- ▶ Infinite memory needed to verify “pairs”?
- ▶ No, instead use a rule of form

$$R \rightarrow uRv$$

Recursive structures

- ▶ Strings may contain certain recursive structures
- ▶ See example 2.4:
 - ▶ Anytime symbol a appears, an entire parenthesized expression might appear instead
 - ▶ ... recursively
- ▶ Use a variable to generate such recursive structures

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Theorem 22

The class of context-free languages is closed under the regular operations, union, concatenation, and star.

Proof:

Left as an tutorial exercise. □

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

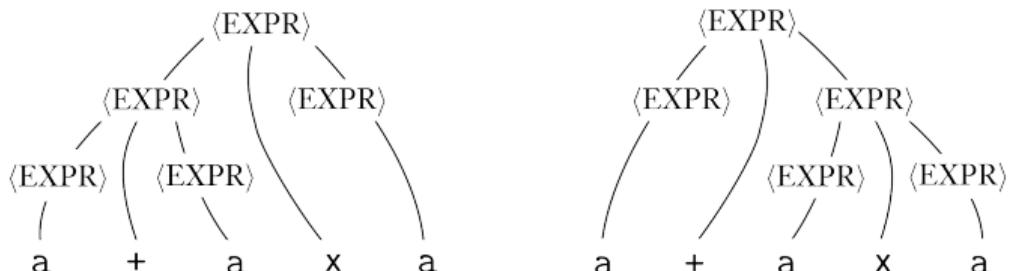
13 Review

Example 5

Example 23 (Grammar G5)

$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle | (\langle \text{EXPR} \rangle) | a$$

G5 generates the string $a + a \times a$ ambiguously:



- ▶ **G5 does not** capture the usual precedence relations
 - ▶ it may group the $+$ before the \times or vice versa
- ▶ **G4** generates exactly the same language
 - ▶ but every generated string has a unique parse tree!
- ▶ **G4** is **unambiguous**, whereas **G5** is **ambiguous**

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Example: mathematical expressions

The following CFG generates a more complete set of arithmetic expressions than G5:

- $\langle \text{EXPR} \rangle \rightarrow \text{number}$
- $\langle \text{EXPR} \rangle \rightarrow (\langle \text{EXPR} \rangle)$
- $\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle$
- $\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle - \langle \text{EXPR} \rangle$
- $\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle$
- $\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle / \langle \text{EXPR} \rangle$

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example

Suppose $\Sigma = \{\epsilon, (,), x, +, -, \times, 0, \dots, 99\}$, and the CFG G has the following production rules:

$$\langle \text{EXPR} \rangle \rightarrow (\langle \text{EXPR} \rangle) \times (\langle \text{EXPR} \rangle)$$

$$\langle \text{EXPR} \rangle \rightarrow x + \langle A \rangle$$

$$\langle \text{EXPR} \rangle \rightarrow x - \langle A \rangle$$

$$\langle A \rangle \rightarrow 0|1|2|\dots|99$$

Questions:

- ▶ What are the variables of the CFG G
- ▶ What mathematical functions correspond to G ?

Ambiguity

Definition 19

*A derivation of a string w in a grammar G is a **leftmost derivation** if at every step the leftmost remaining variable is the one replaced.*

Definition 20 (Def. 2.7)

A string w is derived ambiguously in context-free grammar G if it has two or more different leftmost derivations. Grammar G is ambiguous if it generates some string ambiguously.

- ▶ For some ambiguous grammars an unambiguous grammar generating the same language exists
- ▶ However, some CFLs can be generated only by ambiguous grammars
 - ▶ Such languages are called **inherently ambiguous**

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example

Example 24

The strings of form

$$\{0^n 1^n 2^m 3^m \mid n, m \geq 0\},$$

i.e., the corresponding language clearly belongs to the class of context-free languages.

Write down a formal description of the corresponding CFG!

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Definition 21 (Def. 2.8)

A context-free grammar is in **Chomsky normal form** if every rule is of the form

$$A \rightarrow BC$$

$$A \rightarrow a$$

where a is any terminal and A , B , and C are any variables – except that B and C may not be the start variable. In addition, we permit the rule $S \rightarrow \epsilon$, where S is the start variable.

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 25 (Theorem 2.9)

Any context-free language can be generated by a context-free grammar in Chomsky normal form.

In other words:

- ▶ For any CFL, a CFG in Chomsky normal form exists
(cf. for any regular language, a DFA exists)

We proof this by constructing a compliant CFG from an arbitrary one!

1. **Start state:** Add a new start state S_0 and rule $S_0 \rightarrow S$. Thus the new start state does not appear on the right-hand side.
2. **ϵ -rules:** Remove an ϵ -rule $A \rightarrow \epsilon$, where A is not start state. Then for each rule with A on the right-hand side, we add a new rule with A deleted. This means that
 - ▶ If $R \rightarrow uAv$, add $R \rightarrow uv$
 - ▶ If $R \rightarrow uAvAw$, add $R \rightarrow uvAw$, $R \rightarrow uAvw$, $R \rightarrow uvw$
 - ▶ If $R \rightarrow A$, add $R \rightarrow \epsilon$ (unless it was previously removed)Repeat until no ϵ -rules exists (except for the start state)
3. **Unit rules:** Remove a unit rule $A \rightarrow B$. Then, for every rule $B \rightarrow u$, add a new rule $A \rightarrow u$ (unless this was a unit rule previously removed) Repeat until no unit rules left.
4. **Long rules:** Replace all rules of form $A \rightarrow u_1u_2\dots u_k$, where $k \geq 3$ and each u_i is a variable or terminal symbol, by
$$A \rightarrow u_1A_1, \quad A_1 \rightarrow u_2A_2, \quad \dots \quad A_{k-2} \rightarrow u_{k-1}u_k$$
5. Replace any terminal symbols in $A \rightarrow u_iu_j$ with a new variable U_i and rule $U_i \rightarrow u_i$.

□

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- ▶ Regular languages have limitations
 - ▶ They “cannot count”
 - ▶ *Some* recursive structures however are still impossible . . .
- ▶ CFGs are defined by a set of (recursive) rules
 - ▶ They generate *context-free languages* (CFLs)
 - ▶ The class of regular languages is a subset of CFLs
- ▶ Every CFG can be converted to Chomsky normal form

Thanks!

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Lecture 6

Pushdown Automata

Last week the topic was *context-free languages* (CFL):

- ▶ CFL are more powerful than Regular languages (RL)
 - ▶ CFG is more *flexible* than regular grammars (cf. tutorial)
- ▶ CFG is defined by (recursive) *production rules*
 - ▶ Each rule transforms a variable to a string
 - ▶ String may consist of terminals and variables
- ▶ Examples to remember:
 - ▶ RLs: $\{x^*y^*\}$ and $\{x^n y^n \mid 0 \leq n \leq m\}$ (with m finite)
 - ▶ CFL: $\{x^n y^n \mid n \geq 0\}$

FA are **computational models** for regular languages

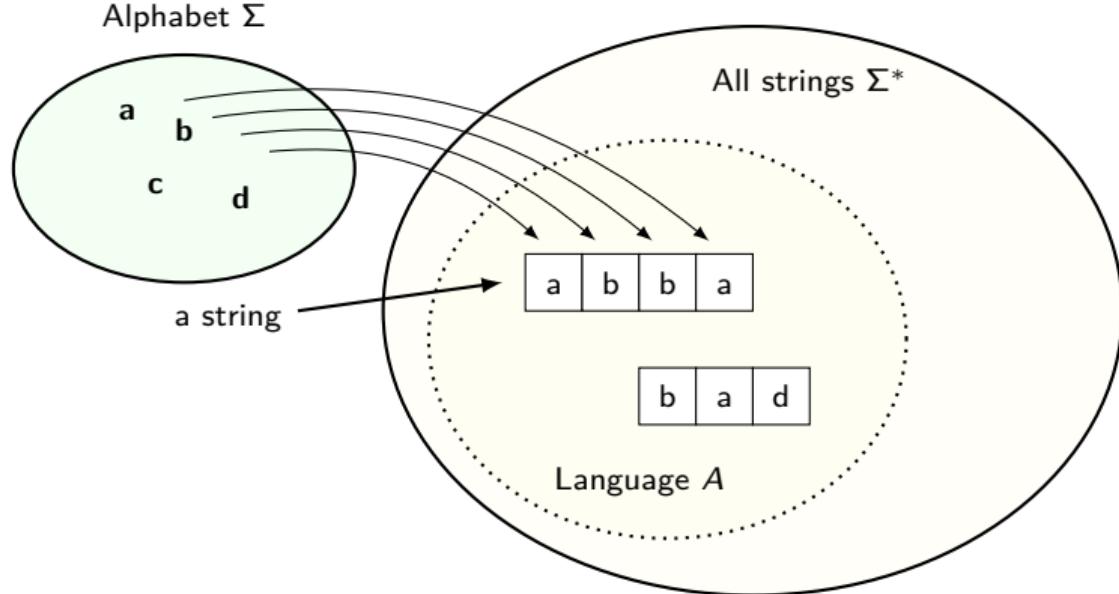
The computational model for CFL is **Pushdown Automata**

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Alphabets, Strings and Languages

TÖL301G

E. Hyttiä

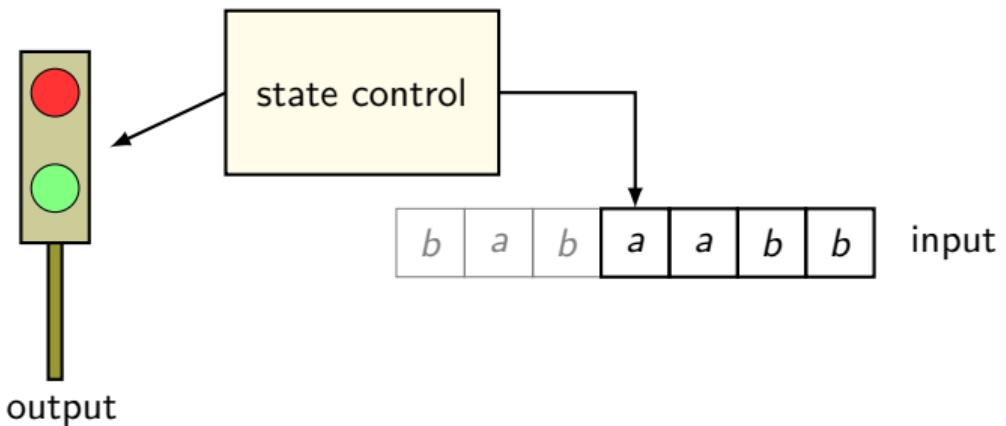


- ▶ Alphabet Σ : a finite set of symbols
- ▶ String w : a finite sequence of symbols, $w = a_1 a_2 \cdots a_n$
- ▶ Language A : set of strings (either finite or countably infinite)
 - ▶ *regular*: DFA, NFA and RE exist
 - ▶ *context-free*: CFG exists

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Schematic representation: Finite Automata



0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

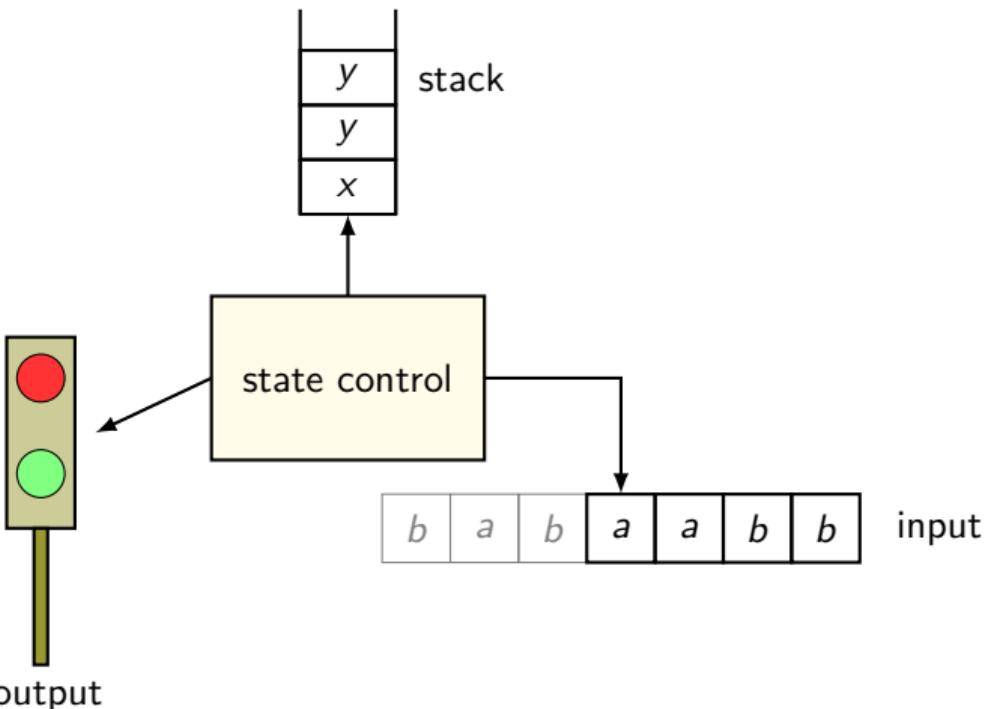
10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Schematic representation: Pushdown Automata



- ▶ Same elements as in Finite Automata
- ▶ and additionally a stack

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Transitions in the State Diagram

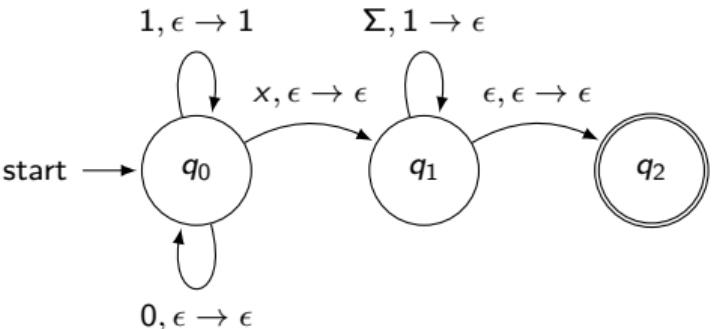


Fig. An example PDA.

Notation:

$$a, b \rightarrow c$$

Explanation	If ϵ
a input symbol read	do not read input
b symbol popped from the stack	do not read stack
c symbol pushed back to the stack	nothing

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Main properties of Pushdown Automata (PDA)

Stack:

- ▶ Stack has infinite capacity (this is model!)
- ▶ Stack has its own alphabet Γ
- ▶ Basic operations: push and pop (LIFO)

Configuration: (complete state, snapshot)

- ▶ Current state $q \in Q$ $(Q$ is finite)
- ▶ Symbols stored in the stack $(\text{infinite capacity})$

Computing capacity: equivalent to CFG

- ▶ Two options to show that a language is context free
 1. Construct an appropriate CFG
 2. Construct an appropriate PDA

Like FAs, PDAs can be

- | | | |
|---------------------|---|--|
| 1. Deterministic | } | Not equivalent in power

We need the nondeterminism! |
| 2. Nondeterministic | | |

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Formal definition of PDA

Definition 22 (Def. 2.13)

A pushdown automata is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

- ▶ Q is the finite set of states
- ▶ Σ is the finite input alphabet
- ▶ Γ is the finite stack alphabet
- ▶ $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$ is the transition function
- ▶ $q_0 \in Q$ is the start state
- ▶ $F \subseteq Q$ is the set of accept states

Note: Q , Σ , Γ and F are all *finite sets*!

When compared to NFA:

- ▶ PDA has the stack alphabet Γ
- ▶ Transition function involves Γ_ϵ both
 - ▶ at input (a symbol, if any, popped from the stack), and
 - ▶ at output (a symbol, if any, pushed to the stack)

Definition 23 (Computing with PDA)

PDA recognizes string w if it can be written as

$$w = w_1 w_2 \dots w_m, \quad w_i \in \Sigma_\epsilon$$

and there are

1. a sequence of states $r_0, \dots, r_m \in Q$, and
 2. a sequence of strings $s_0, s_1, \dots, s_m \in \Gamma^*$ $\Leftarrow \text{stack!}$

such that the following three conditions are satisfied:

- 1. Start:** $r_0 = q_0$ and $s_0 = \epsilon$ (an empty stack)

2. Movement:

$(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ for all $i = 0, \dots, m-1$ where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\epsilon$ and $t \in \Gamma^*$

- ### 3. Final state: $r_m \in F$

Note:

1. Input string w can be augmented with ϵ :s $(m \geq |w|)$
 2. Stack is (i) initially empty, and
(ii) adjusted at most by ± 1 symbol per step

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example PDA M_1

Example 2.14

A formal description for PDA that recognizes the language

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

is given by 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

1. $Q = \{q_0, q_1, q_2, q_3\}$,
2. $\Sigma = \{0, 1\}$,
3. $\Gamma = \{0, \$\}$,
4. $F = \{q_0, q_3\}$, and
5. δ is defined by table

Input: Stack:	0			1			ϵ		
	0	\$	ϵ	0	\$	ϵ	0	\$	ϵ
q_0									
q_1			$\{(q_1, 0)\}$	$\{(q_2, \epsilon)\}$					$\{(q_1, \$)\}$
q_2				$\{(q_2, \epsilon)\}$				$\{(q_3, \epsilon)\}$	
q_3									

where blank means \emptyset

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example PDA M_1 : state diagram

(Example 2.14 continues)

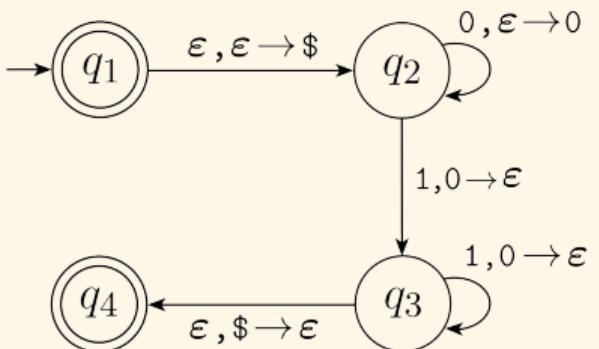


Figure: State diagram for M_1 recognizing $\{0^n 1^n \mid n \geq 0\}$

Example (Tutorial 5, recap)

Show that language

$$L = \{0^i 1^j 2^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } j = k\}$$

is nonregular.

Proof by contradiction using Pumping Lemma:

Suppose that L is regular and let $s = 0^p 1^p$, where p is the pumping length. As $s \in L$ and $|s| \geq p$, PL says that s divides into 3 parts, $s = xyz$, so that

1. $xy^i z \in L$ for all $i = 0, 1, \dots$
2. $|y| > 0$
3. $|xy| \leq p$

The last two conditions imply that y consists of one or more 0. Now $xy^0 z$ “removes” some 0:s from s , which still has no 2:s, and it thus does not belong to L . This contradicts the first condition, and therefore L must be nonregular. \square

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example (Tutorial 5, recap)

Show that language

$$L = \{0^i 1^j 2^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } j = k\}$$

is context free.

Proof:

CFG for $\{0^n 1^n 2^k\}$:

$$S_1 \rightarrow A_1 B_1$$

$$A_1 \rightarrow 0 A_1 1 \mid \epsilon$$

$$B_1 \rightarrow 2 B_1 \mid \epsilon$$

CFG for $\{0^n 1^k 2^k\}$:

$$S_2 \rightarrow A_2 B_2$$

$$A_2 \rightarrow 0 A_2 \mid \epsilon$$

$$B_2 \rightarrow 1 B_2 2 \mid \epsilon$$

Their union, corresponding to L , is obtained with a new start variable $\tilde{S} \rightarrow S_1 \mid S_2$. We have constructed a CFG for L , and therefore L is CFL. \square

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example PDA M_2

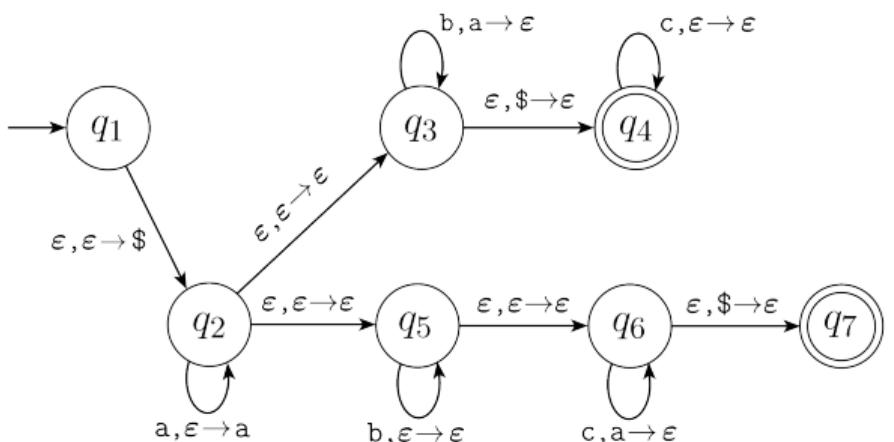
Example 2.16

Derive a PDA that recognizes the language

$$\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$$

Note that the condition is slightly different than in the previous example.

State diagram of PDA M_2 recognizing L is depicted below:



0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example PDA M_3

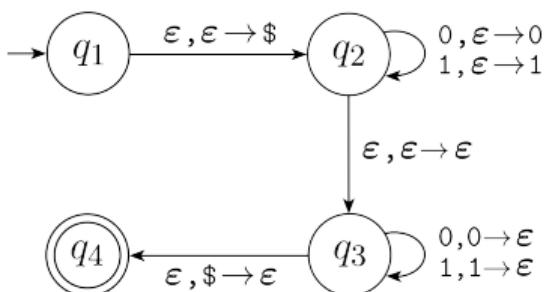
Example 2.18

Derive a PDA that recognizes the language

$$\{ww^R \mid w \in \{0,1\}^*\}$$

i.e., a language where second part is the same as the first part in reverse order,
e.g.,

first part reversed
 $\overbrace{101111} \quad \overbrace{111101}$



Status check:

- ▶ PDAs are clearly more powerful than FAs
 - ▶ The stack enables one form of *counting*
 - ▶ Like CFG, able to answer many questions that FAs cannot

Next step:

- ▶ Show that PDAs are equivalent with CFGs!
- ▶ Thus both characterize the class of *context-free languages*

Theorem 26 (2.20)

A *language* is context free iff some PDA recognizes it.

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Forward direction

Lemma 27 (2.21)

If a language is context free, then some PDA recognizes it.

Proof:

Every CFL has CFG $G = (V, \Sigma, R, S)$ by definition.

Construct the corresponding PDA that recognizes $L(G)$ as follows:

1. **Init:** push marker $\$$ and start variable on the stack
2. **Repeat forever:**
 - ▶ If top of stack is variable A :
 - ▶ **Nondeterministically**, select one rule for A and apply it
 - ▶ If top of stack is terminal symbol a :
 - ▶ Read next input symbol and compare it to a . If mismatch, reject this branch of the nondeterminism.
 - ▶ If top of stack is the marker $\$$, enter the accept state
 - ▶ (If string ends here, it is *accepted*)



Every possible string is checked “exhaustively” using the “massive parallelism” of nondeterminism!

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

State diagram of the PDA for CFG G

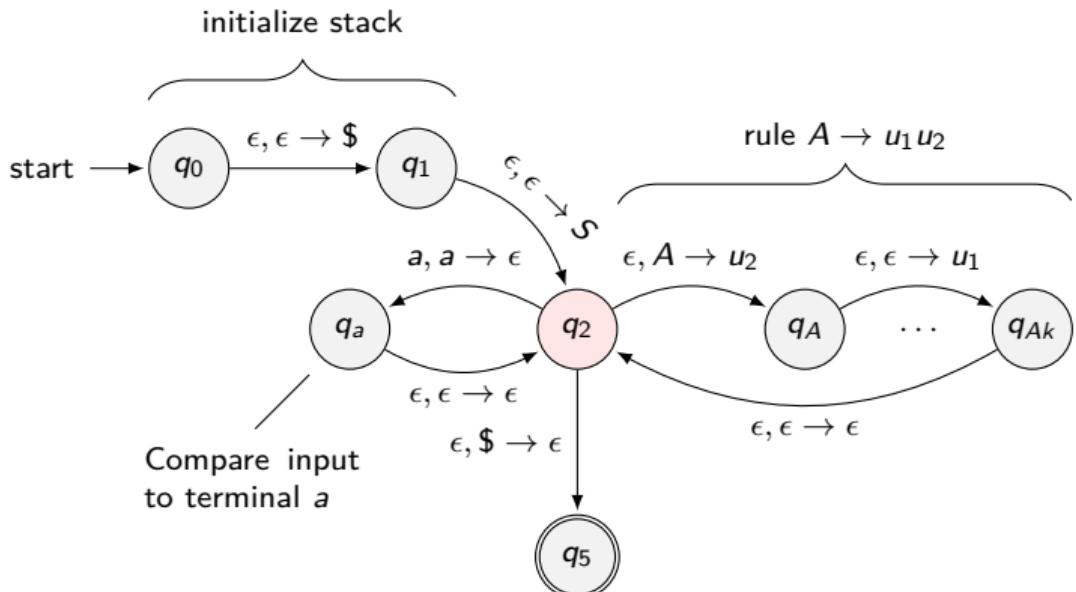


Figure: Constructing a PDA for a given CFG G .

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Some remarks

To define the matching PDA more formally:

- ▶ Input alphabet Σ is CFG's terminal symbols
- ▶ Stack alphabet Γ is CFG's variables and terminals
 - ▶ and the $\$$ marker
- ▶ Single accept state q_{accept} is sufficient
- ▶ Start state leads to two follow-up states that setup the stack to $\{S, \$\}$
- ▶ The “repeat” section:
 - ▶ Consists of “three” types of “action sequences”
 - ▶ Afterwards return back to start of the “repeat” (state)

Give a formal description for the PDA (cf. the book)

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Simplified PDAs

Lemma 28 (Simplified PDA)

Any PDA P can be modified into an equivalent PDA P' s.t.

1. P' has a single accept state q_{accept}
2. P' empties its stack before entering q_{accept}
3. Each transition in P' either
 - ▶ Pushes a symbol to stack (a push move), or
 - ▶ Pops a symbol from stack (a pop move)

Proof:

The first two conditions are easy alterations of PDA P (add states with ϵ input). For the third, we replace transitions with

1. **Both pop and push** by two transitions via a new state where the first executes the pop and the second the push operation
2. **Neither pop or push** by two transitions; the first pushes an arbitrary symbol to stack and the next one removes it

Carrying the above steps yields an equivalent simplified PDA. □

Lemma 29 (2.27)

If a PDA recognizes some language L , then L is context free.

Roadmap in Proof of Lemma 2.27:

1. Convert an arbitrary PDA to the simplified form
 - ▶ Done, see Lemma 28
2. Construct a CFG G based on P
 - ▶ Every CFG defines a CFL ...
3. Show that G and P define the same language
 - ▶ If $A_{pq} \xrightarrow{*} x$, then x takes P from p to q
 - ▶ If x takes P from p to q , then $A_{pq} \xrightarrow{*} x$
(everything “with an empty stack”)

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

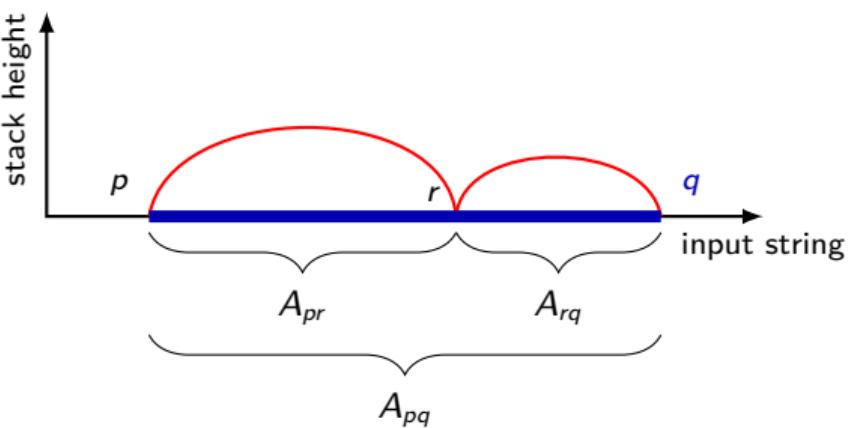
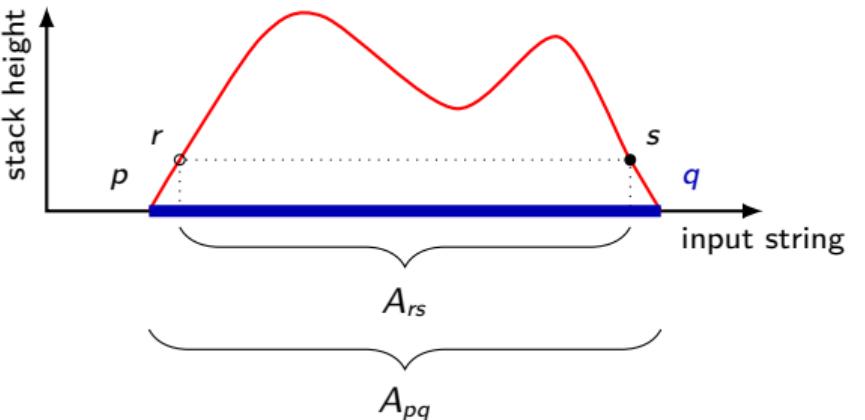
13 Review

Construction of the CFG (Idea)

Construct a CFG for a given (simplified) PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$:

1. For each pair of states p and q in Q , define variable A_{pq}
 - ▶ **Idea:** A_{pq} generates all strings that take P from p to q , with stack empty both at the start and end
 - ▶ These strings thus *preserve* the stack when P moves from p to q
 2. Now strings that (simplified) P accepts
 - ▶ Must start with a push (empty stack initially)
 - ▶ ... and end with a pop (must end with stack empty)
 3. Two possibilities for the last pop:
 - a) Last symbol was pushed at the start i.e., the stack was never empty inbetween
 - b) Stack was empty inbetween when PDA was in state r
- These yield two production rules:
- a) $A_{pq} \rightarrow aA_{rs}b$, where a, b are the inputs at start & end
 - b) $A_{pq} \rightarrow A_{pr}A_{rq}$

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review



Proof: Construct the CFG G (Step 2)

Suppose that the PDA is $P = \{Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\}\}$.

We construct a corresponding CFG G as follows:

- ▶ Variables of G are $\{A_{pq} \mid p, q \in Q\}$
- ▶ Start variable is $A_{q_0, q_{\text{accept}}}$
- ▶ Production rules are:
 1. For each $p, q, r, s \in Q$, $u \in \Gamma$ and $a, b \in \Sigma_\epsilon$:
Add rule $A_{pq} \rightarrow aA_{rs}b$ in G if
 - i) $\delta(p, a, \epsilon)$ contains (r, u) and
 - ii) $\delta(s, b, u)$ contains (q, ϵ)
 2. For each $p, q, r \in Q$: **add** rule $A_{pq} \rightarrow A_{pr}A_{rq}$ in G
 3. For each $p \in Q$: **add** rule $A_{pp} \rightarrow \epsilon$ in G

Notation:

w.e.s. is a short-hand notation for “with empty stack”

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Proof: Equivalency (Step 3a) I

Claim 2.30: If A_{pq} generates x , then x can bring P from p w.e.s. to q w.e.s.

Proof:

We prove this claim by induction on the number of steps in the derivation of x from A_{pq} .

Basis: *The derivation has 1 step*

A derivation with a single step must use a rule whose right-hand side contains no variables. The only rules in G where no variables occur on the right-hand side are $A_{pp} \rightarrow \epsilon$. Clearly, input ϵ takes P from p w.e.s. to p w.e.s., so the basis is proved.

Induction step: Assume true for derivations of length at most k , where $k \geq 1$.

Suppose that $A_{pq} \xrightarrow{*} x$ with $k + 1$ steps. The first step in this derivation is either

1. $A_{pq} \rightarrow aA_{rs}b$ or
2. $A_{pq} \rightarrow A_{pr}A_{rq}$.

We handle these two cases separately.

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Proof: Equivalency (Step 3a) II

(Assumption: A_{pq} derives x with $k + 1$ steps)

First case: First step is $A_{pq} \rightarrow aA_{rs}b$

Because $A_{rs} \xrightarrow{*} y$ with k steps, the induction hypothesis tells us that P can go from r on empty stack to s on empty stack.

Because $A_{pq} \rightarrow aA_{rs}b$ is a rule of G ,

- (a) $\delta(p, a, \epsilon)$ contains (r, u)
- (b) $\delta(s, b, u)$ contains (q, ϵ) ,

for some stack symbol u . Hence,

1. If P starts at p w.e.s., after reading a it can go to state r and push u onto the stack [cf. (a)]
2. Then reading string y can bring P to s and leave u on the stack [cf. induction]
3. Then after reading b it can go to state q and pop u off the stack [cf. (b)]

Therefore, x can bring P from p w.e.s. to q w.e.s.

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Proof: Equivalency (Step 3a) III

(Assumption: A_{pq} derives x with $k + 1$ steps)

Second case: First step is $A_{pq} \rightarrow A_{pr}A_{rq}$

Consider the portions y and z of x that A_{pr} and A_{rq} respectively generate, so $x = yz$.

Because $A_{pr} \xrightarrow{*} y$ in at most k steps and $A_{rq} \xrightarrow{*} z$ in at most k steps, the induction hypothesis tells us that y can bring P from p to r , and z can bring P from r to q , w.e.s: at the beginning and end. Hence x can bring P from p w.e.s. to q w.e.s.

This completes the induction step. □

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Proof: Equivalency (Step 3b) I

Claim 2.31: If x can bring P from p w.e.s. to q w.e.s., A_{pq} generates x .

Proof:

We prove this claim by induction on the number of steps in the computation of P that goes from p to q w.e.s:s on input x .

Basis: *The computation has 0 steps.*

If a computation has 0 steps, it starts and ends at the same state – say, p . So we must show that $A_{pp} \xrightarrow{*} x$. In 0 steps, P cannot read any characters, so $x = \epsilon$. By construction, G has the rule $A_{pp} \rightarrow \epsilon$, so the basis is proved.

Induction step: Suppose the claim is true for computations of length at most k , where $k \geq 0$.

Suppose that P has a computation wherein x brings it from p to q w.e.s:s in $k + 1$ steps. During this computation, either

1. Stack is empty only at the beginning and end, or
2. Stack becomes empty elsewhere, too.

Proof: Equivalency (Step 3b) II

(Assumption: x brings P from p to q w.e.s:s in $k + 1$ steps)

First case: “Stack empties only at the end”

The symbol that is pushed at the first move must be the same that is popped at the last move. Call this symbol u .

Let a be the input read in the first move, b be the input read in the last move, r be the state after the first move, and s be the state before the last move. Then

- ▶ $\delta(p, a, \epsilon)$ contains (r, u) and
- ▶ $\delta(s, b, u)$ contains (q, ϵ) ,

and so rule $A_{pq} \rightarrow aA_{rs}b$ is in G .

Let y be the portion of x without a and b , so $x = ayb$. Input y can bring P from r to s without touching the symbol u that is on the stack, and so P can go from r w.e.s. to s w.e.s. on input y .

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Proof: Equivalency (Step 3b) III

(Assumption: x brings P from p to q w.e.s:s in $k + 1$ steps)

We have removed the first and last steps of the $k + 1$ steps in the original computation on x so the computation on y has $(k + 1) - 2 = k - 1$ steps. Thus the induction hypothesis tells us that $A_{rs} \xrightarrow{*} y$, and therefore $A_{pq} \xrightarrow{*} x$.

Second case: “Stack empty meanwhile”

Let r be a state where the stack becomes empty other than at the beginning or end of the computation on x . Then the portions of the computation from p to r and from r to q each contain at most k steps. Say that y is the input read during the first portion and z is the input read during the second portion. The induction hypothesis tells us that $A_{pr} \xrightarrow{*} y$ and $A_{rq} \xrightarrow{*} z$.

Because rule $A_{pq} \rightarrow A_{pr}A_{rq}$ is in G , $A_{pq} \xrightarrow{*} x$, and the proof is complete. \square

That completes the proof of Lemma 2.27 and of Theorem 2.20. \square

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Corollary 30

Every regular language is context free.

Why?

Steps ...

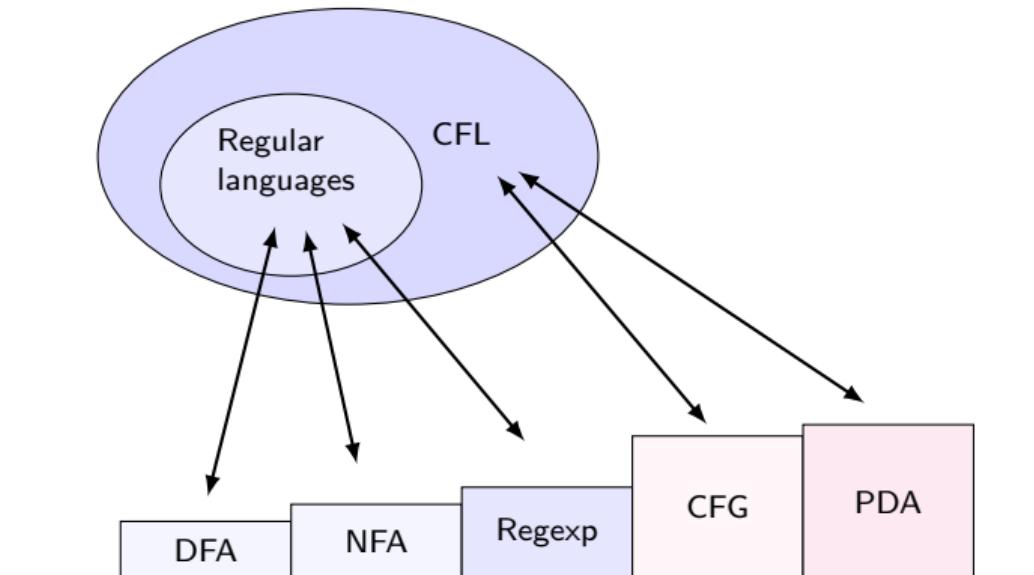
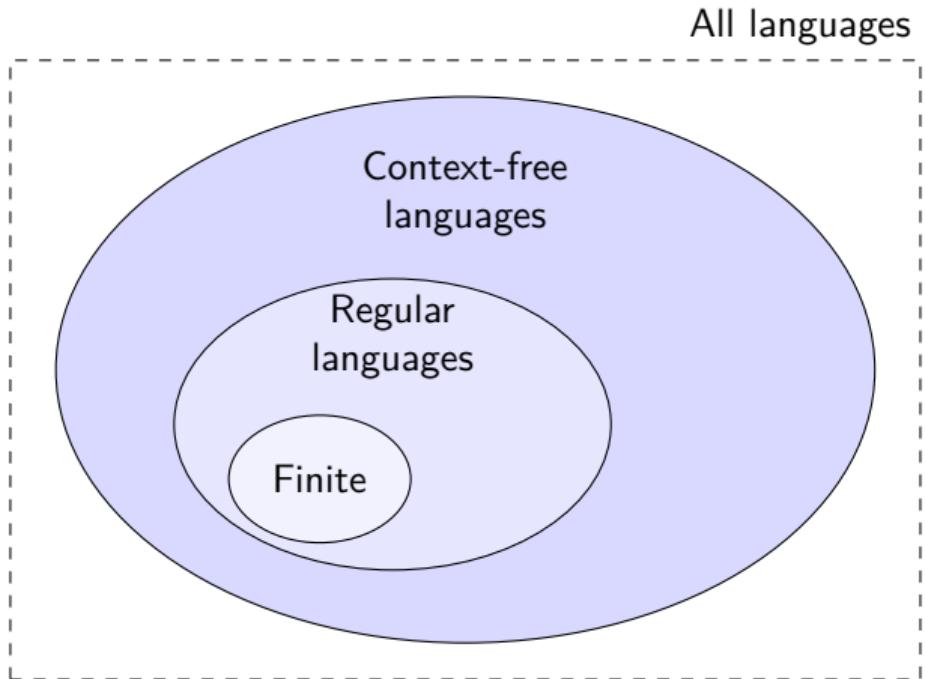


Figure: Stairway to ...

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review



- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Pumping Lemma for CFL

Theorem 31 (Thm. 2.34: Pumping lemma for CFL)

If A is a context-free language, then there is a number p (the pumping length) where, if s is any string in A of length at least p , then s may be divided into five pieces, $s = uvxyz$, satisfying the conditions

1. $uv^i xy^i z \in A$ for each $i \geq 0$
2. $|vy| > 0$
3. $|vxy| \leq p$.

Note:

- ▶ Condition 1: you can pump “up” and “down”
- ▶ Condition 2: both v and y cannot be empty strings
- ▶ Condition 3: the length of substrings vxy is at most p

Proof: see the book.

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Summary

- ▶ Pushdown Automata are like NFA with *a stack*
 - ▶ Stack has infinite size (model!)
 - ▶ At every step, PDA can pop and/or push a stack symbol
 - ▶ Stack alphabet can be different from the input alphabet
- ▶ We focus on nondeterministic PDAs
- ▶ **PDAs are equivalent to CFGs**
 - ▶ Both recognize/generate the same set of strings
- ▶ Alternative characterization for CFLs
 - ▶ CFG *generates* the set of (acceptable) strings (cf. REs)
 - ▶ PDA *checks* if a given string is accepted or not (cf. NFA)
- ▶ PDAs are generalization of NFAs
 - ▶ Simply “refuse” to use the stack
 - ▶ Thus, any regular language is also context free

Thanks!

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Lecture 7

Turing Machines

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Recap of Previous Lectures

We are familiar with

1. Deterministic Finite Automata (DFA) and Nondeterministic Finite Automata (NFA)
 - ▶ Recognize the class of *regular languages*
 - ▶ “(very) small devices with limited memory”
2. Pushdown Automata (PDA)
 - ▶ Recognize the class of *context-free languages*
 - ▶ “Small devices with unlimited stack”

However, all above computational models are severely limited (cf. examples with pumping lemma)

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Turing Machines (TM)

- ▶ Proposed by Alan Turing in 1936³
- ▶ Similar to Finite Automata, but
 - ▶ a tape provides “unrestricted memory”
- ▶ More general model than DFA, NFA and PDA
- ▶ Useful model for general purpose computers
 - ▶ Can do anything your laptop can do
 - ▶ ... still some problems exist that are beyond TMs



Figure: Alan Turing (from <http://www.turingarchive.org>)

³A. M. Turing. “On Computable Numbers, with an Application to the Entscheidungsproblem”. In: *Proceedings of the London Mathematical Society* 2.42 (1936), 230–265.

- ▶ “Pilgrimage” destination for CS enthusiasts in UK
- ▶ Half-way between Oxford and Cambridge
- ▶ <https://bletchleypark.org.uk/>
- ▶ Codebreakers, Enigma machines, Colossus, Alan Turing, The National Museum of Computing ...



Figure: Mansion at Bletchley Park, Milton Keynes (From Wikimedia/Magnus Manske).

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

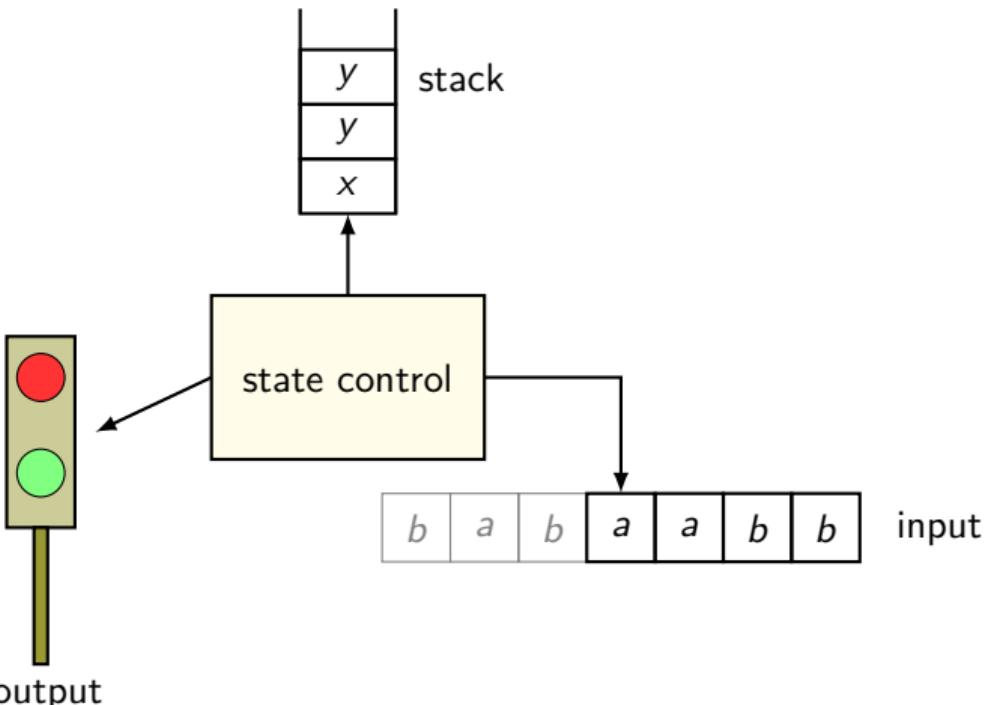
Recently in the News



- 0 Introduction
 - 1 DFA
 - 2 NFA
 - 3 Regexps
 - 4 Nonregular L
 - 5 Context-Free
 - 6 Pushdown
 - 7 Turing
 - 8 Decidability
 - 9 Reducibility
 - 10 Time
 - 11 Complete
 - 12 Brute Force,
SPACE and
Probabilities
 - 13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

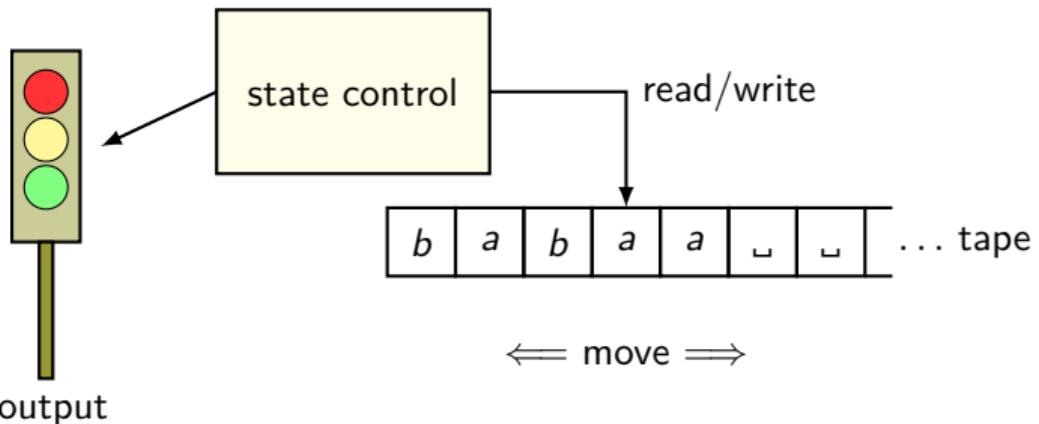
Schematic representation: Pushdown Automata



- ▶ Same elements as in Finite Automata
- ▶ Additionally a stack (pop/push)

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Schematic representation: Turing Machine



- ▶ Same elements as in Finite Automata
- ▶ Instead of an input sequence, a tape w/ read/write/move
- ▶ Special accept and reject states where computation halts



- ▶ *Infinite tape* as the “unrestricted memory”
- ▶ Initialized with the *input string*
 - ▶ followed by blanks
- ▶ **Read** and **Write** where the tape head is
- ▶ Tape head can be moved **left** and **right**
- ▶ Computation ends when a machine enters a **halting state**:
 - ▶ **Accept** indicates a positive outcome
 - ▶ **Reject** indicates a negative outcome

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

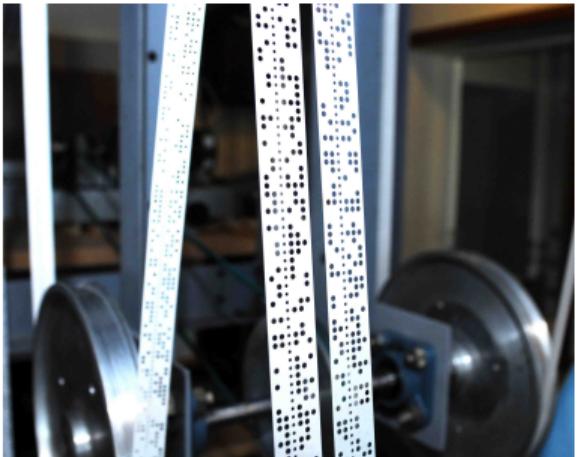


Figure: Paper tapes used in Heath Robinson at Bletchley Park (1943) (from
<http://www.tnmoc.org/special-projects/robinson>)

No wonder Turing machines use tapes!

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Example 32 (Turing Machine M_1)

Consider language $\{w\#w \mid w \in \{0, 1\}^*\}$, i.e., the task is compare if two files, concatenated together, match.^a Which computational models are up to this task?

- ▶ DFA/NFA has a finite memory ...
- ▶ PDA has infinite memory in its stack ...
- ▶ TM has infinite memory in its tape ...

^aSee `cmp` command on Linux and alike.

Steps ...

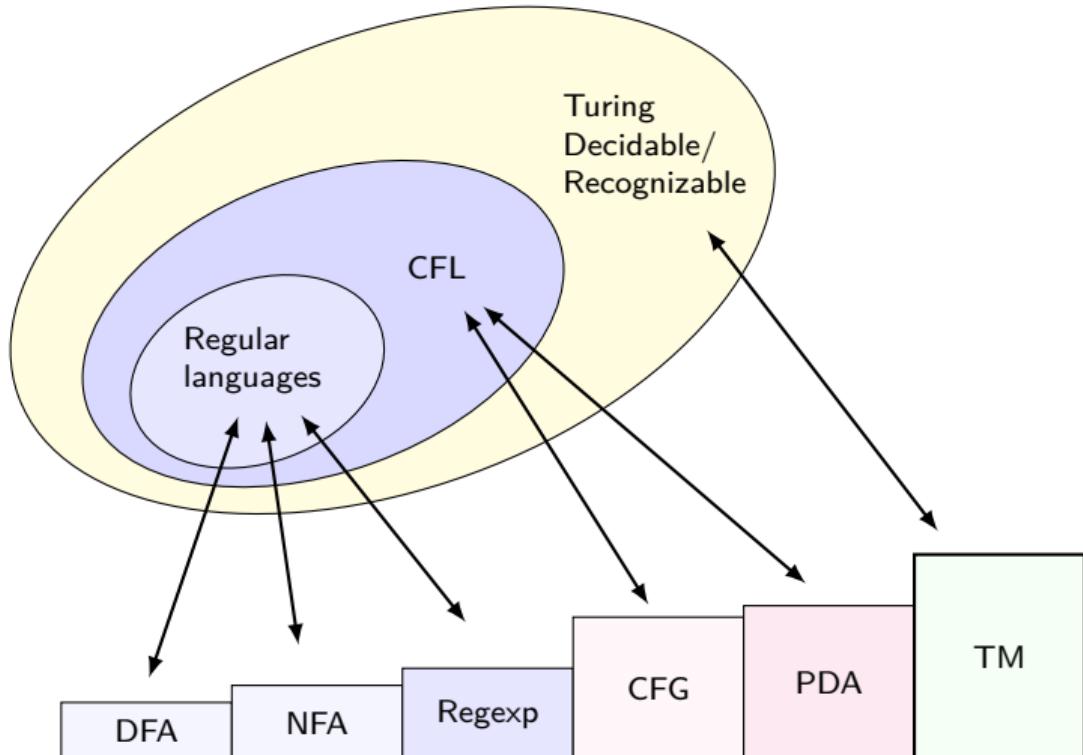


Figure: Stairway to ...

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

Formal definition of TM

Definition 24 (Turing machine TM (Def. 3.3))

Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$:

- ▶ *Q is the finite set of states*
- ▶ *Σ is the finite input alphabet, not containing the blank symbol \sqcup*
- ▶ *Γ is the finite tape alphabet, $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$*
- ▶ *$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function*
- ▶ *$q_0, q_{accept}, q_{reject} \in Q$ are the start, accept and reject states, $q_{accept} \neq q_{reject}$*

Note:

- ▶ Q, Σ, Γ are all *finite sets*
- ▶ Transition function includes no power set,
i.e., (this) TM is **deterministic**

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

About Turing Machines

- ▶ System's **configuration** (complete state) comprises
 1. Current state $q \in Q$
 2. Contents of the tape T
 3. Position r of the tape head
- ▶ Transition function δ operates on these:
 - ▶ Input from $Q \times \Gamma$ is
 1. Current state q
 2. Symbol on tape at r
 - ▶ Action from $Q \times \Gamma \times \{L, R\}$ is
 1. New state $q' \in Q$
 2. Symbol written on tape at r
 3. Whether the tape head is moved *left* or *right*
- ▶ Compute until a halting state is reached: q_{accept} or q_{reject}
(or forever . . .)

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Notation for Configurations

Configuration C describes

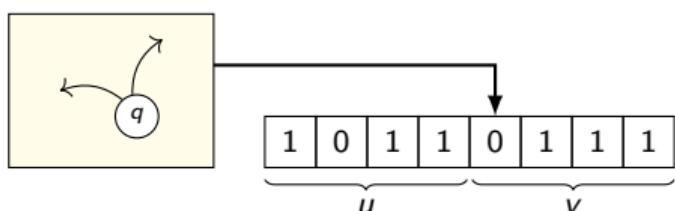
1. Current state q
2. String on tape
3. Position of the tape head

These can be encoded into a single string:

$$uqv = \{u, q, v\}, \quad u, v \in \Gamma^*, \quad q \in Q$$

where

- ▶ string u describes what is on **left of the tape head**
- ▶ $q \in Q$ is the current state (and delimiter for two strings)
- ▶ string v describes what is on **right of the tape head**
including the symbol currently under the tape head



Definition 25

Configuration C_1 yields C_2 if TM can move from C_1 to C_2 in a single step.

- ▶ Let $a, b, c \in \Gamma$ and $u, v \in \Gamma^*$, and $q_i, q_j \in Q$. Then

$uaq_i bv$ yields $uq_j acv$ if $\delta(q_i, b) = (q_j, c, L)$
 $uaq_i bv$ yields $uacq_j v$ if $\delta(q_i, b) = (q_j, c, R)$

Special cases:

Left: $q_i bv$ yields $q_j cv$ if "L" (i.e., ignored)
Right: uaq_i yields $uacq_j$ if "R" (cf. blanks)

- ▶ **Start configuration:** $q_0 w$, where w is the input string
- ▶ **In halting configurations** the computation has stopped:

Accepting configuration: state is q_{accept}

Rejecting configuration: state is q_{reject}

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

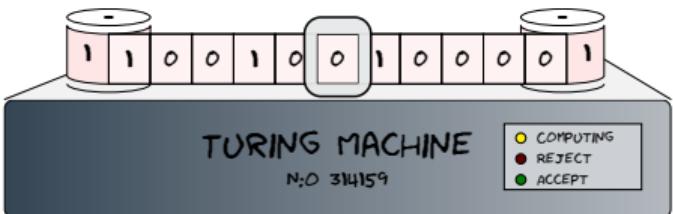
10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

TM simulator



We will use TM simulators, two options:

<https://hi.is/~esa/tol301g/tm2.html>

and

<https://turingmachinesimulator.com/>

Note: Autograding (simulator) uses the same syntax

With the TM simulator(s):

- ▶ Tape is infinite to both left and right
- ▶ Tape-head movement with $<$ and $>$, or no movement with $-$
- ▶ (Multi-tape machines are also supported by the latter)

Equivalent to Formal Description

Turing machine Simulator

Syntax for <https://turingmachinesimulator.com/>

- ▶ Define (i) name, (ii) initial state, and (iii) accept state

- ▶ Define transitions:

[state], [read] current state & symbol read
[new state], [write], [move] response/action

- ▶ Movement with <, >, - and the blank symbol is _

```
1 //----- CONFIGURATION -----
2 name: 2x + 1
3 init: q0
4 accept: qAccept
5 //----- TRANSITIONS -----
6 q0,0
7 q0,0,>
8 q0,1
9 q0,1,>
10 q0,_
11 qAccept,1,-
```

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example: add by 1

Implement a TM that increments a binary number by one?

```
1 //-----CONFIGURATION
2 name:      Increment Tape by One (LSB first)
3 init:      q0
4 accept:    qAccept
5
6 // Sit in state q0 while Carry is set
7 q0,0
8 q1,1,<
9 q0,-
10 q1,1,<
11 q0,1
12 q0,0,>
13
14 // Done, rewind the tape and halt
15 q1,0
16 q1,0,<
17 q1,1
18 q1,1,<
19 q1,-
20 qAccept,-,>
```

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

“Sum of three cubes” – food for your TM

Sum of three cubes problem

A diophantine equation

$$x^3 + y^3 + z^3 = k \quad (2)$$

It is known that for any integer n

$$n^3 \bmod 9 = 0, 1 \text{ or } 8 \text{ (i.e., } -1\text{)}$$

It follows that no solution exists to (2) if

$$k \bmod 9 = 4 \text{ or } 5$$

What about all k such that $k \bmod 9 \notin \{4, 5\}\text{?}$

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Brief history of the “sum of three cubes” problem

TÖL301G

E. Hyttiä

-
- 1954 Formulated at the University of Cambridge:
“*Find integer solutions to $x^3 + y^3 + z^3 = k$ for $k = 1, \dots, 100.$* ”
-

40 years passes ...

-
- 1992 Mathematician Roger Heath-Brown conjectures that all integers k except those with $k \bmod 9 \in \{4, 5\}$ can be expressed as a sum of three cubes
-

- 2000 Mathematician Noam Elkies publishes a practical algorithm to find solutions
-

- 2015 Mathematician Timothy Browning makes a video to *Numberphile* youtube-channel about the problem. At this point, solutions are known for all $k < 100$ except for 33, 42 and 74.
-

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force, SPACE and Probabilities

13 Review

The “sum of three cubes” problem (2)

-
- Apr 2016 Phycisist Sander Huisman gets interested in the problem after seeing the video and finds a solution for $k = 74$ (<https://arxiv.org/abs/1604.07746>)

$$\begin{aligned} & (-284650292555885)^3 \\ & + 66229832190556^3 \\ & + 283450105697727^3 = 74 \end{aligned}$$

Browning publishes a new video in youtube and mathematician Andrew Booker (Univ. of Bristol) gets interested in the problem

-
- Feb 2019 After weeks on a university supercomputer, Booker finds a solution for $k = 33$

$$\begin{aligned} & 8866128975287528^3 \\ & + (-8778405442862239)^3 \\ & + (-2736111468807040)^3 = 33 \end{aligned}$$

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force, SPACE and Probabilities

13 Review

With only $k = 42$ left, Booker realizes that it is beyond the power of the supercomputer available to him, and starts to collaborate with Andrew V. Sutherland with (MIT)

Sep 2019 By using a network of ≈ 500000 home computers (Charity Engine), the team finds a solution

$$\begin{aligned} & (-80538738812075974)^3 \\ & + 80435758145817515^3 \\ & + 12602123297335631^3 = 42 \end{aligned}$$

The unsolved cases below 1000 are now 114, 165, 390, 579, 627, 633, 732, 906, 921, and 975

Time to put your Turing machine to work!

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexp
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

The “sum of three cubes” problem (4)

The solutions can be rare

Already in 1953, Louis J. Mordell gives two solutions for $k = 3$

$$\begin{aligned} & 1^3 + 1^3 + 1^3 \\ &= 4^3 + 4^3 + (-5)^3 \\ &= 3 \end{aligned}$$

Only in September 2019, Booker and Sutherland find the third(!) solution for it

$$\begin{aligned} & 569936821221962380720^3 \\ &+ (-569936821113563493509)^3 \\ &+ (-472715493453327032)^3 = 3 \end{aligned}$$

More food for your Turing machines!

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Recognizable vs. Decidable

Definition 26

The set of strings TM M accepts is the language of M , or the language recognized by M , denoted by $L(M)$.

Definition 27 (Def. 3.5)

Language L is **Turing-recognizable** if Turing machine M exists that recognizes L .

- ▶ Three possible outcomes: accept, reject and loop
- ▶ Loop is unfavorable “in practice”
- ▶ TMs that **never** loop are called **deciders**
- ▶ Decider **decides** on a language

Definition 28 (Def. 3.6)

A language is **Turing-decidable** (or **decidable**) if a Turing Machine exists that decides it.

Example 33

We have

1. Regular languages
2. Context-free languages
3. Turing-recognizable languages
4. Turing-decidable languages

What are their relationships?

- which is subset of which?

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example 34 (Turing machine M_2)

Let $A = \{0^{2^n} \mid n \geq 0\}$, i.e., all zero strings whose length is a power of 2. The task is to describe a Turing machine that *decides* on A .

Implementation description for M_2

Input string w

while true **do**

Sweep left to right across the tape, cross every other 0

if a single zero **then**

return accept

else if odd number of zeroes **then**

return reject

end if

end while

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

State diagram of M_2

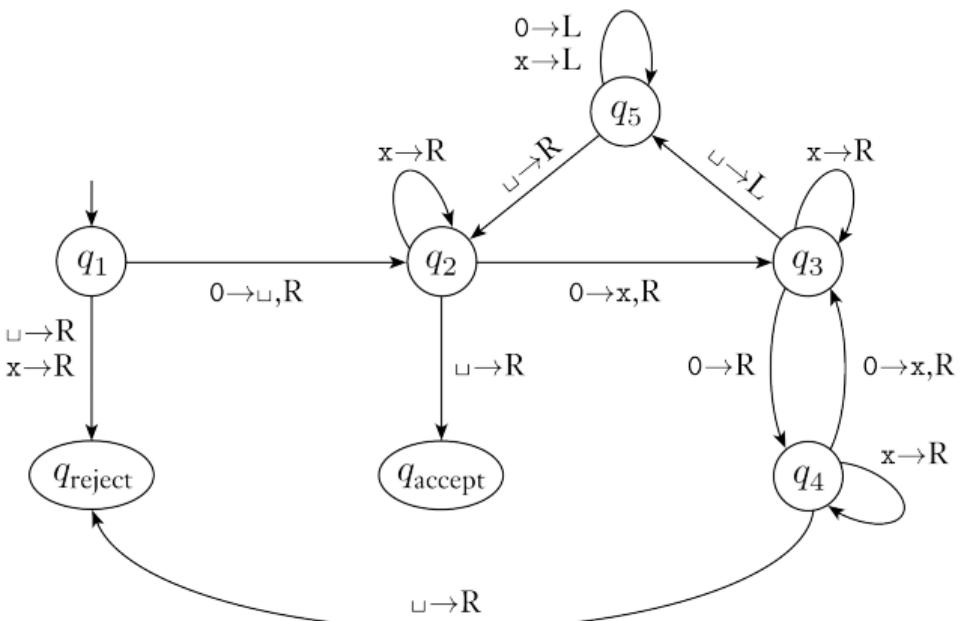


Figure: State diagram for Turing machine M_2 (fig. 3.8)

- ▶ Our Turing machine
 1. a single tape
 2. deterministic operation
- ▶ Would more features make it more powerful?
- ▶ If so, what would be useful additions?

SHOPPING LIST

- Eggs*
- Two stacks*
- One queue*
- Non-determinism*
- ...

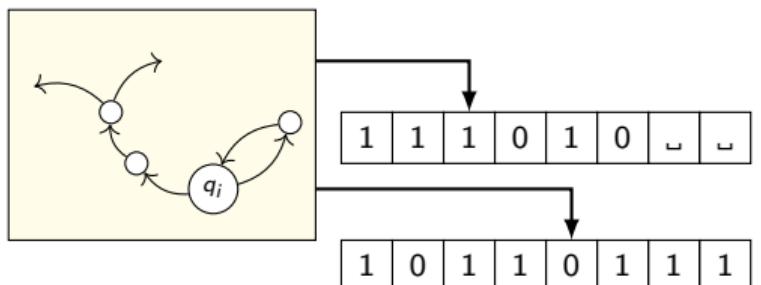
0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Multitape Turing Machines

Definition 29

A *multitape Turing machine* is like an ordinary Turing machine, but has $k > 1$ tapes.

- ▶ Initially the input is on tape 1, other tapes are blank
- ▶ Transition function can operate on any combination of tapes
- ▶ Movement per tape from (L,R,S) (stop) instead of (L,R)



- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexp
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

Theorem 35 (Thm. 3.13)

Every multitape Turing machine has an equivalent single-tape Turing machine.

Proof:

(Sketch): Show how a single-tape TM can simulate a multitape one. □

As every single-tape Turing machine is a special case of a multitape Turing machine:

Corollary 36 (Corollary 3.15)

A language is Turing-recognizable iff some multitape Turing machine recognizes it.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Nondeterministic variant works as expected:

At any point of computation, the TM may proceed according to several possibilities.

That is, the transition function has form

$$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

Computation realizes as a tree instead of a sequence.

If any branch ends with the accept state, the input string is accepted

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Nondeterministic Turing Machines (2)

Theorem 37 (Thm. 3.16)

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof:

(sketch) Again, simulate a nondeterministic variant with an ordinary TM. Linear sequence of configurations becomes a tree, of which every branch must be searched. Breadth-first search avoids a problem with never ending branches. \square

Corollary 38 (Corollary 3.18)

A language is Turing-recognizable iff some nondeterministic Turing machine recognizes it.

Hence, the deterministic single-tape Turing machine is as powerful as the “more general” variants.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

A natural generalization for decidability:

Definition 30

A nondeterministic Turing machine is a decider if all branches halt on all inputs.

Corollary 39 (Corollary 3.19)

A language is Turing-decidable iff some nondeterministic Turing machine decides it.

- ▶ Turing machines can be extended by adding “a printer”
- ▶ Printer is used to print strings
- ▶ Result is a list/sequence of strings
- ▶ These are called **enumerators**
- ▶ List of strings forms a language

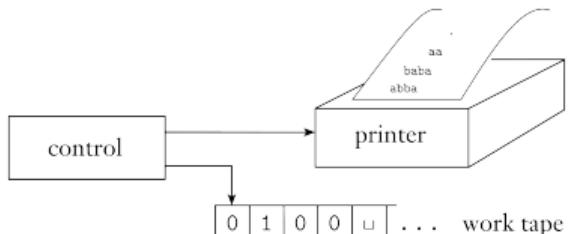


Figure: Enumerator (Fig. 3.20)

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Theorem 40 (Thm. 3.21)

A language is Turing-recognizable iff an enumerator exists that enumerates the language.

Proof:

Let E denote an enumerator and T a Turing machine.

1. From E to M :

Informally, on input w , TM M runs E that produces strings. Compare every output with w , and accept if a match.

2. From M to E :

Let s_1, s_2, \dots denote a list of all possible strings in Σ^* .

For $i = 1, 2, \dots$

2.1 Run M for i steps on each s_1, \dots, s_i

2.2 If computation finishes with an accept, print s_i out



Examples of algorithms:

- ▶ At elementary school:
 - ▶ Long multiplication and division
- ▶ Graph theory
 - ▶ Shortest path algorithms (e.g., Dijkstra, 1956)
 - ▶ Node coloring algorithms (e.g. greedy)
- ▶ Numerical analysis
 - ▶ Newton's algorithm (1669) for finding roots $y(x) = 0$
 - ▶ Euler's and Runge-Kutta methods for differential equations $y'(t) = f(t, y(t))$
- ▶ Signal processing
 - ▶ FFT (Cooley & Tukey, 1965) to compute DFT, $X_k = \sum_{j=0}^{N-1} x_j e^{i2\pi kj/N}$

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

23 problems proposed by David Hilbert in 1900:

9. Find the most general law of the reciprocity theorem in ...

10. A procedure to determine if a polynomial has integral root

11. Solving quadratic forms with algebraic numerical coefficients

The 10th problem is about algorithms!

Hilbert's 10th problem

Given a polynomial

$$p(x, y, \dots) = a_0 + a_1x + b_1y + a_2x^2 + b_2y^2 + c_1xy + \dots$$

with integer coefficients a_i, b_i, c_i, \dots , and variables x, y, \dots , is there an algorithm that determines if an integral root exists?

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

- ▶ Back in 1900, theoretical means inadequate
 - ▶ Possible to find an algorithm for some tasks
 - ▶ But no means to argue if a desired algorithm exists
- ▶ Notion of algorithm formalized in 1936
 - ▶ Church, “ λ -calculus
 - ▶ Turing, “machines”
- ▶ Equivalent \Rightarrow **Church-Turing thesis**



- ▶ Yuri Matijasevič (1970): no such algorithm exists!
Hilbert's 10th problem is algorithmically unsolvable

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Solving the 10th problem

TÖL301G

E. Hyttiä

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Define

$$D \triangleq \{p(x, y, \dots) \mid p \text{ has integral root}\}$$

The 10th problem: “**Is D decidable?**”

For single variable polynomials, define

$$D_1 \triangleq \{p(x) \mid p(x) \text{ has an integral root}\}$$

Clearly, $D_1 \subset D$

10th problem: recognizers

Define TM M_1 as follows:

```
for  $x = 0, 1, 2, \dots$  do
    if  $p(x) = 0$  or  $p(-x) = 0$  then
        return accept
    end if
end for
```

M_1 recognizes if p has integral root

$\Rightarrow D_1$ is **Turing-recognizable!**

- ▶ M_1 easy to generalize to a multivariable case $\Rightarrow M$
- ▶ However, both M_1 and M are recognizers, not deciders

Note: Above is a **high-level description** of an algorithm.

We do not describe how the tape is utilized.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

10th problem: Deciders

Theorem 41

The roots of a polynomial with a single variable,

$$p(x) = c_1 x^{k-1} + c_2 x^{k-2} + \dots + c_k$$

lie within

$$\pm k \frac{c_{\max}}{c_1}, \quad c_{\max} = \max |c_i|$$

Proof:

(Problem 3.10)

□

Implications:

1. With bounds, M_1 can be modified **to reject** p
 - ▶ D_1 is Turing-decidable!
2. Matijasevič theorem:
such bounds for multivariable case cannot be calculated

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Algorithms – level of detail

"TM is a model for algorithms"

1. Formal description:

- ▶ 7-tuple (states, transition function, alphabets ...)
- ▶ Sometimes a state diagram with transitions is also acceptable
 - ▶ See *an example*
- ▶ In our case, source code for the simulator
 - ▶ 7-tuple gets implicitly defined

2. Implementation description:

- ▶ English prose to describe how TM operates with the tape ("scan right until ... " etc.)
 - ▶ See *an example*

3. High-level description:

- ▶ English prose to describe an algorithm
- ▶ No need to discuss how TM operates with its tape

- ▶ Encoding object x to string: $\langle x \rangle$
 - ▶ Polynomials with integer coefficients?
 - ▶ Integer numbers
 - ▶ DFA, NFA, RE
 - ▶ Graphs
- ▶ Several objects: $\langle x_1, \dots, x_k \rangle$
- ▶ Indentation as always (cf. pseudo code)
 - ▶ Improves clarity

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Example 3.23

Let A denote the class of connected graphs:

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

A is decidable, i.e., a TM exists that *decides* A

1. High-level description is relatively easy:

- ▶ “Brute-force”
- ▶ Faster w/ list of boundary nodes
- ▶ cf. FloydWarshall algorithm

} Few lines

2. Implementation-level details includes

- ▶ Encoding of graphs to strings
- ▶ General operation (multiple stages)
- ▶ How everything happens on tape

} Long!

3. Formal description:

- ▶ Specify states, alphabets and transitions
- ▶ Transitions define how the tape is operated

} HUGE!

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Background information: Rice coding

Let x be an n -bit integer number, $x = (b_0 b_1 \dots b_{n-1})$

If x is typically small, the MSBs are often zero (inefficient)

Let us consider the Rice coding with parameter $k > 0$

$$\overbrace{b_0 b_1 \dots b_{k-1}}^{k \text{ LSBs}} \overbrace{b_k \dots b_{n-1}}^{\text{MSBs}} \Rightarrow b_0 b_1 \dots b_{k-1} \overbrace{11 \dots 1}^{m \text{ times}} 0$$

where $m = \lfloor x/2^k \rfloor$. The sequence of 1s is terminated by 0. (Instead of transmitting n bits, we transmit $k + \mathbb{E}[M] + 1$ bits)

Decoding x from $\{b_0, \dots, b_{k-1}, m\}$ works the same way

$$x = (b_0 b_1 \dots b_{k-1}) + m \cdot 2^k.$$

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Summary

Turing machines are like FA with **a tape**:

- ▶ “Unrestricted” memory access
(unlike a stack)
- ▶ Move Left/Right & Read/Write
- ▶ Halt states: accept and reject
... or loop forever
- ▶ TMs come in different forms:
 - ▶ Yet they are equivalent in power
- ▶ Language A is
 - ▶ Turing *recognizable* if a TM recognizing A exists
 - ▶ Turing *decidable* if the computation always ends
- ▶ TM is a model for algorithms – whether a problem is decidable or not



Thanks!

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Lecture 8

Decidability

- ▶ **Turing machines (TM):**
 - ▶ Tape for reading and writing
 - ▶ Position can be controlled (left/right)
- ▶ More powerful model than PDA
 - ▶ Stack of PDA can be simulated with the tape
- ▶ Accept and reject are *halting states*
- ▶ Two classes of languages:
 1. Turing-recognizable (TM exists)
 2. Turing-decidable (no loops)
- ▶ Church-Turing thesis:
 1. Informal definition of algorithm/procedure
 2. Turing machine
 3. (λ -calculus)

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

4.1 Decidable Problems

TÖL301G

E. Hyttiä

Suppose we have computational models X and Y .

We can ask

1. Whether X accepts a string w or not
2. Whether X accepts any string, or not
3. Whether X and Y recognize the same language, or not

Next we consider the above in the context of

1. Regular languages
2. Context-free languages
3. Turing machines

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Acceptance problem

“Does DFA B accept string w ”

This problem can be expressed as a *language*

Definition 31 (Acceptance)

$$A_{\text{DFA}} \triangleq \{\langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w\}$$

- ▶ Elements in A_{DFA} are pairs of
 - ▶ DFA and
 - ▶ matching string
- ▶ $\langle B, w \rangle$ belongs to A_{DFA} iff DFA B recognizes w

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 42 (Theorem 4.1)

A_{DFA} is a decidable language.

Proof:

We construct TM M that decides A_{DFA} :

$M = \langle B, w \rangle$, DFA B and string w :

1. (Note that B can be represented as a string)
2. First M verifies that the input is valid
3. Then M simulates B with input w
 - ▶ DFA can be simulated using a TM
4. Simulation takes a finite number of steps, $|w|$
5. The final state of B yields the desired result for M



13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 43 (Theorem 4.2)

A_{NFA} is a decidable language.

Theorem 44 (Theorem 4.3)

A_{RE} is a decidable language.

Proof:

Any NFA and RE can be first converted to DFA, and then *Theorem 4.1* applied. \square

Problem:

“Does DFA accept any strings?”

Also this problem can be expressed as a *language*

Definition 32 (Empty language)

$$E_{DFA} \triangleq \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Theorem 45 (Theorem 4.4)

E_{DFA} is a decidable language.

That is, we can devise an algorithm (a Turing Machine) that determines if a DFA accepts no strings!

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Proof:

Depending on if a DFA can move from state i to state j in one step, with a suitable input, or not, induces a directed graph G with vertices Q and edges according to δ . DFA accepts a string if a path from start state q_0 to an accept state in F exists in G . The states reachable from q_0 can be found iteratively as follows:

Mark q_0

repeat

 Mark states reachable from an already marked state

until no new states marked

if any accept state has been marked **then**

return Reject

else

return Accept

end if

The above algorithm (which corresponds to a TM) takes at most $|Q| - 1$ rounds, after which either an accept or reject is returned, and therefore E_{DFA} is decidable. \square

Two DFAs A and B are *equivalent* if they recognize the same language.

Question:

“Is this also a decidable problem?”

Definition 33 (Equivalency)

$$EQ_{DFA} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

In other words, is language EQ_{DFA} decidable?

Theorem 46 (Theorem 4.5)

EQ_{DFA} is a decidable language.

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Proof:

Idea: construct a new DFA C that accepts only strings that are accepted by exactly one of A or B . If $L(A) = L(B)$ then $L(C) = \emptyset$.

Hence, $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$ (symm. diff.)

$F = F(\langle A, B \rangle)$:

1. Construct DFA C as in above
 2. Run TM T from Theorem 4.4 on input $\langle C \rangle$
 3. If T accepts, accept, otherwise reject
-

TM F accepts if $L(C) = \emptyset$, i.e., when $L(A) = L(B)$, and otherwise rejects.

□

Acceptance

$$A_{\text{DFA}} = \{\langle B, w \rangle \mid B \text{ is a DFA that accepts string } w\}$$

Empty Language

$$E_{\text{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Equivalency

$$EQ_{\text{DFA}} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

All three are *decidable languages*.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

The same languages can be defined for CFGs:

Definition 34 (Decidability of CFGs)

Acceptance $A_{CFG} = \{\langle G, w \rangle \mid G \text{ is a } CFG \text{ that generates string } w\}$

Empty Language $E_{CFG} = \{\langle G \rangle \mid G \text{ is a } CFG \text{ and } L(G) = \emptyset\}$

Equivalency $EQ_{CFG} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are } CFGs \text{ and } L(G) = L(H)\}$

Are all above *decidable languages* again?

Theorem 47 (Theorem 4.7)

A_{CFG} is decidable.

Proof:

Construct a TM S as follows:

$S = S(\langle G, w \rangle)$, CFG G and string w :

1. Convert G to Chomsky form $\Rightarrow G'$
 - ▶ Any derivation of w has $2n - 1$ steps,⁴ where $n = |w|$
2. List all derivations for G' with $2n - 1$ steps
(with 1 step if $n = 0$)
3. If any derivation generates w , accept, otherwise reject



13 Review

12 Brute Force,
SPACE and
Probabilities

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

⁴See Problem 2.26 from the book.

Theorem 48 (Theorem 4.8)

E_{CFG} is decidable

Proof:

We construct a TM that decides on E_{CFG}

$M = M(\langle G \rangle)$, G is a CFG:

1. **Mark all terminal symbols in G**
2. **Repeat:**
 - ▶ **Mark any variable A where G has a rule $A \rightarrow U_1 U_2 \dots U_k$ and each U_i is already marked**
 - Until no new variables got marked**
3. **If the start variable is marked, reject, otherwise accept**



13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Exercise 2.2:

Lemma 49

The class of context-free languages is not closed under complementation or intersection.

Hence, a similar proof as with EQ_{DFA} does not work.

A technique for showing that EQ_{CFG} **is undecidable** is presented later (Ch. 5).

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 50 (Theorem 4.9)

Every context-free language is decidable.

Faulty proof:

Let G be the CFG and P its PDA. Simulate P with a Turing machine M . □

Oops, M may have infinite loop!

Correct proof:

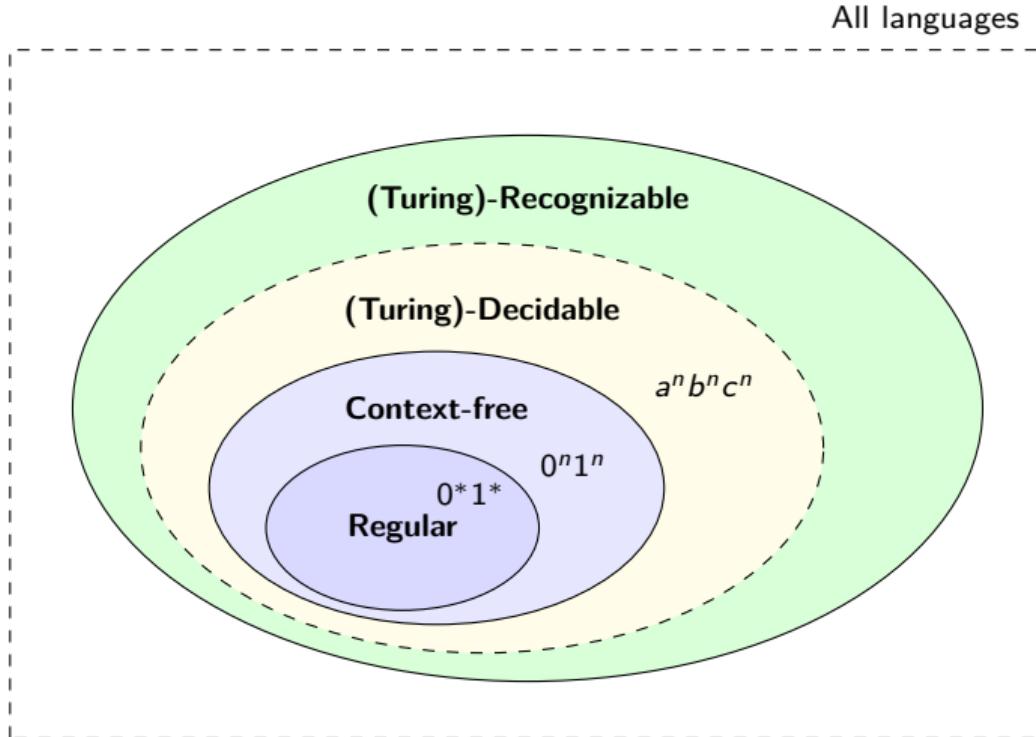
Let A be a CFL with CFG G . We need a TM M_G that **decides** A . Informally:

- ▶ Let TM S be from Theorem 4.7 for $\langle G, w \rangle$
- ▶ S is decider, execute it for $\{G, w\}$
- ▶ Accept if S accepts w , otherwise reject

M_G = "On input w :

1. Run TM S from Theorem 4.7 on input $\langle G, w \rangle$
2. If S accepts, accept; If S rejects, reject.





Any “life” in the two outer “regions”?

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

- ▶ So far we have studied computational models
 1. DFAs and NFAs
 2. PDAs
 3. TMs
- ▶ Many problems that can be solved using them
- ▶ Pumping Lemma can be used to show the opposite for DFAs, NFAs and PDAs
- ▶ TM is an abstract model for algorithms/procedures
 - ▶ If TM decides on A , then “answer” can be computed
 - ▶ If not, is there a TM that can recognize A ?
 - ▶ Or are there languages that are beyond TMs altogether?

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

TM: Acceptance

Definition 35 (Acceptance)

$$A_{TM} \triangleq \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$$

Problem: *Can we always determine if a TM accepts a w?*

Proposition 51

A_{TM} is Turing-recognizable

Proof:

Construct a TM⁵ U for input $\langle M, w \rangle$:

$U = U(\langle M, w \rangle)$:

1. Simulate M with w
 2. If M enters *accept*, accept; If M enters *reject*, reject
-

Note: As M may loop, U is a recognizer not decider.

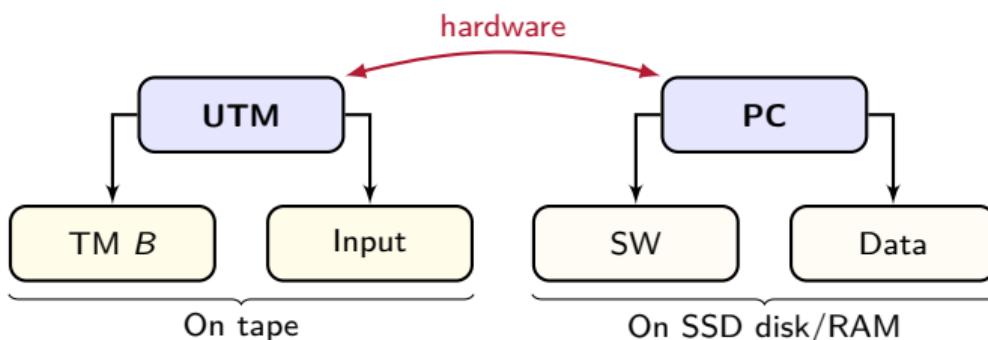


⁵Letter U stands for the *Universal* TM, i.e. UTM (Turing/1936)

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Universal Turing Machines (UTM)

- ▶ Turing's 1936 paper⁶: "*TM M can simulate any TM B*"
- ▶ Tape contains both i) the description of the TM B , and
ii) the input w
- ▶ TM M is known as the *Universal Turing Machine* (UTM)
- ▶ Important concept:
We do not have to build a new machine for every problem!



⁶ A. M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* 2.42 (1936), pp. 230–265.

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

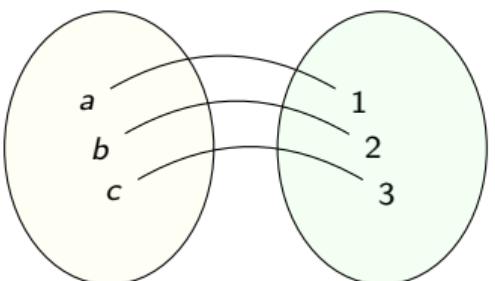
10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Cardinality (Size) of a Set



- ▶ For finite sets, cardinality of A is $|A|$
 - ▶ Same size if elements can be *paired*
- ▶ For infinite sets A and B , define
 - ▶ $|A| = |B|$ if a bijection $A \rightarrow B$ exists
 - ▶ $|A| \leq |B|$ if injection $A \rightarrow B$ exists
 - ▶ $|A| < |B|$ if (i) $|A| \leq |B|$ and (ii) $|A| \neq |B|$

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Countable set

Definition 36

Set \mathbb{X} is countable if it is finite, or a bijection between \mathbb{X} and $\mathbb{N} = \{1, 2, \dots\}$ exists

Informally, set \mathbb{X} can be stored in an (infinite) list.

- ▶ Each element can be given a unique “queueing number”

Examples of countable sets:

- ▶ Even numbers, 0, 2, 4, 6, ...
- ▶ Rational numbers \mathbb{Q}
- ▶ All strings in Σ^* with alphabet Σ

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Set of real numbers \mathbb{R}



Set of real numbers, \mathbb{R} , includes

- ▶ All rational numbers, \mathbb{Q}
- ▶ Irrational numbers such as $\pi, \sqrt{2}, e, \dots$

“Continuous mass to fill the gaps between the rational numbers”

Remark: Decimal representation is not unique

$$1 = 1.000\dots = 0.999999\dots$$

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

\mathbb{R} is uncountable

Theorem 52 (Theorem 4.17)

\mathbb{R} is uncountable

Proof:

Proof by contradiction: Suppose \mathbb{R} is countable, i.e., a bijection $f : \mathbb{N} \rightarrow \mathbb{R}$ exists.

The sequence $f(1), f(2), \dots$ thus lists all real numbers.

Next we construct $x \in (0, 1)$ that yields a contradiction.

Consider the decimal digit representation.

Choose the k th digit of x to be different from the k th digit of $f(k)$

- ▶ 0.999... = 1 problem is avoided by never choosing 0 or 9

The resulting x is a real number that no $f(n)$ maps to. □

This proof technique is known as the *diagonalization*

Lemma 53

The set of (infinite) binary sequences \mathbb{B} is uncountable

Proof:

Similarly as with Theorem 4.17 (this is also a tutorial problem!) □

Note:

- ▶ \mathbb{B} represents $[0, 1] \subset \mathbb{R}$
 - ▶ binary base
 - ▶ from .000000... to .1111111...
- ▶ The set of binary *strings* is countable
(check *the definition*)

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Lemma 54

Set of all languages \mathbb{L} over alphabet Σ is uncountable.

Proof:

Idea: construct a bijection $\mathbb{L} \rightarrow \mathbb{B}$

Let $\Sigma^* = \{s_1, s_2, \dots\}$ (countable!)

The characteristic sequence of language $A \in \mathbb{L}$ is $\{c_i\}_{i>0}$ where

$$c_i = \begin{cases} 1, & \text{if } s_i \in A \\ 0, & \text{otherwise} \end{cases}$$

Clearly, each language has a unique characteristic seq. in \mathbb{B} .

We have a bijection $\mathbb{L} \rightarrow \mathbb{B}$, and thus \mathbb{L} is uncountable. □

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Countable		Uncountable
Finite	Countably infinite	
$\{1, \dots, 10\}$	$\mathbb{N} = \{1, 2, \dots\}$	\mathbb{R}
$\{a, b, \dots, z\}$	\mathbb{Q}	\mathbb{C}
alphabet Σ	all strings Σ^*	\mathbb{L}

It follows,

- ▶ Power set of a finite set is finite (2^n elements)
- ▶ Power set of an countably infinite set is uncountable
 - ▶ cf. the set of binary sequences

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Lemma 55

Set of all Turing machines is countable.

Proof:

- ▶ Set of all strings with alphabet Σ is countable
- ▶ Each Turing machine has encoding to a string $\langle M \rangle$
- ▶ Skip strings not corresponding to a TM

□

Corollary 56 (Corollary 4.18)

Some languages are not Turing-recognizable.

Proof:

Set of all TMs is countable, yet set of all languages is uncountable.

□

Now this is interesting!

A_{TM} is Undecidable I

Theorem 57 (Theorem 4.11)

A_{TM} is undecidable

Proof in high-level:

1. Assume A_{TM} is decidable, and TM H decides on it
 - H analyzes whether TM M accepts string w
2. Define a new TM D , “a doctor”, that analyzes TMs:
“Healthy TMs reject the string representing themselves”
 - D accepts M if M rejects string $\langle M \rangle$, and otherwise rejects
3. Then someone asks this doctor, which is a TM, to analyze himself!
 - D shorts circuits and does not know what to say!

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Proof by contradiction:

Assume that A_{TM} is decidable and let H be a decider.

$H = H(\langle M, w \rangle)$:

1. H halts and accepts if M accepts w
2. ~~H halts and rejects if M does not accept w~~

Let M be a TM and define TM D with input $\langle M \rangle$:

$D = D(\langle M \rangle)$:

1. Run H on input $\langle M, \langle M \rangle \rangle$
2. If H accepts, reject
3. Otherwise, accept

Thus, D is a decider and works as follows:

$$D(\langle M \rangle) = \begin{cases} \text{accept}, & \text{if } M \text{ does not accept } \langle M \rangle \\ \text{reject}, & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

Consider then D with input $\langle D \rangle$, yielding

$$D(\langle D \rangle) = \begin{cases} \text{accept}, & \text{if } D \text{ does not accept } \langle D \rangle \\ \text{reject}, & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

That is: D accepts itself if it does not accept itself, and vice versa. This is a contradiction, and thus A_{TM} is undecidable.

□

We forced a contradiction by feeding a TM to itself!

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Summary so far

We have shown that languages can be

- ▶ Turing-decidable
 - ▶ e.g., A_{DFA} and any CFL
- ▶ Turing-recognizable but not decidable
 - ▶ e.g., A_{TM}

Interestingly, it turns out that also

Turing-unrecognizable languages exist!

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 58 (Theorem 4.22)

A language is decidable iff it is Turing-recognizable and co-Turing-recognizable.

Proof:

First, if A is decidable, then (i) it is recognizable, and (ii) also its complement is recognizable.

Next, let A and \overline{A} be Turing-recognizable with M_1 and M_2 as respective TMs. Define a new TM as follows:

$M = M(w)$:

1. Run both M_1 and M_2 on input w in parallel (two tapes!)
2. If M_1 accepts, accept
3. If M_2 accepts, reject

Clearly,

- ▶ Every string w is either in A or in \overline{A}
- ▶ Exactly one of M_1 or M_2 accepts it
- ▶ Thus, M always halts, and it is a decider that accepts $w \in A$



Turing-unrecognizable languages

As a corollary, we have found a language that is **not** even Turing-recognizable:

Corollary 59 (Corollary 4.23)

$\overline{A_{TM}}$ is a Turing-unrecognizable language.

Cross-check: What does all this really mean!?

- ▶ $s = \langle M, w \rangle$ is a string that *decodes* into
 1. TM M
 2. string w
- ▶ $s \in A_{TM}$ if M is a valid TM and it accepts w
- ▶ All other strings belong to $\overline{A_{TM}}$

Corollary says that there is no Turing machine to decide whether an arbitrary string s belongs to $\overline{A_{TM}}$ or not.

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

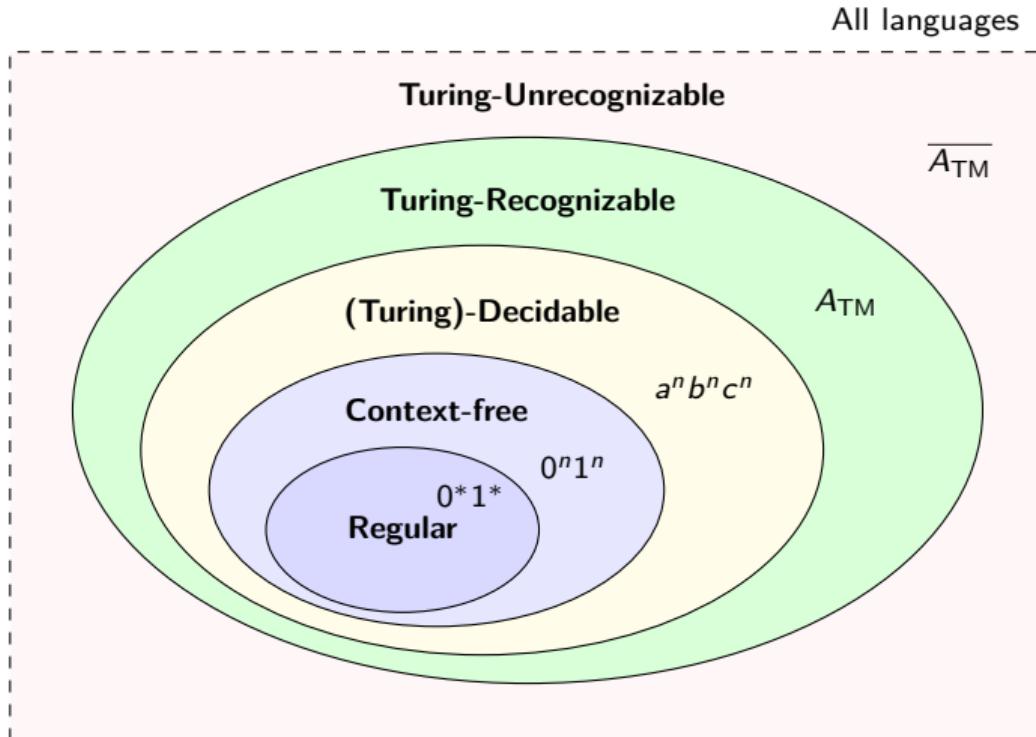
9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review



Now no “region” is uninhabited!

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

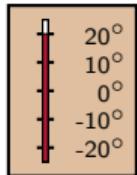
10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

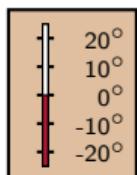
Summary



Turing-decidable languages are *desirable*

- ▶ They are algorithmically solvable
- ▶ For example A_{DFA}

“Get a TM, run it with input w , and answer will be available”

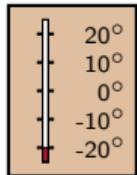


Some language are only Turing-recognizable

- ▶ For example A_{TM}

“TM may return *accept/reject*, but may just keep quiet!”

Even worse, some languages are Turing-unrecognizable!



- ▶ For example $\overline{A_{\text{TM}}}$
- ▶ No TM can even recognize them!

Thanks!

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Lecture 9

Reducibility

1. Turing machines

- ▶ Finite automaton with an infinite tape
- ▶ Two halting states: *accept* and *reject*

2. Church-Turing thesis

- ▶ “Anything that can be computed by an algorithm, can be computed with a TM”
- ▶ Not formally proven, hence a *conjecture*
- ▶ No known counterexamples . . . so generally accepted

3. Recognizable vs. decidable languages

4. . . and the unrecognizable languages

Today: “*methods to show (un)decidability*”

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

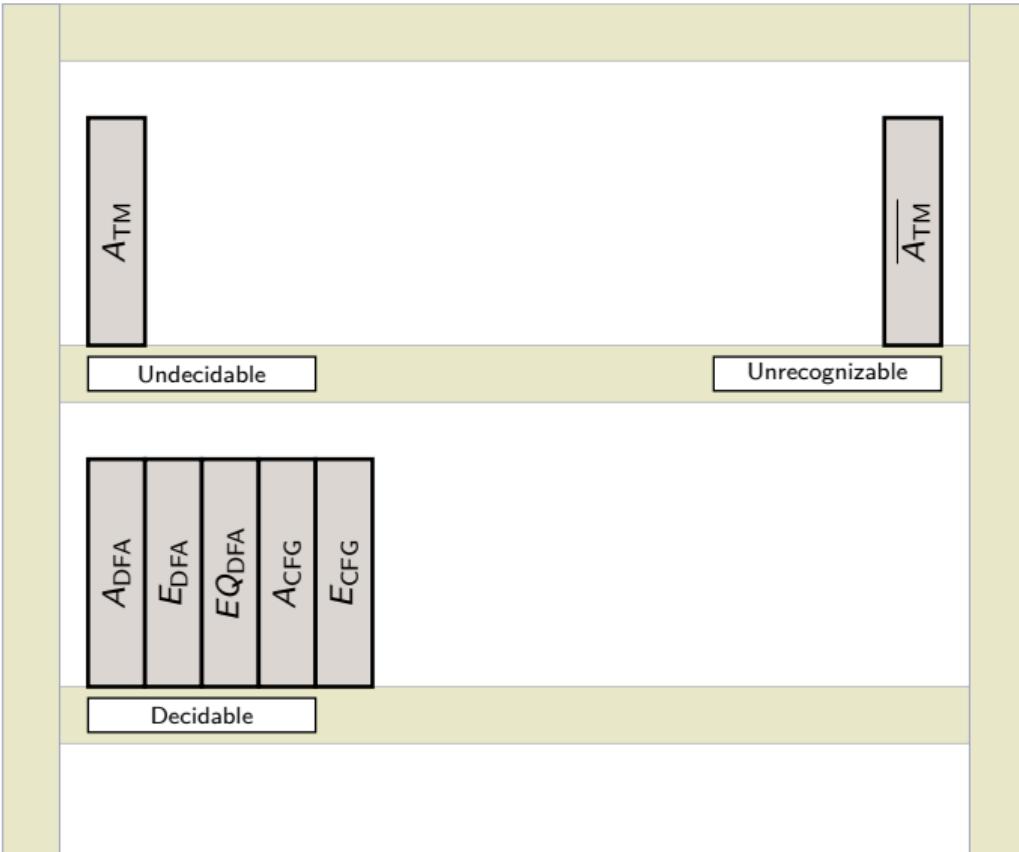
12 Brute Force,
SPACE and
Probabilities

13 Review

Great Library of Computability

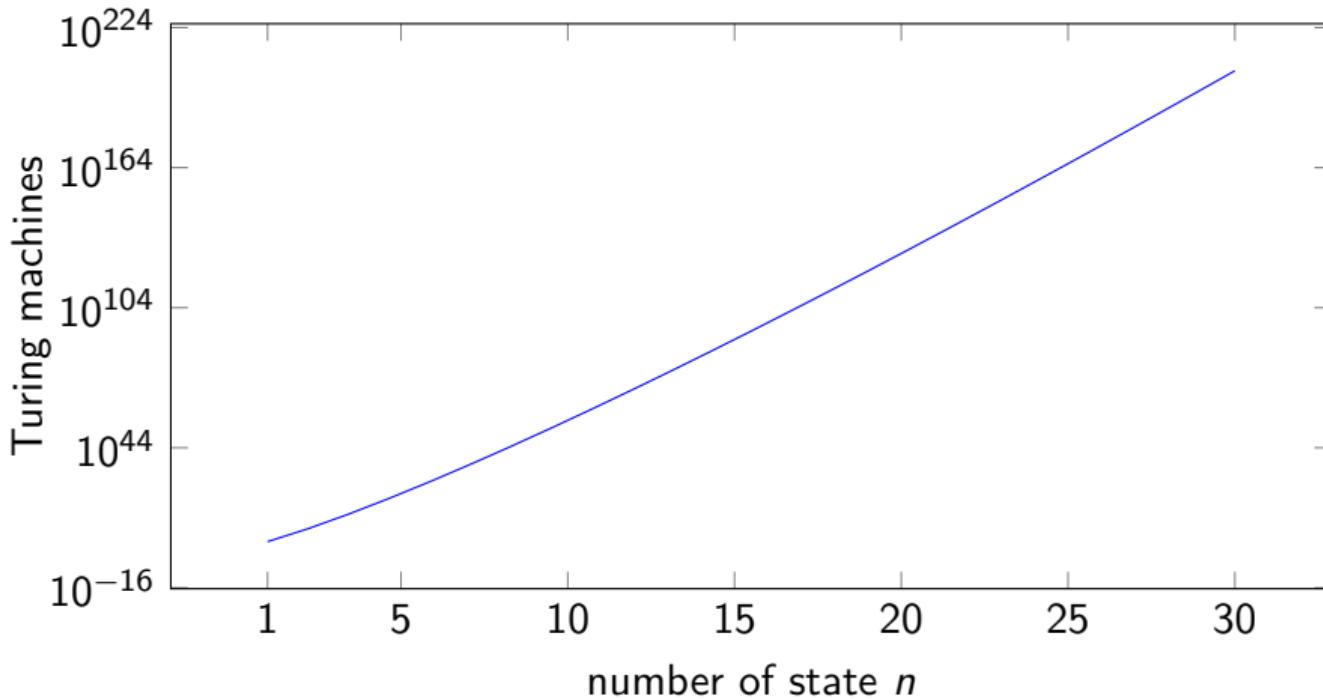
TÖL301G

E. Hyttiä



- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

Enumerating Turing machines



- ▶ Set of Turing machines is countable
- ▶ Let $\Sigma = \{0, 1\}$
- ▶ Above graph shows the number of Turing machines with n states (excluding

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Reducibility

Reduction is a way of converting one problem to another problem in such a way that a solution to the second problem can be used to solve the first problem.

Reducibility involves two problems, A and B . If A reduces to B , we can use a solution to B to solve A .

Examples:

1. All-pairs shortest path problem reduces to the problem of finding a shorter path from a to b
2. Problem of solving a system of linear equations reduces to the problem of inverting a matrix

Computability theory: Suppose A reduces to B

1. If B is decidable, then A also is decidable
2. If A is undecidable then B is undecidable

“Determine whether a given Turing machine halts (by accepting or rejecting) on a given input?”

Let us first define the corresponding language:

Definition 37 (Halting)

$$\text{HALT}_{TM} \triangleq \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem 60 (Theorem 5.1)

HALT_{TM} is undecidable.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Typical steps to show undecidability:

Steps

- ▶ Assume that a TM R exists that decides on A
- ▶ Use R to assist/construct TM S that decides on A_{TM}
- ▶ A_{TM} is undecidable, a contradiction, no such R can exist!
- ▶ Therefore A must be undecidable

“TM R would be too powerful exist”

Proof:

(Proof by contradiction) The result follows by reducing A_{TM} to HALT_{TM} and noting that A_{TM} is undecidable.

Suppose TM R decides on HALT_{TM} .

Then construct another Turing machine, TM S :

$S = S(\langle M, w \rangle)$:

1. Run TM R on $\langle M, w \rangle$
2. If R rejects, halt on reject
3. If R accepts, simulate M on w until it halts⁷
4. If M accepts, accept; otherwise reject

Clearly S decides on A_{TM} , which is known to be *undecidable*, and therefore no such R can exist, and HALT_{TM} is undecidable. □

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

⁷Guaranteed to halt now

“Does a given TM accept any strings?”

Definition 38 (Empty language)

$$E_{TM} \triangleq \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

Recall that E_{DFA} and E_{CFG} are decidable.

Unfortunately, with TMs the result is “negative” again:

Theorem 61 (Theorem 5.2)

E_{TM} is undecidable.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Proof:

First, define M_1 that depends on input w :

$M_1 = M_1^{(\langle M, w \rangle)}(\langle x \rangle)$:

1. Run M with w
 2. Halt on reject if M rejects. Otherwise halt on accept
-

Note: string w and TM M are part of M_1 :s description, x is ignored.

Suppose a TM R decides E_{TM} . Construct another TM S :

$S = S(\langle M, w \rangle)$:

1. Construct M_1 for $\langle M, w \rangle$
 2. Run R on input $\langle M_1 \rangle$
 3. Halt on accept if R rejected; otherwise halt on reject
-

Note that S must be able to compute a description of M_1 from $\langle M, w \rangle$.

Given R decides on E_{TM} , then S decides on A_{TM} , which contradicts the fact that A_{TM} is undecidable. Therefore no such R can exist, and E_{TM} is undecidable. \square

From TM to Regular languages

“Is the language $L(M)$ of TM M regular?”

Is a TM an unnecessarily powerful computational model for a given problem?
Could we use a DFA instead?

“Can a TM determine if $L(M)$ is regular?”

Definition 39

$REGULAR_{TM} \triangleq \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$

Unfortunately, the answer is negative again:

Theorem 62 (Theorem 5.3)

$REGULAR_{TM}$ is undecidable.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

REGULAR_{TM} is undecidable

Proof:

Suppose that REGULAR_{TM} is decidable and let R be a TM that decides on it. Then construct TM S as follows:

$S = S(\langle M, w \rangle)$:

1. Construct the following TM M_2

$M_2 = M_2(x)$:

- 1.1 If x has the form $0^n 1^n$, accept
- 1.2 If x does not have this form, run M on input w ,
and accept if M accepts w

2. Run R on input $\langle M_2 \rangle$

3. If R accepts, accept; otherwise reject

Hence, S decides on A_{TM} , which is impossible, and therefore no such R can exist, and REGULAR_{TM} is undecidable. □

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

TM: equal languages

“Do two TMs define the same language?”

Definition 40 (EQ for Turing machines)

$$EQ_{TM} \triangleq \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

Theorem 63 (Theorem 5.4)

EQ_{TM} is undecidable.

Proof:

Suppose that TM R decides EQ_{TM} . Then construct a TM S :

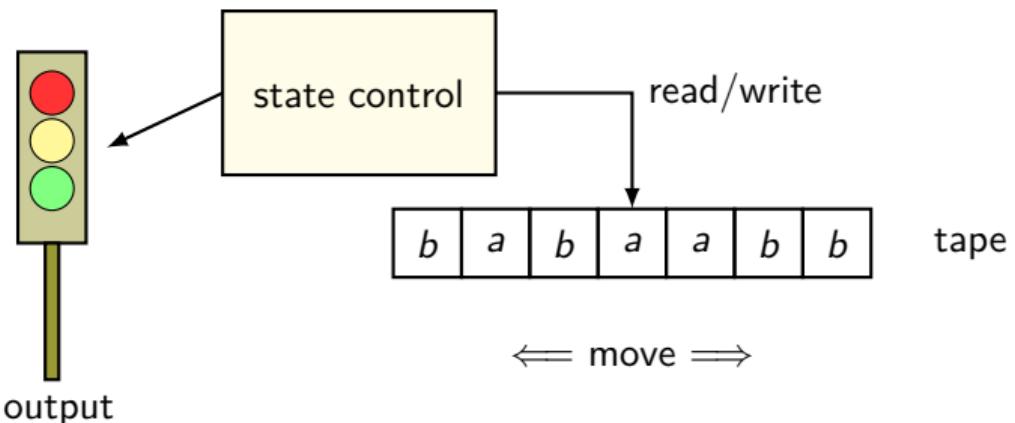
$S = S(\langle M \rangle)$, M is a TM:

1. Run R on input $\langle M, M_1 \rangle$, where TM M_1 rejects all inputs
2. If R accepts, accept; Otherwise reject

The above S decides E_{TM} , which is undecidable by *Theorem 5.2*. A contradiction and thus EQ_{TM} must be undecidable. \square

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Schematic: Linear Bounded Automaton (LBA)



- ▶ Same elements as in Turing machine
- ▶ Tape is finite with length equal to the input string w
 - ▶ It scales with the problem description size
- ▶ Tape alphabet can be larger than input alphabet

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Linear bounded automaton (LBA)

Definition 41 (Def. 5.6)

A **linear bounded automaton (LBA)** is a restricted type of TM wherein the tape head isn't permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is.

- ▶ Finite tape where you can only *overwrite*!
- ▶ Expanding the tape alphabet increases *available memory* linearly
- ▶ Despite of memory constraint, LBAs are quite powerful:
 - ▶ Deciders for A_{DFA} , A_{CFG} , E_{DFA} , and E_{CFG} all are LBAs
 - ▶ Every CFL can be decided by an LBA

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

LBA: some results

Lemma 64 (Lemma 5.8)

Let M be an LBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of M for a tape of length n .

Theorem 65 (Theorem 5.9)

A_{LBA} is decidable.

Proof:

The algorithm that decides A_{LBA} is as follows.

$L = L(\langle M, w \rangle)$, M is an LBA, w a string:

1. Simulate M on w for qng^n steps, or until it halts.
2. If M has halted, accept if it has accepted, otherwise reject
3. If M has not halted, reject

Latter must be a repeating configuration (Lemma 5.8), and therefore M is looping. \square

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Computation history

Computation history is the sequence of configurations on input w until a halting state:

Definition 42 (Def. 5.5)

Let M be a Turing machine and w an input string. An accepting computation history for M on w is a sequence of configurations, C_1, C_2, \dots, C_l , where C_1 is the start configuration of M on w , C_l is an accepting configuration of M , and each C_i legally follows from C_{i-1} according to the rules of M . A rejecting computation history for M on w is defined similarly, except that C_l is a rejecting configuration.

- ▶ Computation histories are finite sequences
- ▶ If M does not halt \Rightarrow no history
- ▶ (Nondeterministic TMs may have many histories on a single input)
- ▶ **Useful for undecidability proofs!**

"Is $L(M)$ an empty language for a given LBA M ?"

First, define again the corresponding language:

Definition 43 (E-LBA)

$$E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA and } L(M) = \emptyset\}$$

Theorem 66 (Theorem 5.10)

E_{LBA} is undecidable.

Proof:

See the book. (uses the computation histories)



0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

“Is there an algorithm that can deduce if a CFG generates all possible strings?”

Definition 44 (ALL for CFG)

$$\text{ALL}_{\text{CFG}} \triangleq \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$$

Theorem 67 (Theorem 5.13)

ALL_{CFG} is undecidable.

Proof:

See the book. (By contradiction, using computation histories) □

Homework problem: Also EQ_{CFG} is undecidable.

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

[0 Introduction](#)[1 DFA](#)[2 NFA](#)[3 Regexp](#)[4 Nonregular L](#)[5 Context-Free](#)[6 Pushdown](#)[7 Turing](#)[8 Decidability](#)[9 Reducibility](#)[10 Time](#)[11 Complete](#)[12 Brute Force,
SPACE and
Probabilities](#)[13 Review](#)

Computable function

Definition 45

A function $f : \Sigma^* \rightarrow \Sigma^*$ is a computable function if some TM, on every input w , halts with just $f(w)$ on its tape.

Examples of computable functions:

- ▶ All arithmetic operations on integers
- ▶ Computing the (integer) square root of x
- ▶ Sorting a list of strings / numbers
- ▶ Counting the length of the input string

Definition 46 (Turing complete)

Any system (computing model, programming language, ...) is called Turing complete if it can compute every possible computable function, i.e., it can simulate a Turing machine.

Some examples:

- ▶ Common programming languages (C, Java, Python, ...)
- ▶ Conway's Game of life

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Definition 47 (Def. 5.20)

Language A is mapping reducible to language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^ \rightarrow \Sigma^*$, where for every w , $w \in A \iff f(w) \in B$. The function f is called the reduction from A to B.*

Example:

- A_{TM} reduces to $HALT_{TM}$

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Mapping reducibility (2)

Theorem 68 (Theorem 5.22)

If $A \leq_m B$ and B is decidable, then A is decidable.

Proof:

Let M be the decider for B and f the reduction from A to B .

$N = N(w)$:

1. Compute $f(w)$
2. Run M on input $f(w)$, output whatever M outputs

If $w \in A$, then $f(w) \in B$ as f is a reduction from A to B , i.e., M accepts $f(w)$ iff $w \in A$ and thus N decides on A . \square

Corollary 69

If $A \leq_m B$ and A is undecidable, then B is undecidable.

Note: We have used the above already multiple times!

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 70 (Theorem 5.28)

If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable.

Proof:

The proof is the same as that of Theorem 5.22, except that M and N are recognizers instead of deciders. \square

Corollary 71 (Corollary 5.29)

If $A \leq_m B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Theorem 72 (Theorem 5.30)

EQ_{TM} is neither Turing-recognizable or co-Turing-recognizable.

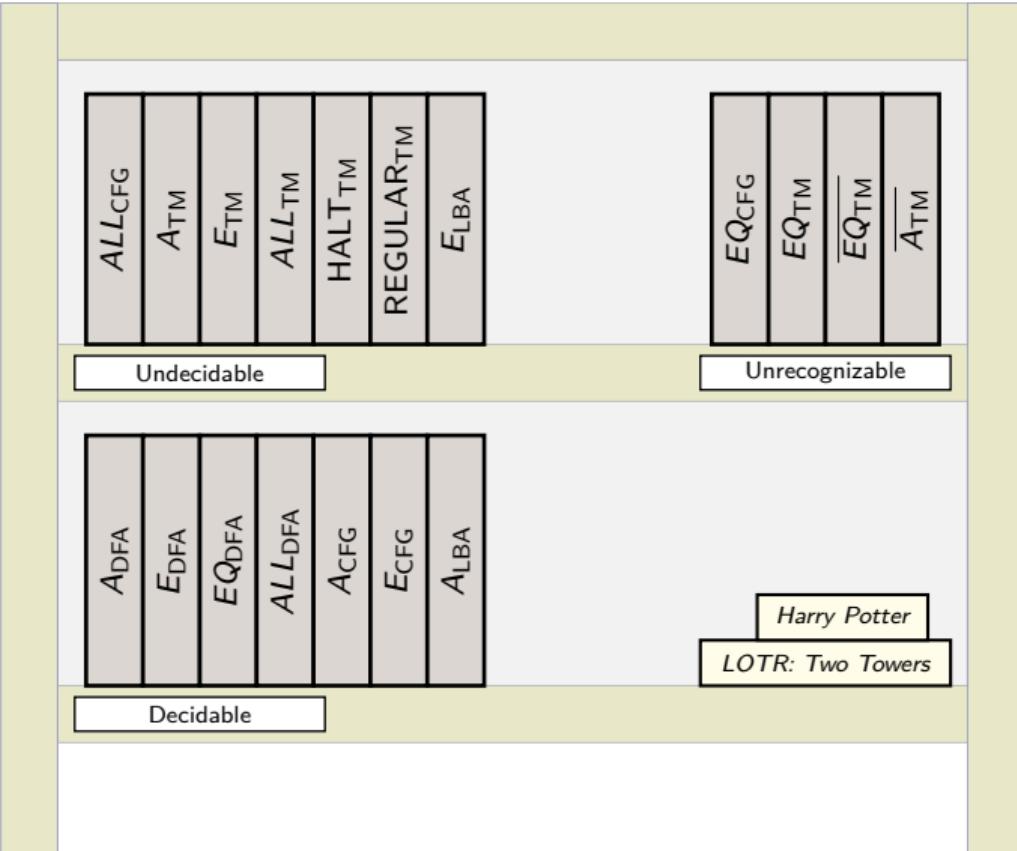
Proof:

See the book. \square

Great Library of Computability

TÖL301G

E. Hyttä



Summary

1. Undecidable languages:

HALT_{TM} , E_{TM} , EQ_{TM} , $\text{REGULAR}_{\text{TM}}$

2. Linear bounded automaton:

- ▶ More powerful than PDA – can decide on any CFL
- ▶ A_{LBA} is decidable
- ▶ E_{LBA} is not

3. Reducibility: A reduces to B ...

4. Mapping reducibility:

- ▶ Computable function $f(w)$ from A to B
(and their complements)
- ▶ Notion: A reduces to B

Thanks!

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Unary encoding

Unary numbers are perhaps the most elementary number system. Number n is represented by n symbols

1	1
2	11
3	111
:	:

Unary coding works the same way, but some symbol is used as terminator. For example,

0	0
1	10
2	110
3	1110
:	:

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time**
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Lecture 10

Time Complexity (P,NP)

1. Turing machines

- ▶ Computational model for “real computers”
- ▶ Results about “computability”
 - ▶ Decidable: Given time and memory . . . can be done

2. Reducibility

- ▶ Proof technique for (un)decidable problems
- ▶ Computational function, Turing machine exists

... So far we have ignored the time aspect!

Today: *Time complexity*

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

- ▶ Recap: TM is a model for computation
 - ▶ Input string w
 - ▶ Result:
 1. Accept or Reject state (decision problems)
 2. String on the tape
- ▶ Default TM:
 - ▶ Deterministic
 - ▶ Single-tape

$$r(w) \triangleq \text{"the number of steps before halt on input } w\text{"}$$
 (3)

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

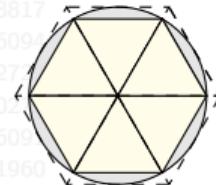
3.

The beginning:

In 250BC, Archimedes developed an *algorithm* and computed

$$\frac{223}{71} < \pi < \frac{22}{7}$$

i.e., $3.1408 < \pi < 3.1429$, almost **3 digits!**



480 AD, Chinese mathematician Zu Chongzhi calculated 7 digits,

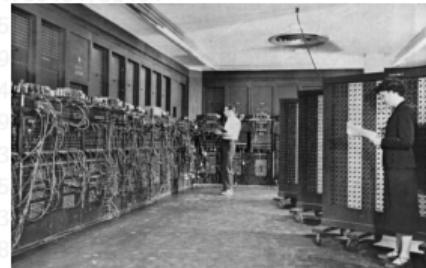
$3.1415926 < \pi < 3.1415927$

Computer era:

In 1949, John von Neumann et al. computed **2037**

digits in 70 hrs with ENIAC (the supercomputer of the era) using Machin's formula

$$\frac{\pi}{4} = 4 \arctan \frac{1}{5} - \arctan \frac{1}{239} \quad (\text{Machin, 1706})$$

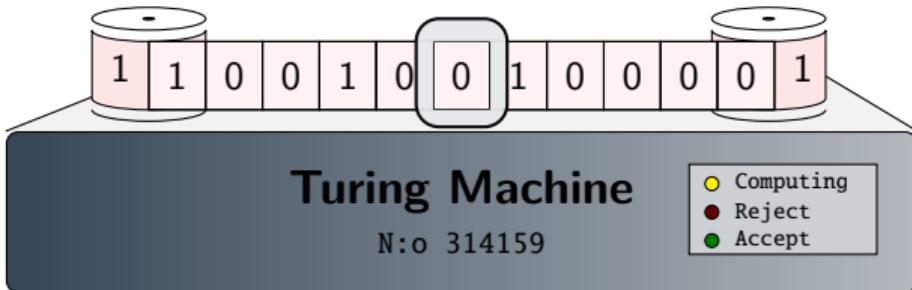


- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexp
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

Digits of π with a Turing machine

TÖL301G

E. Hyttiä



- The infinite tape \Rightarrow any number of digits!
- Two-tape TM M_π implements Machin's formula

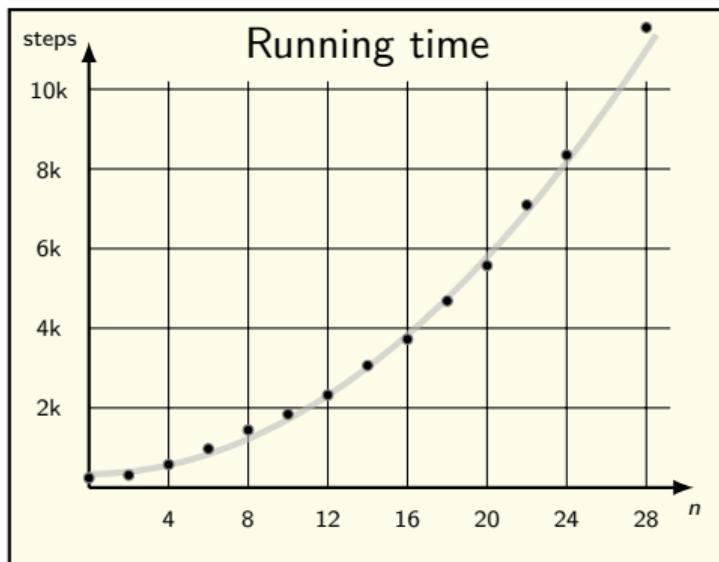
Results with a two-tape machine:

n	result		time
8	11.00100101	3.14	1446
12	11.001001000101	3.142	2324
16	11.0010010001000001	3.1416	3728
20	11.00100100001111110110	3.14159	5567
24	11.001001000011111101101011	3.1415927	8344

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

Running time with TM M_π computing digits of π

Experiments gives some insight to the running time:



The gray curve is a 2nd degree polynomial function of n .

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

Good news: Running time $r(w)$ is well-defined:

- ▶ number of steps before TM halts

Bad news: $r(w)$ can be extremely complicated function...

- ▶ ... and w can be of arbitrary length

How to summarize if $r(w)$ was available?

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Measuring Time Complexity

Idea 1: Determine the maximum running time

$$\max_w r(w)$$

- ▶ Input string can be arbitrarily long
- ▶ ... and the above often gives the infinity (cf. TM M_π)

Idea 2: Determine the mean running time

- ▶ Countably infinite number of input strings
- ▶ Need a probability distribution $p(w)$ for input strings
- ▶ Input W is a random variable, and

$$m = \mathbb{E}[r(W)] = \sum_w p(w) \cdot r(w)$$

- ▶ But we do not know the distribution of W !

Idea 3: Condition on the length of the input string

- ▶ Bigger problem instances *tend to be* computationally more demanding

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Worst-case analysis

So we proceed as follows:

Definition 48 (Def. 7.1)

Let M be a deterministic Turing machine that halts on all inputs. The running time or time complexity of M is the function $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the **maximum number of steps** that M uses on any input of length n .

In terms of $r(w)$

$$f(n) = \max_{|w|=n} r(w)$$

If $f(n)$ is the running time of M , we say that

1. " M runs in time $f(n)$ ", and that
2. " M is an $f(n)$ time Turing machine".

Note: No assumptions regarding the distribution

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

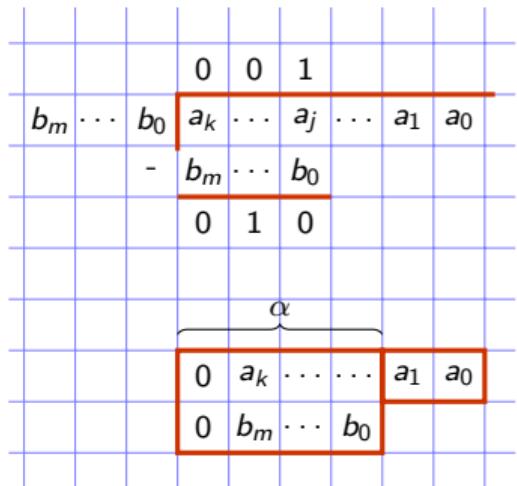
10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example: Long division



Repeat while $|a| \geq |b|$:

1. Check if $\alpha \geq b$
2. If yes, subtract it (in place) and write 1 to the result
3. If not, write 0 to the result
4. Shift a one left

What is the running time?

Our two-tape Turing machine M_{div} carrying out the long division expects the input string to be of form $w = a_{k-1} \dots a_0 / b_{m-1} \dots b_0$, so that $n = |w| = k + m + 1$. We find that M_{div} runs in time

$$f(n) = n^2 + 6n + 7.$$

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Similarly, the mean running time on condition $|w| = n$:

Definition 49

Let M be a deterministic Turing machine that halts on all inputs. The average-time complexity of M is the function $g : N \rightarrow N$, where $g(n)$ is the **mean number of steps** that M uses on inputs of length n (uniform distribution).

In terms of $r(w)$

$$g(n) = \frac{1}{|\Sigma|^n} \sum_{|w|=n} r(w)$$

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Asymptotic Analysis

Issues with $r(w)$ and $f(n)$:

1. Exact running time $r(w)$ is often a complex expression
 - ▶ We were “lucky” with $TM M_{div}$
 - ▶ With M_π , the trend is clear but there is “inherent noise”
 - ▶ In practice, the situation is often much worse!
2. Short inputs often irrelevant as they are “fast” anyway

Instead:

Asymptotic behavior is interesting when n gets larger

- ▶ This is where the difficult instances tend to be

Recall that with polynomials, the highest degree term will eventually dominate:

$$a_0 + a_1x + a_2x^2 + \dots + \underline{a_kx^k} \Rightarrow \underbrace{x^k}_{\text{Interesting part!}}$$

(In plots, consider the log-scale)

Definition 50 (Def. 7.2)

Let f and g be functions $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$. We say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$, $f(n) \leq cg(n)$. When $f(n) = O(g(n))$, we say that $g(n)$ is an upper bound for $f(n)$, or more precisely, that $g(n)$ is an asymptotic upper bound for $f(n)$, to emphasize that we are suppressing constant factors.

Big-O notation gives an upper bound (for large n):

Polynomial bounds: n^c , $c > 0$

Exponential bounds: $2^{(n^\delta)}$, $\delta > 0$

Big-O: “No more than” bound

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

[0 Introduction](#)[1 DFA](#)[2 NFA](#)[3 Regexp](#)[4 Nonregular \$L\$](#) [5 Context-Free](#)[6 Pushdown](#)[7 Turing](#)[8 Decidability](#)[9 Reducibility](#)[10 Time](#)[11 Complete](#)[12 Brute Force,
SPACE and
Probabilities](#)[13 Review](#)

Small-o notation

Definition 51 (Def. 7.5)

Let f and g be functions $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$. We say that $f(n) = o(g(n))$ if

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0.$$

That is, $f(n) = o(g(n))$ means that for any real number $c > 0$, a number n_0 exists s.t. $f(n) < cg(n)$ for all $n \geq n_0$.

Small-o: $f(n)$ is asymptotically less than $g(n)$

Example 73 (TM deciding on lang. $A = \{0^n 1^n \mid n \geq 0\}$)

$M_1 = M_1(w)$:

1. Scan across the tape
Reject if a 0 is found to the right of a 1
2. Repeat if both 0s and 1s remain on the tape:
Scan across the tape
crossing off a single 0 and a single 1
3. If 0s or 1s still remain, *reject*, otherwise, *accept*

Stage 1:

- ▶ Scan verifies that the input is of the form $0^* 1^*$
- ▶ Scan uses n steps, where n is length of the input
- ▶ Repositioning the head at the start uses another n steps
- ▶ In total $2n$ steps \Rightarrow Big-O, $O(n)$ steps

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Stages 2:

- ▶ Repeated scans crossing off a 0 and 1 each time
- ▶ Each scan
 - ▶ Uses $O(n)$ steps
 - ▶ Crosses off two symbols \Rightarrow at most $n/2$ rounds
- ▶ Total time: $(n/2) O(n) = O(n^2)$ steps

Stage 3:

- ▶ Single scan to decide whether to accept or reject
- ▶ The time taken is at most $O(n)$

Sum up:

- ▶ The total time of M_1 on an input of length n is

$$O(n) + O(n^2) + O(n) = \boxed{O(n^2)}$$

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- ▶ We have Big-O and small-o notations
- ▶ We know how to analyze a given TM (that halts)
- ▶ Thus, we can argue how algorithm performs asymptotically
 - ▶ Ignoring constant factors

(In practice a constant factor improvement can be significant!)

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Definition 52 (Def. 7.7)

Let $t : \mathbb{N} \rightarrow \mathbb{R}^+$ be a function. Define the **time complexity class**, $\text{TIME}(t(n))$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

Example 74

TM M_1 deciding on language $A = \{0^n 1^n \mid n \geq 0\}$ was $O(n^2)$, and thus $A \in \text{TIME}(n^2)$.

Faster algorithm for $A = \{0^n 1^n \mid n \geq 1\}$

Example 75

$M_2 = M_2(w)$ for $A = \{0^n 1^n \mid n \geq 1\}$:

1. Scan the input and *reject* if a 0 is found to the right of a 1
2. Repeat as long as some 0s and some 1s remain on the tape:
 3. Scan, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*
 4. Scan again, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1
 5. If no 0s and no 1s remain, *accept*. Otherwise, *reject*

TODO

1. Verify the algorithm
2. Determine the time complexity

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Time complexity of M_2

- ▶ Every stage takes $O(n)$ time
- ▶ How many times each is executed?
- ▶ Stages 1 and 5 are executed once:
 - ▶ Takes a total $O(n)$ time
- ▶ Stage 4 crosses off at least half the 0s and 1s each time it is executed, so at most $1 + \log_2 n$ iterations
- ▶ Total time of stages 2, 3, and 4 is $(1 + \log 2n)O(n) = O(n \log n)$
- ▶ The running time of M_2 is $O(n) + O(n \log n) = \boxed{O(n \log n)}$

Improvement, from $O(n^2)$ to $O(n \log n)$

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Summary of TMs deciding on CFL $A = \{0^n 1^n \mid n \geq 1\}$:

- ▶ First, $A \in \text{TIME}(n^2)$
- ▶ Now, $A \in \text{TIME}(n \log n)$
 - ▶ This cannot be further improved on single-tape TM

Theorem 76 (Kobayashi 1985)

Any language that can be decided in $o(n \log n)$ time on a single-tape Turing machine is regular.

Language $A = \{0^n 1^n \mid n \geq 1\}$ with two-tape TM

Time-complexity can be reduced further with “*better hardware*”:

Example 77

$M_3 = M_3(w)$ for $A = \{0^n 1^n \mid n \geq 1\}$:

1. Scan across tape 1 and reject if a 0 is found to the right of a 1
2. Scan across the 0s on tape 1 until the first 1
 - At the same time, copy the 0s onto tape 2
3. Scan across the 1s on tape 1 until the end of the input.
 - For each 1 read on tape 1, cross off a 0 on tape 2.
 - If all 0s are crossed off before all the 1s are read, *reject*.
4. If all the 0s crossed off, *accept*; otherwise *reject*.

M_3 decides on A in **linear time**, $O(n)$

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

“Time out”

Computability theory: The Church-Turing thesis

“All reasonable models of computation are equivalent”

They all decide the same class of languages.

Complexity theory:

The choice of model affects the time complexity.

Example 78

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

- ▶ Decidable in $O(n \log n)$ time with 1-tape TMs
- ▶ Decidable in linear time $O(n)$ with 2-tape TMs

Still we want to classify computational problems according to their time complexity!

But with which model?

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Single-tape TM vs. Multitape TM

Theorem 79 (Thm. 7.8)

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

Proof:

(Idea): In *Theorem 3.13*, we showed how to convert any *multitape TM* into a single-tape TM that simulates it. One can analyze that simulation to determine how much additional time it requires.

It turns out that simulating each step of the multitape machine uses at most $O(t(n))$ steps on the single-tape machine. Hence the total time used is $O(t^2(n))$ steps. \square

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Nondeterministic TMs:

- ▶ How to measure time with them?
- ▶ Recall the power sets!

Definition 53 (Def. 7.9)

Let N be a nondeterministic Turing machine that is a decider. The running time of N is the function $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n .

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Nondeterministic TMs: running time

The running time with NTMs is illustrated below:

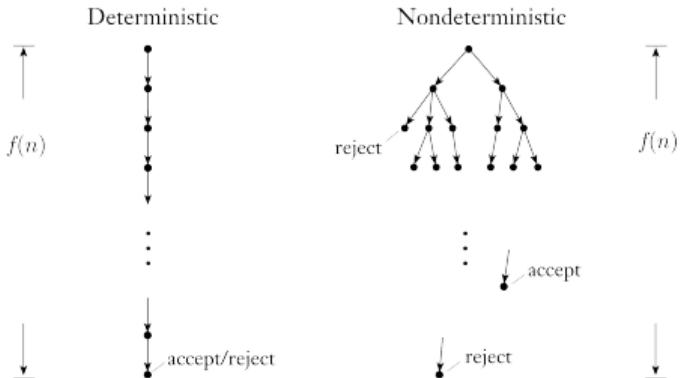


Figure: Deterministic vs. nondeterministic TMs (Fig. 7.10 / book).

Note:

- ▶ Def. of the running time of NTMs **is not** intended to correspond to any real-world computing device
- ▶ A useful mathematical definition that assists in characterizing the complexity of an important class of computational problems (see later)

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

NTM vs. Single-tape TM

Theorem 80 (Thm. 7.11)

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

Proof:

(Sketch) Proof of [Thm 3.16](#) constructed a deterministic TM D that simulated N by searching N 's nondeterministic computation tree. Results follows from analyzing that simulation. □

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Theorem 7.8:

“At most a square or polynomial difference between the time complexity of problems measured on deterministic single-tape and multitape TMs”

Theorem 7.11:

“At most an exponential difference between the time complexity of problems on deterministic and nondeterministic TMs”

*Polynomial differences are considered to be small,
whereas exponential differences are considered to be large*

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Summary (2)

- ▶ Exp. time is typical with **exhaustive search** a.k.a. **brute-force search**
- ▶ All reasonable deterministic computational models are **polynomially equivalent**
 - ▶ They can simulate each another with a polynomial increase in running time
- ▶ What is “reasonable”?
 - ▶ “models that closely approximate running times on actual computers”
 - ▶ E.g., *Theorem 7.8* shows that the deterministic single-tape and multitape Turing machine models are polynomially equivalent.

“How to multiply two n-bit integers?”

	The long multiplication was used for centuries	$O(n^2)$
1960	Anatoly Karatsuba proposes a better way	$O(n^{1.58})$
1965	FFT enters the game	$O(n \log(n) \cdot \log(\log(n)))$
1971	Schönhage and Strassen conjecture <i>“the true complexity is $O(n \log(n))$”</i>	
	(50 years passes . . .)	
2019	Harvey's and van der Hoeven's algorithm ⁸	$O(n \log n)$

⁸David Harvey, Joris van der Hoeven. *Integer multiplication in time $O(n \log n)$* . March 2019.
(hal-02070778).

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Next steps

- ▶ Next we focus on aspects of time complexity theory that are unaffected by polynomial differences in running time
- ▶ This allows a theory that is independent of the selection of a computation model (“hardware”)
- ▶ Goal: fundamental properties of computation
(not properties of TMs or alike)

Definition 54 (Def. 7.12)

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine,

$$P = \bigcup_k \text{TIME}(n^k)$$

Note:

- ▶ k can be arbitrarily large, but still finite
- ▶ P is *invariant* for all models of computation that are polynomially equivalent to the deterministic single-tape Turing machine
- ▶ P roughly corresponds to the class of problems that are realistically solvable on a computer

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

- ▶ Encoding method, e.g., $\langle w \rangle$, affects the result
- ▶ *Reasonable* methods allow for *polynomial time* encoding and decoding of objects into natural internal representations or into other reasonable encodings
- ▶ Familiar encoding methods for graphs, automata, and the like all are reasonable
- ▶ Reasonable encoding for graphs:
 1. List of nodes and edges
 2. Adjacency matrix (binary)
- ▶ Unary notation for encoding numbers isn't reasonable
 - ▶ It is exponentially larger than truly reasonable encodings, such as base k notation for any $k \geq 2$

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Problem: Path from s to d

The PATH problem is to determine whether a directed path exists from s to d .

Definition 55 (PATH)

Let $G = (V, E)$ denote a directed graph, with $m = |V|$ nodes, and (s, d) a source-destination node pair. Then

$$\text{PATH} \triangleq \{\langle G, s, d \rangle \mid G \text{ is a directed graph w/ a path from } s \text{ to } d\}$$

Algorithm to find such a path?

Possible approaches:

1. Brute-force
2. Random walk (cf. Machine learning)
3. Dynamic programming

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Path $s \rightarrow d$: Brute force

- ▶ Examine all potential paths in G
- ▶ How to enumerate them?
 - ▶ Loops obviously can be excluded
- ▶ **Idea 1:**
 - ▶ Path has at most m nodes
 - ▶ Consider m -tuples of nodes
 - ▶ Search space has m^m elements
 - ▶ Not very efficient but includes all potential paths
- ▶ **Idea 2:**
 - ▶ Each permutation π of V maps to a potential path
 - ▶ Search space has $m!$ elements
 - ▶ All potential paths are included

Both can be further reduced, e.g., by fixing the first node:

- ▶ Idea 1: $(m - 1)^{(m-1)}$
- ▶ Idea 2: $(m - 1)!$

Both are still exponential in the number of nodes ...

Path $s \rightarrow d$: Random walk

Probabilistic “random walk” algorithm:

$r_0 \leftarrow s$

repeat

$e \leftarrow$ random edge from E with origin r

$r \leftarrow$ destination of e

until $r = d$

return True

- ▶ This algorithm may eventually find the answer ...
- ▶ Maximum running time, however, is infinite⁹
- ▶ Hence, not very satisfactory for us
- ▶ But often used, e.g. in Machine learning

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

⁹Except in trivial cases where “wrong” turns cannot be made.

Theorem 81 (Thm. 7.14)

$\text{PATH} \in P$

Proof:

Construct a polynomial time algorithm:

$M = M(\langle G, s, d \rangle)$:

1. Mark node s
2. Repeat until no additional nodes are marked:
 - Scan all the edges (a, b) in G : If a is marked and b unmarked, mark b
3. If d is marked, accept, otherwise reject

TM M clearly “finds” a path if such exists, its running time is polynomial, and hence M is a polynomial time algorithm for PATH. \square

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Recap on Prime numbers

Definition 56

A positive integer number $n > 1$ is a **prime number** if it is divisible only by itself and 1.

- ▶ Hence 2, 3, 5, 7, 11, ... are the prime numbers
- ▶ Each $k > 1$ is either a prime or a *composite number*

Theorem 82 (Unique factorization theorem)

Each positive integer n has an unique factorization to primes

$$n = p_1^{n_1} \cdots p_k^{n_k} \quad (n > 1)$$

where the p_i are k distinct primes and the $n_i \in \mathbb{N}$

Factoring of large integers is an interesting problem:

- ▶ Brute force: try 2 and every odd k less than \sqrt{n}
- ▶ If n is large, choose candidates randomly?
- ▶ Much better algorithms exist ... but still demanding!

Prime numbers are used in public key cryptography:

Example 83 (RSA)

Choose two prime numbers p and q , and compute

$$n = pq$$

$$\phi = (p - 1)(q - 1)$$

Choose the public exponent e such that $1 < e < \phi$ and $\gcd(e, \phi) = 1$

Compute the secret key d such that $de \equiv 1 \pmod{\phi}$

Encoding: compute $c = m^e \pmod{n}$

(public key (n, e) is shared)

Decoding: compute $m = c^d \pmod{n}$

(private key (n, d) is kept)

Relies on assumption that factoring large numbers is difficult!

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example

Example 84 (Xbox)

The original Xbox used the following public key in signing binaries:

```
2074011932725872376027602350906301713845599360627488352673195511324110900735  
4362374128996096291046353572306742110305456946824862203867115042369878729703  
4757651122801674981890464377946029661688124194233651969796694319295889511268  
0464874302938783366603176573433716594963473137559247167029424618087781510481  
2674626967450097045005117546657068700545263064105024888769118032059917845867  
6530404194040036845598825091953986309228240504053796205135896999939802056942  
6697323609577215347638826741847653366351274624331031785386194643005307289050  
2949319703765023792161144942611323629444409600173894963797156859916567288947  
565058003
```

Can you factor the beast?!

(and run your own binaries)



The original Xbox (from Wikipedia).

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

PRIMES

“Is x a prime?”

Determining if x is a prime is an important question:

- ▶ cf. RSA algorithm

Definition 57 (Language of prime numbers)

$$\text{PRIMES} = \{\langle x \rangle \mid x \text{ is prime}\}$$

So $10^{6400} - 10^{6352} - 1 \in \text{PRIMES}!$

Theorem 85 (AgrawalKayalSaxena (2002))

$$\text{PRIMES} \in P.$$

Further reading:

<http://www.ams.org/notices/200305/fea-bornemann.pdf>

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Relative primes

Definition 58

Two positive integer numbers $p, q > 1$ are **relative primes** if $\gcd(p, q) = 1$

Let RELPRIME be the problem of testing whether two numbers are relatively prime

Definition 59

$\text{RELPRIME} = \{\langle x, y \rangle \mid x \text{ and } y \text{ are relatively prime}\}$

Theorem 86 (Thm. 7.15)

$\text{RELPRIME} \in P$.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Proof (1)

Proof:

TM E computes the gcd using the Euclidean algorithm:

$E = E(\langle x, y \rangle)$, $x, y \in \mathbb{N}$ in binary (Euclidean Alg.):

1. Repeat until $y = 0$
 2. Assign $x \leftarrow x \bmod y$
 3. Exchange x and y
 4. Output x ($=\gcd(x, y)$)
-

TM R solves RELPRIME using E as a subroutine:

$R = R(\langle x, y \rangle)$, $x, y \in \mathbb{N}$ in binary:

1. Run $E(x, y)$
 2. If the result is 1, accept, otherwise reject
-

Note:

- ▶ R runs in polynomial time, if E does
- ▶ Correctness of E is well-known
- ▶ Remains to analyze the running time of E

Proof (2): Running time of E (Euclid's alg.)

- ▶ Stage 2:
 - ▶ After the first round, $x > y$
 - ▶ In consecutive rounds, x drops by at least half as¹⁰
 - ▶ If $x/2 \geq y$, then $x \bmod y < y \leq x/2$
 - ▶ If $x/2 < y$, then $x \bmod y = x - y < x/2$
- ▶ The values of x and y are exchanged every time stage 3 is executed, so each of the original values of x and y are reduced by at least half every other round.
- ▶ Stages 2 and 3 are run at most $2 \log_2 \min\{x, y\}$ times
- ▶ These logs are proportional to the lengths of the representations \Rightarrow the number of stages run is $O(n)$.
- ▶ Each stage of E uses only polynomial time, so the total running time is polynomial.

¹⁰The modulo operation corresponding to remainder can be computed with an almost identical TM to M_{div} with a polynomial running time.

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 87 (Thm. 7.16)

Every context-free language is a member of P

Proof:

See the book.



0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example Languages from P

Example 88 (Perfect square)

The question whether \sqrt{x} is an integer or not,

$$\text{PERFECT} = \{x \in \mathbb{N} \mid \exists n \in \mathbb{N} \text{ s.t. } x = n^2\}.$$

- ▶ Basic (inefficient) brute force solution:
For $i = 0, 1, \dots$ until $i^2 \geq x$, and report if $i^2 = x$
- ▶ Plenty of faster options exist!

Example 89 (Digits of Pi)

Does a given string w represents π (up to given accuracy):

$$\text{PI} = \{w \in \{0, 1\}^* \mid w \text{ has } |w| \text{ digits of } \pi\}$$

Note: the binary point is implicit, e.g., $11001 \in \text{PI}$ as $11.001 \approx \pi$.

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example Languages from P (2)

Example 90 (Loops)

The question whether a path in the triangular grid is a loop

$$\text{LOOP} = \{w \in \{L, \ell, s, r, R\}^* \mid \text{route } w \text{ is a loop}\}.$$

- ▶ w defines the changes in the direction:

L steep left

ℓ left

s straight

r right

R steep right

- ▶ State is the (position, heading)-pair



0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- ▶ So we managed to avoid brute-force search in several problems
- ▶ ... and obtained polynomial time solutions
- ▶ Attempts to avoid brute force in some other problems has been unsuccessful
 - ▶ No polynomial time algorithm is known
 - ▶ Neither is it proven that they do not exist
 - ▶ This class is known as **NP**

(NP does not stand for “*no problem*” in our context!)

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Hamiltonian path

Hamiltonian path is a well-known combinatorial problem:

- ▶ Given a directed graph $G = (V, E)$
- ▶ Find a path that visits every node exactly once

Fixing two nodes gives HAMPATH problem:

Definition 60 (Hamiltonian Path problem)

$$\text{HAMPATH} \triangleq \{\langle G, s, d \rangle \mid G \text{ is a directed graph with h. p. } s \rightarrow d\}$$

Polynomial time verifiability:

Claim that $\langle G, s, d \rangle \in \text{HAMPATH}$, i.e., a hamiltonian path from s to d exists, can be verified in polynomial time if such path is provided as a *certificate*.

Note: all exponential time algorithms devised for PATH (*Thm. 7.14*) can be modified for the HAMPATH problem

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Definition 61

$$\text{COMPOSITES} = \{x \mid x = pq \text{ for integers } p, q > 1\}$$

Polynomial time verifiability:

Claim that $x \in \text{COMPOSITES}$ can be verified in polynomial time given a factor (p or q in above) is provided as a *certificate*.

In general, a **certificate** c can be any string that makes the verification process “easy”.

“We can verify that $w \in A$ as we were given also c ”



8960453
0100200

ICEMESSENGER

TÖL301G

E. Hyttiä

10:23

Hi! Did you know that 64738067 is a composite!

10:24

Hola! No, and I'm not going to waste my time

10:25

Try dividing it by 8039?

10:35

Ok, got it, $8039 \times 8053 \dots$ Thanks!

10:25

NP :)



Type to compose

Send

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Polynomially Verifiable

Definition 62 (Def. 7.18)

A verifier for a language A is an algorithm V , where

$$A = \{w \mid V \text{ accepts } \langle w \rangle \text{ for some string } c\}$$

We measure the time of a verifier only in terms of the length of w , so a polynomial time verifier runs in polynomial time in the length of w . A language A is polynomially verifiable if it has a polynomial time verifier.

- ▶ w describes a problem instance
(e.g., a composite number x)
- ▶ c is **the certificate**, that makes the verification easy (e.g., a factorization $x = pq$)

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Definition 63 (Def. 7.19)

NP is the class of languages that have polynomial time verifiers.

- ▶ The term NP comes from **nondeterministic polynomial time** and is derived from an alternative characterization by using nondeterministic polynomial time Turing machines.
- ▶ Problems in NP are sometimes called NP-problems
- ▶ Problems in P are obviously also in NP

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example

Example 91 (NTM for HAMPATH)

$N_1 = N_1(\langle G, s, d \rangle)$:

1. Write a list of m numbers, p_1, \dots, p_m , $m = |V|$
Each number in the list is nondeterministically selected to be between 1 and m .
2. Check for repetitions in the list. If any are found, *reject*
3. Check whether $s = p_1$ and $d = p_m$. If either fail, *reject*
4. For each i between 1 and $m - 1$, check whether $(p_i, p_{i+1}) \in E$. If any are not, *reject*. Otherwise, *accept*

Each stage runs in polynomial time. Thus N_1 runs in nondeterministic polynomial time.

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 92 (Thm. 7.20)

A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

Proof (sketch):

We show how to convert a polynomial time verifier to an equivalent polynomial time NTM and vice versa. The NTM simulates the verifier by guessing the certificate. The verifier simulates the NTM by using the accepting branch as the certificate. \square

The nondeterministic time complexity class $\text{NTIME}(t(n))$ is defined similarly as $\text{TIME}(t(n))$:

Definition 64 (Def. 7.21)

$$\text{NTIME}(t(n)) = \{L \mid L \text{ is a language decided by an } O(t(n)) \text{ time NTM}\}$$

Corollary 93 (Corollary 7.22)

$$NP = \bigcup_k \text{NTIME}(n_k).$$

- ▶ The class NP is *insensitive* to the choice of reasonable nondeterministic computational model
- ▶ All such models are polynomially equivalent

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0	Introduction
1	DFA
2	NFA
3	Regexps
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

- ▶ Each stage of an NP algorithm must have an obvious implementation in nondeterministic polynomial time on a reasonable nondeterministic computational model
- ▶ Show that every branch uses at most polynomially many stages

Definition 65

A **clique** in an undirected graph is a subgraph, wherein every two nodes are connected by an edge. Similarly, k -clique is a clique with k nodes.

The **k -clique problem** is to determine whether a graph contains a clique of size k :

Definition 66

$\text{CLIQUE} = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

A related problem is about finding the largest clique:

Definition 67 (max-clique problem)

Determine the largest clique in graph $G = (V, E)$.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 94 (Thm. 7.24)

CLIQUE is in NP.

Proof:

The following is a verifier V for CLIQUE.

$V = V(\langle G, k, c \rangle)$:

1. Test whether c is a subgraph with k nodes in G
2. Test whether G contains all edges connecting nodes in c
3. If both pass, accept; otherwise, reject



0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Definition 68 (0-1 Knapsack Problem)

Given n items with weights w_i and values v_i , choose a subset of items that maximizes the total value without exceeding a given maximum weight W .

Definition 69 (Subset-Sum Problem)

Given n items with weights w_i , is there a subset of items with a total weight of W .

Alternatively, is there a non-empty subset whose sum is zero?

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Time complexity of SUBSET-SUM

Definition 70 (SUBSET-SUM language)

$SUBSET\text{-}SUM = \{\langle S, t \rangle \mid S = \{x_1, \dots, x_k\},$
and for some $\{y_1, \dots, y_l\} \subseteq \{x_1, \dots, x_k\},$
we have $\sum y_i = t\}$

Theorem 95 (Thm. 7.25)

SUBSET-SUM is in NP

Proof:

(sketch) The subset is itself the certificate. □

NP class:

1. solvable in polynomial time on an NTM, or
2. membership can be verified in polynomial time
(on a TM)

P class:

1. solvable in polynomial time on a TM

In other words:

- ▶ **NP class:** membership can be **verified** quickly
- ▶ **P class:** membership can be **decided** quickly

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

Is $P = NP$?

- ▶ For example, *HAMPATH* and *CLIQUE* are in NP
 - ▶ Are they also in P?
- ▶ In fact, we do not know if $NP=P$ or $NP \neq P$

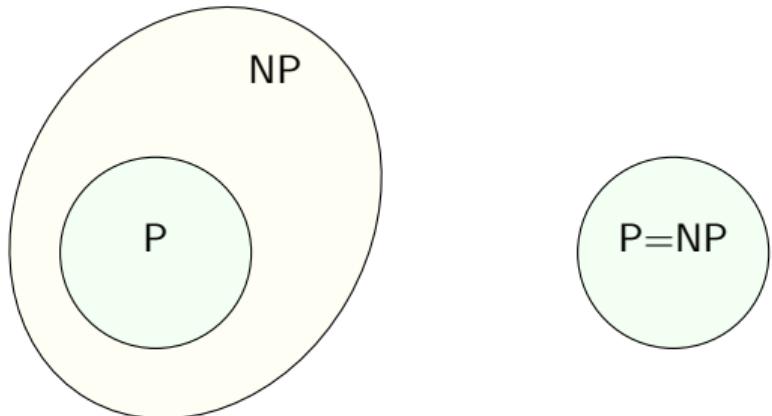


Figure: Two options, but only one can be true!

One of the greatest unsolved mysteries in CS!

Common belief is that $NP \neq P$

- ▶ Huge efforts invested to show the (in)equality
- ▶ ... but no success (yet!?)



Science

P \neq NP proof fails, Bonn boffin admits

Norbert Blum says his proposed solution doesn't work

By Thomas Claburn in San Francisco 31 Aug 2017 at 19:16 40 □ SHARE ▾



0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

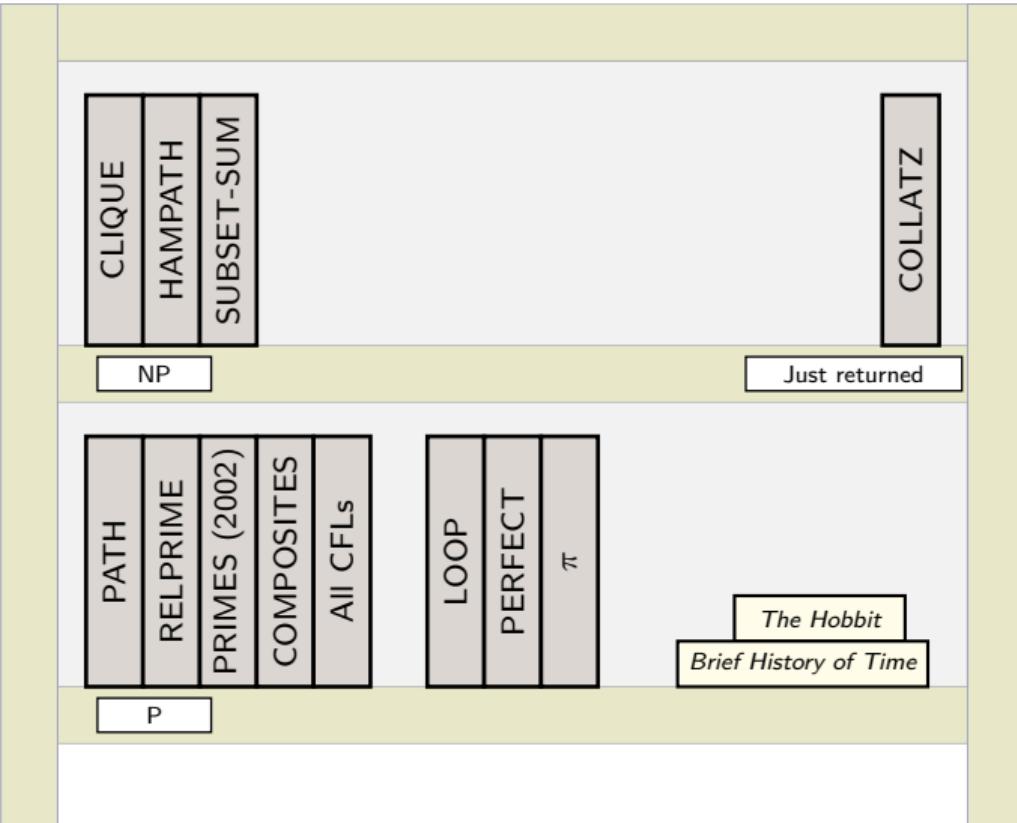
12 Brute Force,
SPACE and
Probabilities

13 Review

Great Library of Time Complexity

TÖL301G

E. Hyttä



Books in order! (if $N \neq NP$)

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

- ▶ Running time = *number of steps TM takes to decide*
- ▶ Asymptotic behavior, $n \rightarrow \infty$, is interesting
- ▶ Time Complexity Analysis of TM:
 - ▶ Condition on input length $|w| = n$
 - ▶ Derive an asymptotic upper bound for running time
 - ▶ Depends on computational model ...
- ▶ Class P problems: *decidable* in polynomial time
- ▶ Class NP problems: *verifiable* in polynomial time

Thanks!

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Lecture 11

NP-Complete Problems

Recap of the Previous Lecture

1. Time Complexity
 - ▶ Computational model still(!) Turing machines
2. Running time, worst-case analysis:

$$f(n) = \text{max. number of steps with } |w| = n$$

3. Big-O notation for asymptotic bounds
4. Polynomial bound desirable – exponential bad news
5. $\text{TIME}(t(n))$ = languages with $O(t(n))$ time complexity
6. Class P:
 - ▶ languages with a polynomial time complexity
 - ▶ any finite degree allowed
7. Class NP:
 - ▶ Polynomial time verifier, OR
 - ▶ Polynomial time decider with NTMs

Profound open question: Is $P = NP$?

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

This lecture

NP-complete problems:

- ▶ NP-complete problems are a subset in NP
 - ▶ In NP, so a polynomial time verifier exists
- ▶ Complexity of each NP-complete problem is related to that of **the entire class**:
“Solution to a such problem can be transformed in polynomial time to solve an arbitrary problem in NP”
- ▶ Important both theoretically and in practice

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

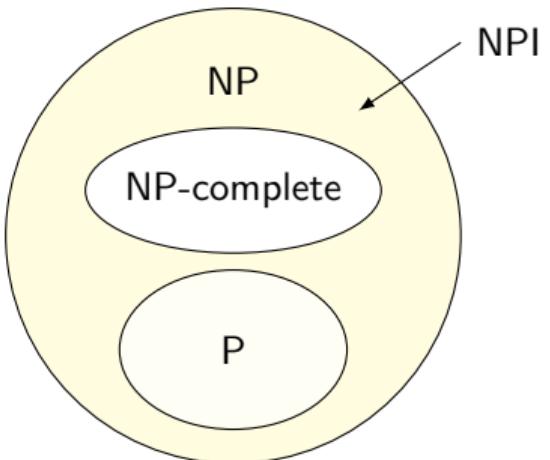
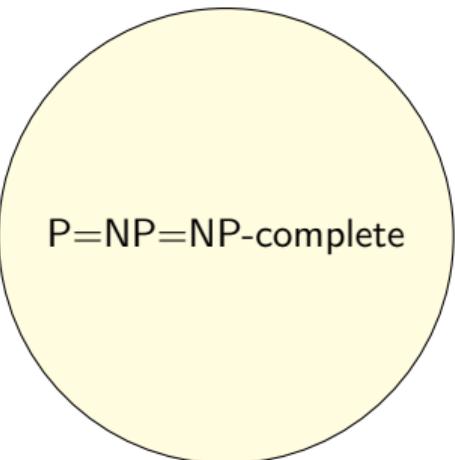
10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Two possible scenarios

(a) If $P \neq NP$ (b) If $P = NP$

Bets are accepted!

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Definitions

Definition 71 (Def. 7.28)

A function $f : \Sigma^* \rightarrow \Sigma^*$ is a **polynomial time computable function** if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w .

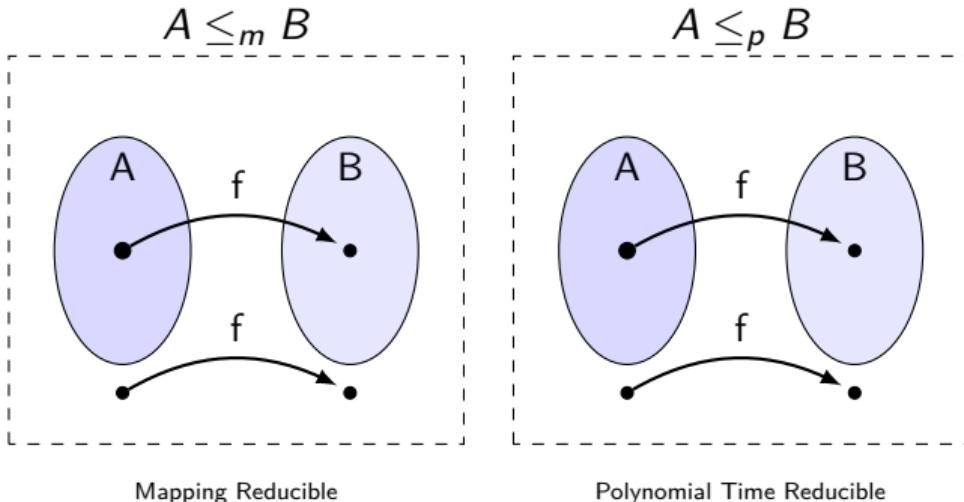
Definition 72 (Def. 7.29)

Language A is **polynomial time (mapping) reducible** to language B , written $A \leq_P B$, if a polynomial time computable function $f : \Sigma^* \rightarrow \Sigma^*$ exists, where for every w ,

$$w \in A \iff f(w) \in B.$$

The function f is called the **polynomial time reduction** of A to B .

Mapping reducible vs. polynomial time reducible



- ▶ Similar to mapping reducibility, section 5.3 (slide 291)
- ▶ Membership test: $w \in A$ iff $f(w) \in B$
- ▶ Latter just guarantees an *efficient* conversion

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Constructing a polynomial time reduction f

1. Choose a candidate f
2. Show that f is polynomial time computable
3. Show that f is a reduction:
 - ▶ If $x \in A$, then $f(x) \in B$
 - ▶ If $x \in A^c$, then $f(x) \in B^c$

Definition 73

$\text{HAMCYCLE} \triangleq \{\langle G \rangle \mid G \text{ is a directed graph with a hamiltonian cycle}\}$

Example 96

Show that $\text{HAMPATH} \leq_p \text{HAMCYCLE}$.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 97 (Thm 7.31)

If $A \leq_P B$ and $B \in P$, then $A \in P$.

Proof:

Let M be a polynomial time algorithm deciding B , and f be the polynomial time reduction from A to B . Then define a TM for A :

$N = N(w)$:

1. Compute $f(w)$
2. Run M on input $f(w)$ and output whatever M outputs.

Clearly,

- ▶ $w \in A \Leftrightarrow f(w) \in B$, as f was a reduction from A to B
- ▶ Thus, M accepts $f(w)$ whenever $w \in A$
- ▶ N runs in polynomial time as both stages do so



0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Boolean algebra and terminology

Variables:	x_1, x_2, x_3, \dots	TRUE or FALSE; or 1 or 0
Literals:	x or \bar{x}	Negation, $\bar{x} = \neg x$)
\vee	$a \vee b$	true, if a OR b is true
\wedge	$a \wedge b$	true, if both a AND b is true
Clauses:	$x_1 \vee x_2 \vee \bar{x}_3$	literals connected with \vee :s
cnf:	$(\dots) \wedge (\dots) \wedge \dots$	clauses connected with \wedge :s
3cnf:		all clauses have three literals

cnf=conjunctive normal form, AND, OR, NOT: \wedge, \vee, \neg

For example:

$$\phi = (\underbrace{\bar{x}_1 \vee x_2}_{\text{1st clause}}) \wedge (\underbrace{\bar{x}_3 \vee x_3 \vee \bar{x}_2}_{\text{2nd clause}})$$

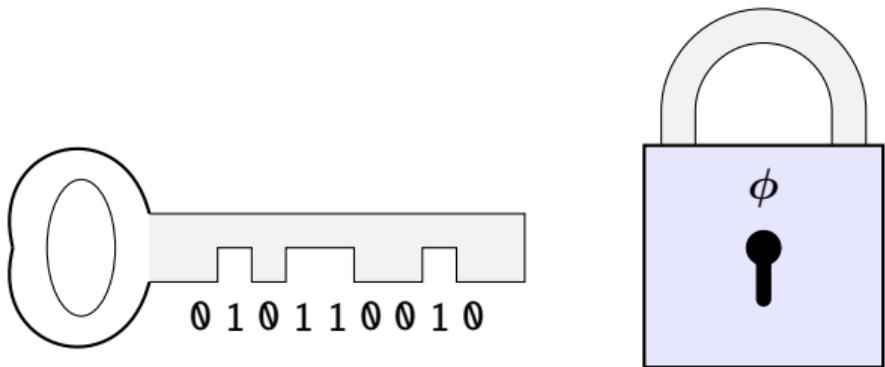
Definition 74

Boolean expression ϕ is satisfiable if it is TRUE with some x_i .

Satisfiability Problem

TÖL301G

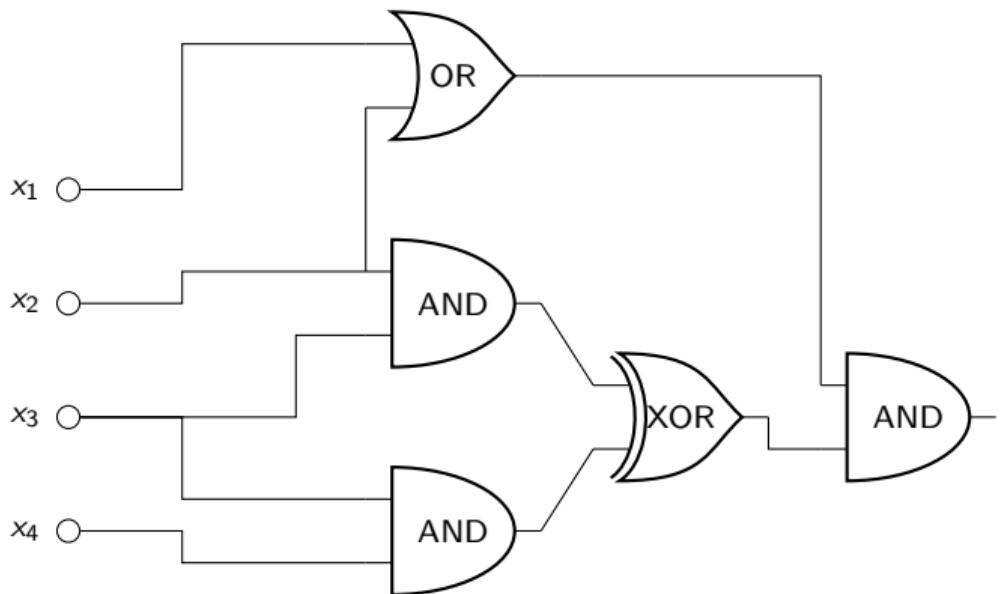
E. Hyttiä



“Does any of the 2^n keys open this lock?”

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review



Satisfiability Problems SAT and 3SAT

Definition 75

$$\text{SAT} = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Theorem 98 (Thm. 7.27)

$\text{SAT} \in P$ iff $P = NP$.

Proof omitted (in the book).

Definition 76

$$3\text{SAT} = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$$

Assignment satisfies a cnf-formula \Leftrightarrow each clause must contain at least one literal that evaluates to 1

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

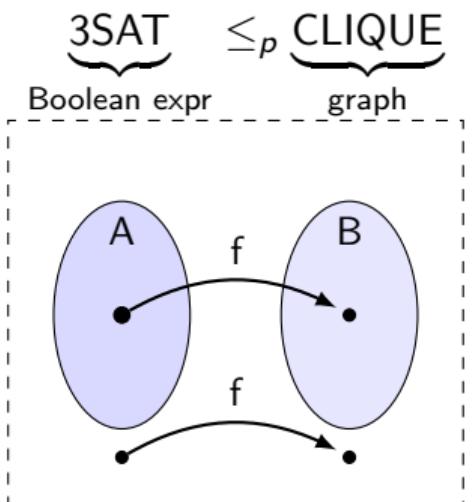
13 Review

From 3SAT to CLIQUE

Theorem 99 (Thm. 7.32)

3SAT is polynomial time reducible to *CLIQUE*.

That is, we need a function f that transforms *3SAT* problems to *CLIQUE* problems in polynomial time,



0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

From 3SAT to CLIQUE (2)

Proof:

Let Φ be a formula with k clauses such as

$$\Phi = \overbrace{(a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \dots \wedge (a_k \vee b_k \vee c_k)}^{\text{a clause}}$$

For every Φ , we construct a corresponding graph $G = (V, E)$ as follows.

Vertices: For each clause, add three nodes to G (a triple).

1. Each literal, e.g., a_1 and \bar{a}_1 corresponds to a node in G
2. In total k triples / clauses

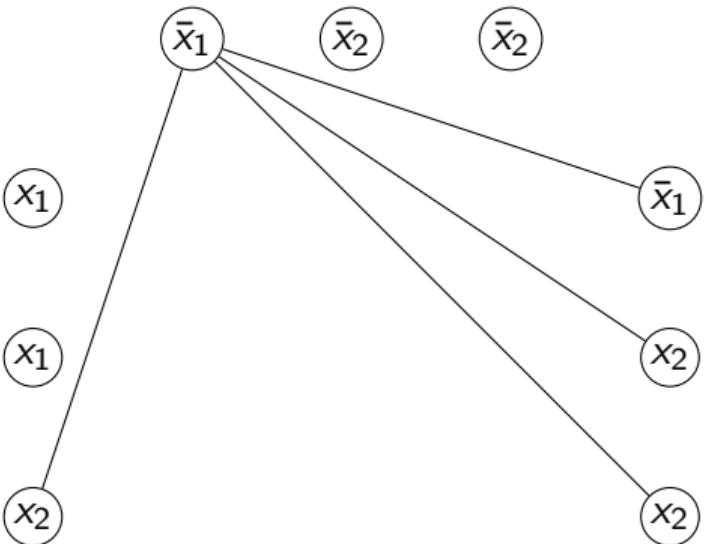
Edges: connect all nodes with an edge except:

1. Nodes in the same triple
2. Between two nodes with contradictory labels (e.g., a_1 and \bar{a}_1)

Then we seek a k -clique in the constructed graph G !

- ▶ If k -clique exists, it implies that Φ is satisfiable
- ▶ If Φ is satisfiable, the corresponding assignment includes a k -clique
(choose one set node from each triple $\Rightarrow k$ -nodes)





$$\Phi = (x_1 \vee x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_2)$$

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Definition 77 (Def. 7.34)

A language B is NP-complete if it satisfies two conditions:

1. B is in NP
2. Every A in NP is polynomial time reducible to B

Theorem 100 (Thm. 7.35)

If B is NP-complete and $B \in P$, then $P = NP$.

Proof:

Follows directly from the polynomial time reducibility. □

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

NP-completeness illustrated

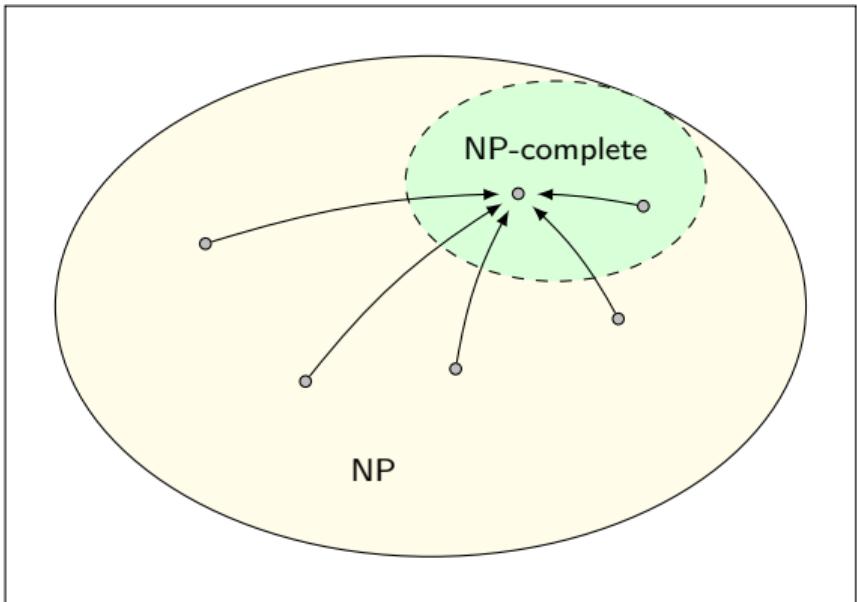


Figure: All problems in NP can be reduced to any NP-complete problem B in polynomial time!

A single hammer is enough . . . if it is NP-complete!

Theorem 101 (Thm. 7.36)

If B is NP-complete and $B \leq_p C$ for C in NP, then C is NP-complete.

Proof:

Let ptr refer to polynomial time reducible.

1. By def. C is in NP
2. Remains to show that all A are ptr to C

- ▶ B is NP-complete, so any A can be converted to B in pt
- ▶ If $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$
- ▶ That is, any A in NP is ptr to C

□

13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

CookLevin theorem

Theorem 102 (Thm. 7.37, CookLevin (1971))

SAT is NP-complete.

Note: This implies Theorem 7.27.

Proof:

Omitted (see the book), but consists of two parts.

1. Show that SAT is in NP (easy)
2. Show that any language A in NP is p_tr to SAT (hard)

(Idea: simulate a NTM with input w using Boolean expression)



Corollary 103 (Cor. 7.42)

3SAT is NP-complete.

Proof:

Omitted (see the book).



Vertex cover

Definition 78

A **vertex cover** in an undirected graph is such a subset of vertices that every edge is connected to at least one of them.

The **vertex cover problem** is to determine whether a graph has a k vertex cover:

Definition 79

$\text{VERTEX-COVER} = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k \text{ vertex cover}\}$

Theorem 104 (Thm. 7.44)

VERTEX-COVER is NP-complete.

Proof:

By reduction from 3SAT (see Fig. 7.45 from book). □

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Three coloring

Definition 80

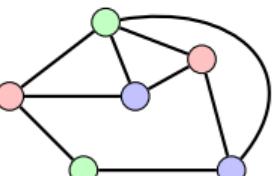
Vertex k coloring of an undirected graph G is an assignment of k colors to vertices in such a way that no neighboring vertices share the same color.

Definition 81

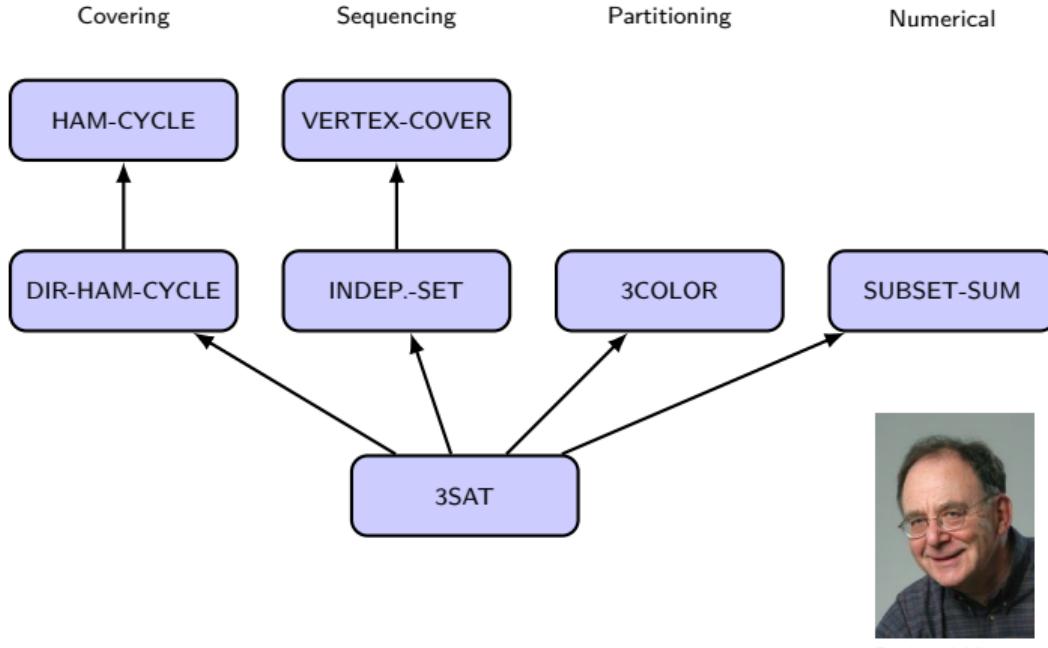
$3\text{COLOR} \triangleq \{\langle G \rangle \mid G \text{ is a 3-colorable undirected graph}\}$

A three colorable graph G :

$\Rightarrow \langle G \rangle \in 3\text{COLOR}$



Polynomial Time Reductions by Richard Karp



Richard Karp

R. Karp, *Reducibility Among Combinatorial Problems*, 1972.

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

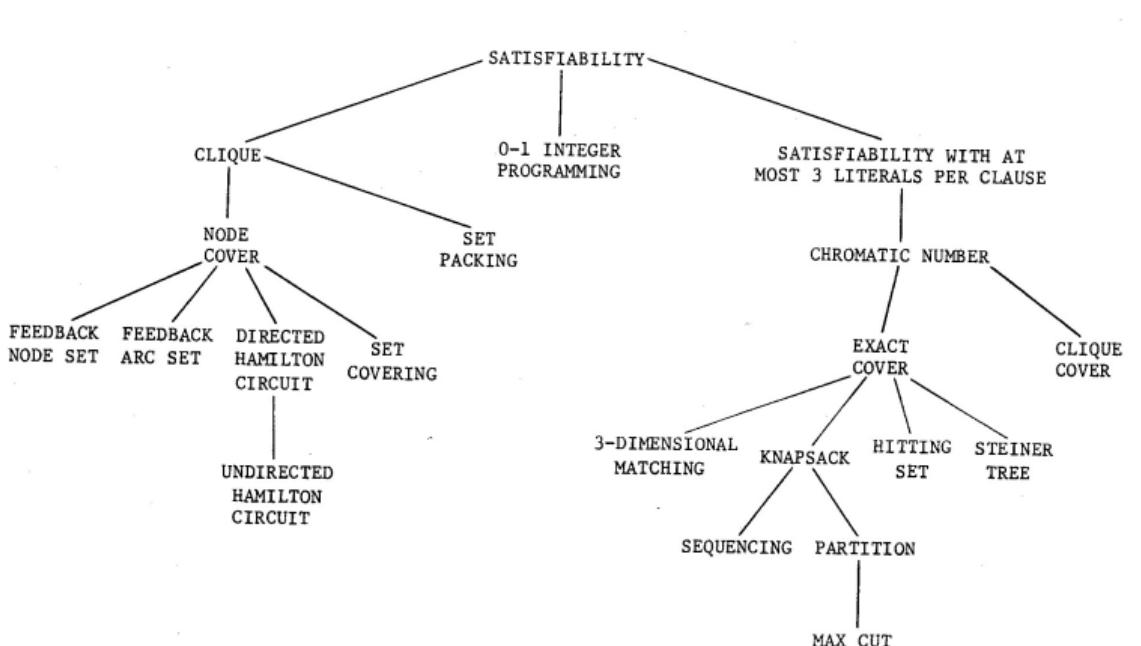
9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

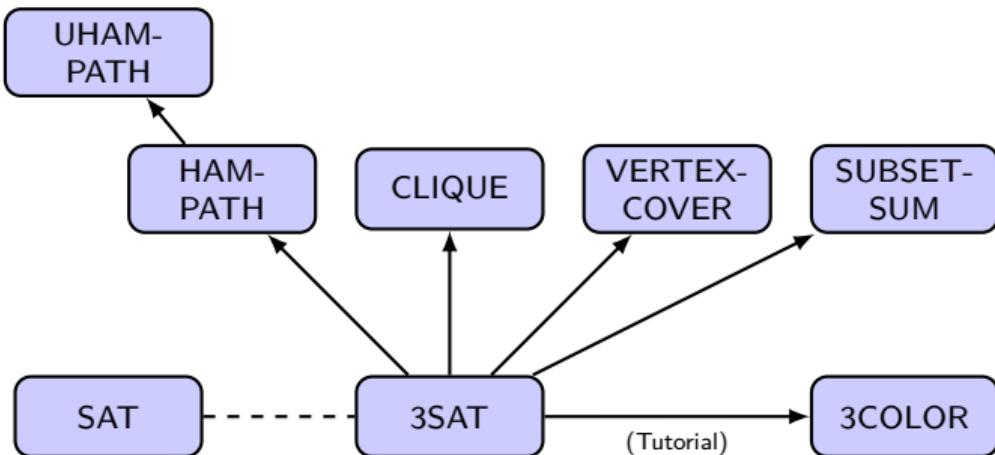


RICHARD M. KARP

FIGURE 1 - Complete Problems

0	Introduction
1	DFA
2	NFA
3	Regexps
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Our “Reduction Tree”



1. *HAMPATH* and *UHAMPATH* (undirected): Does G have a hamiltonian path from s to d
2. *CLIQUE*: Is there a size k clique in G
3. *VERTEX-COVER*: Is there a k -node vertex cover?
4. *SUBSET-SUM*:
Is there a (non-empty) subset of integers in A with sum k
5. *3COLOR*: Can G be colored with 3 colors?

- ▶ Problems in NP-complete are difficult
- ▶ Recall that B is NP-complete if
 1. B is in NP, and
 2. Any A in NP is polynomial time reducible to B
- ▶ If we give up (i), and require only (ii), we get the class of NP-hard problems:

Definition 82 (NP-hard)

A language B is NP-hard if every $A \in NP$ is polynomial time reducible to B .

Corollary 105

A language B is NP-complete if it is (i) NP and (ii) NP-hard.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Some NP-hard (function) Problems

TÖL301G

E. Hyttiä

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Decision:

- ▶ HALT; whether a TM halts with w or not

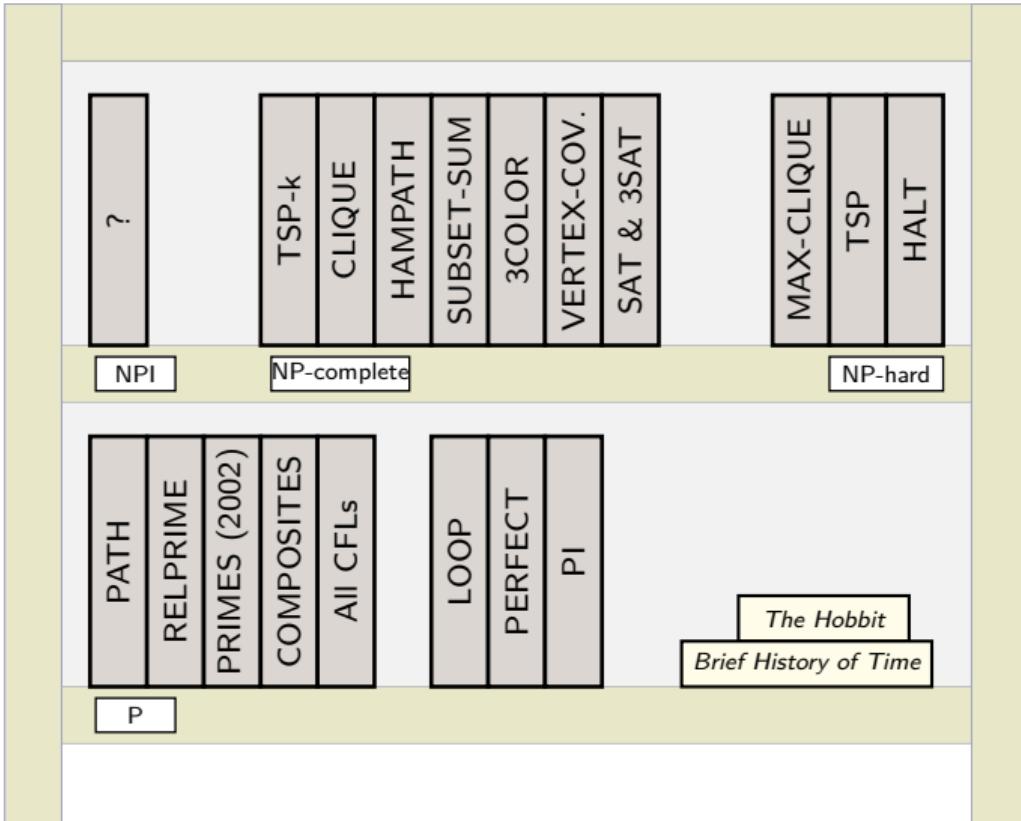
Function:

- ▶ TSP; Find the shortest circuit visiting every node
- ▶ Knapsack 0/1; Find the subset maximum value
- ▶ MAX-CLIQUE; Find the largest clique in graph

Common feature:

There is no obvious way to verify if a proposed solution is actually correct!

Great Library of Time Complexity (v.2)



Books in order! (if $N \neq NP$)

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

- ▶ NP-complete problems are in NP
- ▶ Any problem in NP reduces to such!
 - ▶ ... in polynomial time!
- ▶ A proof that one such problem is in P would be groundbreaking!
- ▶ NP-hard problems give up the requirement of being in NP itself.

Thanks!

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Lecture 12

Brute Force, SPACE and Probabilities

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

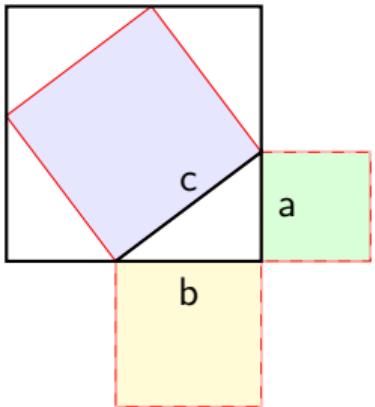
1. On Solving SATs

- ▶ Find a feasible assignment to boolean expression
- ▶ ... or does such exist? (SAT problem)

Pythagorean Equation: $a^2 + b^2 = c^2$

Pythagorean equation is

$$a^2 + b^2 = c^2 \quad (4)$$



It has infinitely many solutions:

$$3^2 + 4^2 = 5^2$$

$$6^2 + 8^2 = 10^2$$

$$5^2 + 12^2 = 13^2, \dots$$

Integer solutions are known as *Pythagorean Triples*: $\{3,4,5\}$, $\{6,8,10\}$, ...

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Similarly, equation

$$a + b = c$$

has infinitely many solutions when a, b, c are positive integers

Theorem 106 (Fermat's Last Theorem)

The equation

$$a^n + b^n = c^n,$$

has no solutions for positive integers a, b and c when $n > 2$.

Proved in 1994 by A. Wiles.

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Boolean Schur Triple Problem

Consider triples $\{a, b, c\}$ satisfying

$$a + b = c$$

where a, b, c are from $A = \{1, 2, \dots, n\}$

For example, with $n = 5$

$$1 + 2 = 3 \Rightarrow \{1, 2, 3\}$$

$$1 + 3 = 4 \Rightarrow \{1, 3, 4\}$$

$$1 + 4 = 5 \Rightarrow \{1, 4, 5\}$$

$$2 + 3 = 5 \Rightarrow \{2, 3, 5\}$$

Definition 83 (Boolean Schur triple problem)

Assign two colors to A so that no triple is monochromatic.

- ▶ Give a coloring that satisfies the constraints?
- ▶ Express as *SAT problem*?

Solving SAT with Mathematica

TÖL301G

E. Hyttiä

```
In[455]:= e1 = (x1 || x2 || x3) && (Not[x1] || Not[x2] || Not[x3]);  
e2 = (x1 || x3 || x4) && (Not[x1] || Not[x3] || Not[x4]);  
e3 = (x1 || x4 || x5) && (Not[x1] || Not[x4] || Not[x5]);  
e4 = (x2 || x3 || x5) && (Not[x2] || Not[x3] || Not[x5]);  
  
In[460]:= SatisfiableQ[e1 && e2 && e3 && e4, {x1, x2, x3, x4, x5}]  
Out[460]= True  
  
In[461]:= SatisfiabilityInstances[e1 && e2 && e3 && e4, {x1, x2, x3, x4, x5}]  
Out[461]= {{False, True, False, True, False}}
```

- ▶ With $n = 5$ we found a solution (see above!)
- ▶ With $n = 8$, coloring according to odd-even works (Check!)
- ▶ However, for $n = 9$ every coloring has at least one monochromatic triple
 - ▶ (We still need to show the unsatisfiability!)

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Solving SAT: Simplification rules

Pure literals: A literal is *pure* if it appears only in positive, or only in negative, form in the expression φ .

- ▶ Trivial to satisfy
- ▶ Simplification rule:
Remove all clauses from φ containing the pure literal

Unit propagation: A clause is a *unit clause* if it has only one *unassigned* literal ℓ and other literals are **False** (or removed)

- ▶ This literal must be assigned True in order to satisfy φ
- ▶ *Unit (Clause) Propagation (UCP) step:*
 1. Assign True value to the unassigned literal ℓ of the unit clause
 2. Remove clauses that become satisfied
 3. Remove literal $\bar{\ell}$ from φ
(It is False and thus cannot make any clause True)

(Recall the *related terminology*)

Solving SAT: DPLL-Algorithm

DPLL(φ) Backtracking Algorithm for SAT

Function DPLL(φ):

if All clauses satisfied **then**

return True

end if

if φ has an empty clause **then**

return False

end if

$\varphi \leftarrow$ Apply UCP for every unit clause

$\varphi \leftarrow$ Apply the pure literal rule for every pure literal

Choose a literal ℓ in φ {unassigned **branching variable**}

if DPLL($\varphi \wedge \ell$) **then**

return True

else

return DPLL($\varphi \wedge \neg \ell$)

end if

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Solving SAT: tuning DPLL

Performance depends on the choice of the branching literal $\ell \dots !$

MOMS is an example heuristic rule:¹¹

MOMS

- ▶ Let S be the set of clauses with minimal size
- ▶ Let $n(x_i, S)$ denote the frequency of variable x_i in S
- ▶ Let $Z = \arg \max_{x_i} \{n(x_i, S)\}$
- ▶ If $|Z| = 1$, choose that x_i
- ▶ Otherwise, consider the frequency of $x \in Z$ in clauses with one literal more, etc. (tie-breaking)

¹¹Maximum Occurrences in clauses of Minimal Size

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Boolean Schur's problem SAT encoding for $n = 9$

SAT encoding with $n = 9$:

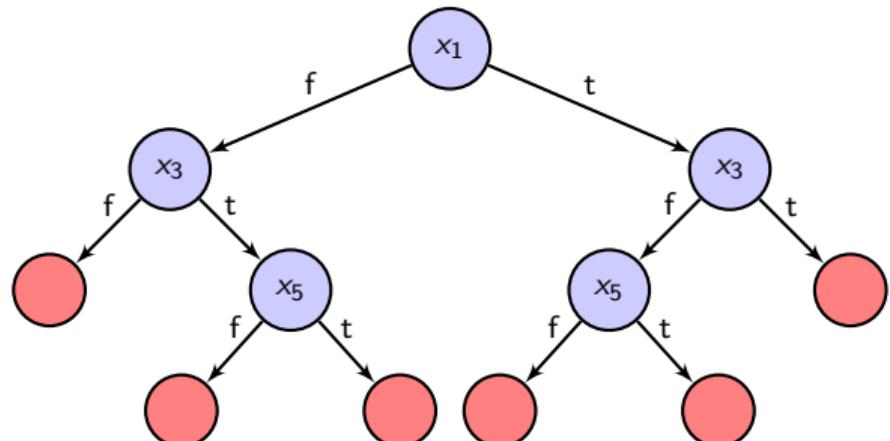
$$\begin{aligned} & (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4) \wedge \\ & (x_1 \vee x_4 \vee x_5) \wedge (\bar{x}_1 \vee \bar{x}_4 \vee \bar{x}_5) \wedge (x_2 \vee x_3 \vee x_5) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_5) \wedge \\ & (x_1 \vee x_5 \vee x_6) \wedge (\bar{x}_1 \vee \bar{x}_5 \vee \bar{x}_6) \wedge (x_2 \vee x_4 \vee x_6) \wedge (\bar{x}_2 \vee \bar{x}_4 \vee \bar{x}_6) \wedge \\ & (x_1 \vee x_6 \vee x_7) \wedge (\bar{x}_1 \vee \bar{x}_6 \vee \bar{x}_7) \wedge (x_2 \vee x_5 \vee x_7) \wedge (\bar{x}_2 \vee \bar{x}_5 \vee \bar{x}_7) \wedge \\ & (x_3 \vee x_4 \vee x_7) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_7) \wedge (x_1 \vee x_7 \vee x_8) \wedge (\bar{x}_1 \vee \bar{x}_7 \vee \bar{x}_8) \wedge \\ & (x_2 \vee x_6 \vee x_8) \wedge (\bar{x}_2 \vee \bar{x}_6 \vee \bar{x}_8) \wedge (x_3 \vee x_5 \vee x_8) \wedge (\bar{x}_3 \vee \bar{x}_5 \vee \bar{x}_8) \wedge \\ & (x_1 \vee x_8 \vee x_9) \wedge (\bar{x}_1 \vee \bar{x}_8 \vee \bar{x}_9) \wedge (x_2 \vee x_7 \vee x_9) \wedge (\bar{x}_2 \vee \bar{x}_7 \vee \bar{x}_9) \wedge \\ & (x_3 \vee x_6 \vee x_9) \wedge (\bar{x}_3 \vee \bar{x}_6 \vee \bar{x}_9) \wedge (x_4 \vee x_5 \vee x_9) \wedge (\bar{x}_4 \vee \bar{x}_5 \vee \bar{x}_9) \end{aligned}$$

There are 16 triples (solutions) in this case.

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Brute Force vs. Brute Reasoning

- ▶ Brute Force: with $n = 9$ gives $2^9 = 512$ cases
 - ▶ Doable as each check is fast, but ...
- ▶ *Brute Reasoning*
 - ▶ Add automated reasoning to Brute Force
 - ▶ E.g., by pruning the search tree:



- ▶ All leafs (red circles) generate a conflict \Rightarrow unsatisfiable

Boolean Pythagorean Triples Problem

Definition 84 (Pythagorean Triples Problem)

Is it possible to color each of the positive integers with two colors, so that no Pythagorean triple of integers a, b, c , satisfying $a^2 + b^2 = c^2$, are all the same color.

For example,

- ▶ If 3 and 4 are “red”, then 5 must be “blue”
- ▶ If 3 and 4 have a different color, then 5 can have any color

This can be stated as SAT problem ...

$$(x_3 \wedge \bar{x}_4) \vee (\bar{x}_3 \wedge x_4) \vee (x_3 \wedge \bar{x}_5) \vee \dots$$

which, unfortunately is in NP ...

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 107 (Heule et. al (2016))

The set $\{1, \dots, 7824\}$ can be partitioned into two parts, such that no part contains a Pythagorean triple, while this is impossible for $\{1, \dots, 7825\}$.

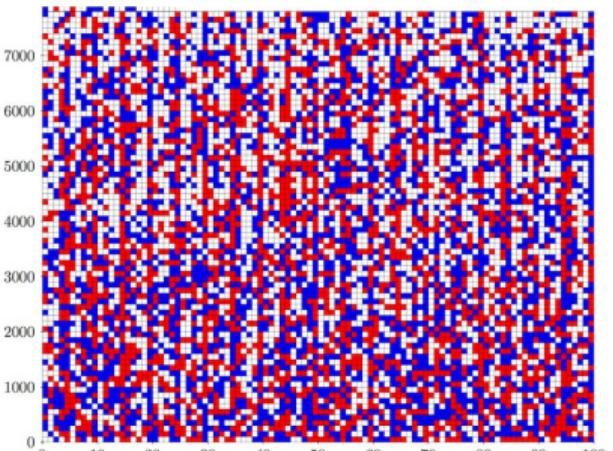


Figure: solution for $1, \dots, 7824$

See also the related youtube video:

<https://www.youtube.com/watch?v=9RoF-VuZMj8>

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

On solution approach

7825 does not sound too large ... right?

- ▶ However, the size of the proof is **200TB!**

High-level steps:

1. Problem stated as a SAT problem
 - ▶ Brute force: about $2^{7824} \approx 10^{2300}$ cases
2. Analytical reasoning
 - ▶ Add some “symmetries + number theory”
⇒ about 1 trillion cases
3. HPC and a powerful SAT solver
 - ▶ 2 days with 800 CPUs ... and Voilá!

(and another similar “independent” procedure to verify everything!)

Conjecture: No such coloring is possible even with $n = 3$ (or more) colors

Who's up to this task? :)

Enabling technology:

1. Availability of large HPC clusters
2. Powerful SAT solvers

The progress has been fast:

- ▶ Early 1990s: thousands of clauses
- ▶ Today: millions of clauses

There are downsides:

- ▶ Brute force approach yields little insight as for why?
- ▶ e.g., is 7825 some magical constant?

(perhaps there is still use for mathematicians?)

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Some other computer-assisted proofs

- ▶ The four-color theorem (1976, Appel & Haken)
 - ▶ Every planar map is 4-colorable
 - ▶ The first major theorem proved using a computer
- ▶ Erdös discrepancy problem (2015, Terence Tao)
 - ▶ For any infinite string of 1 and -1, counting only the numbers at a fixed interval for a finite number of steps gives an arbitrarily large (positive or negative) value.
- ▶ Optimal solutions to Rubik's cube, Sudoku . . .

- ▶ Brute Force (reasoning) approach can be useful
- ▶ ... especially when combined with appropriate analysis
- ▶ Enabling technology is developing fast
 - HW Pure distributed computing power available is enormous
 - SW Powerful SAT solvers are under constant development
- ▶ Creates possibilities to solve
 1. Famous combinatorial problems (test cases)
 2. Validate hardware designs
 3. Validate software
 4. Validate safety/security of complex systems
 5. ...

For interested minds: <http://www.satcompetition.org/>

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

2. Space complexity

1. So far we have neglected the amount of “memory” algorithm needs
That is, how long tape is needed!
2. Real computers have finite memory
⇒ memory requirements may also be of interest!



(Example devices from TÖL103M course).

Definition 85 (Def. 8.1)

Let M be a deterministic TM that halts on all inputs. The **space complexity** of M is the function $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of tape cells that M scans on any input of length n .

If M is a NTM wherein all branches halt on all inputs, we define the space complexity to be the maximum number of tape cells M scans on any branch for any input of length n .

We say M runs in space $f(n)$ if its space complexity is $f(n)$.

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Definition 86 (Def. 8.2)

Let $f : \mathbb{N} \rightarrow \mathbb{R}^+$ be a function. The **space complexity classes**, $\text{SPACE}(f(n))$, and $\text{NSPACE}(f(n))$, are defined as

$$\text{SPACE}(f(n)) = \{A \mid A \text{ is decided by an } O(f(n)) \text{ space TM}\}$$

$$\text{NSPACE}(f(n)) = \{A \mid A \text{ is decided by an } O(f(n)) \text{ space NTM}\}$$

Recall that *Big-O* was the “no more than” bound.

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example

Example 108

Recall that SAT is an NP-complete problem, where we are seeking a right binary key among the set of 2^m possible keys.

$M_1 : M_1(\Phi)$ (Φ is a Boolean expression):

1. For $i = 0, \dots, 2^m - 1$
 - ▶ If i “unlocks” Φ ; then halt on accept
- Halt on reject

M_1 runs in linear space, $O(n)$ as m cannot be more than the input length n .

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Theorem 109 (Savitch's theorem (Thm. 8.5))

For any function $f : \mathbb{N} \rightarrow \mathbb{R}^+$, where $f(n) \geq n$

$$NSPACE(f(n)) \subseteq SPACE(f^2(n)).$$

Note that this is quite different from the time complexity, where simulating a NTM with TM meant

$$2^{O(f(n))}$$

increase in the running time.

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

Definition 87 (PSPACE and NPSPACE)

PSPACE is the class of languages that are decidable in polynomial space

$$\text{PSPACE} = \bigcup_k \text{SPACE}(n^k)$$

and similarly for the NPSPACE.

How about P vs. PSPACE etc?

1. $P \subseteq \text{PSPACE}$ (tape can be accessed only linearly in time)
 $NP \subseteq \text{NPSPACE}$ (for the same reason)
2. Thus $NP \subseteq \text{PSPACE}$

It follows that

$$P \subseteq NP \subseteq \text{PSPACE} = \text{NPSPACE}$$

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

3. Probabilistic Turing Machines

Probabilistic Turing Machines:

- ▶ TMs execute a single branch deterministically
- ▶ NTMs: nondeterministic, massive parallelism, any number of branches
 - ▶ but all are suppose to halt if decider...
- ▶ Probabilistic algorithms “throw dice” or “coin”:
 - ▶ Machine learning, simulation,...

Idea: probabilistic TMs, a single branch and “coin-flips”

Why PTMs?

- ▶ Efficient computations for some problems
- ▶ Good at estimating (primes, Monte Carlo)

Definition 88 (Probabilistic Turing Machine (PTM))

Probabilistic TMs are like NTM, but

1. Single branch with two possible moves in every step
 - ▶ 50:50 probability, known as the coin-flip
 - ▶ independent events!
2. Outcome quantified using probabilities

$$\mathbf{P}\{\text{accept}\} = \sum_{b \in \mathcal{B}_A(w)} 2^{-k_b},$$

$$\mathbf{P}\{\text{reject}\} = \sum_{b \in \mathcal{B}_R(w)} 2^{-k_b},$$

where $\mathcal{B}_A(w)$ and $\mathcal{B}_R(w)$ denote the sets of branches with accepting and rejecting final state, and k_b is the number of steps in branch b .

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

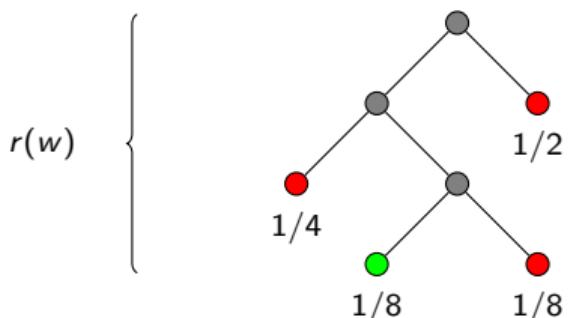
13 Review

Time complexity

Definition 89

Time complexity of PTM is according to the worst case of all branches.

Time complexity with PTMs is thus similar as with NTMs.



0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Definitions

Definition 90 (Probabilistic TM and Decidability)

Probabilistic TM M decides on language A with error probability ϵ iff

1. $P\{\text{accept}\} \geq 1 - \epsilon$ for all $w \in A$
2. $P\{\text{reject}\} \geq 1 - \epsilon$ for all $w \notin A$

Note: “two-sided errors” (both false positives and false negatives allowed)

Definition 91 (Class BPP)

Language $A \in BPP$ if there is a probabilistic polynomial time TM M (PPT-TM) that decides on language A with error probability $\epsilon = 1/3$.

“Maybe . . . NO!”

Definition 92

RP is the class of languages that are decided by PPT-TMs, where $w \in A$ are accepted with probability $\epsilon \leq 1/2$ and $w \notin A$ are rejected with probability 1.

- ▶ False positives are allowed
- ▶ but no false negatives!

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

- ▶ Each invocation of a TM \approx Bernoulli trial
independent samples!
- ▶ Repeated sampling reduces the probability of wrong conclusion
- ▶ We can reduce the error probabilities to arbitrary small
- ▶ These are known as *amplification lemmas*

"no matter how weak signal, we can get enough data (Big Data!?)"

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

1. Pseudo random numbers, “a free side product”
2. Monte Carlo estimation
3. Primality tests

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Theorem 110

If p is a prime, then for any integer $a \in \{1, \dots, p - 1\}$,

$$a^{p-1} \equiv 1 \pmod{p}. \quad (5)$$

- ▶ Notation: $\mathcal{Z}_p^+ = \{1, \dots, p - 1\}$
- ▶ Pseudoprimes: numbers $p > 1$ for which (5) holds for all $a \in \mathcal{Z}_p^+$
- ▶ Basis for the “pseudoprime” test: (*Fermat’s test*)
 - ▶ Check if $a^{p-1} \equiv 1 \pmod{p}$ for random a ’s (with $a \pmod{p} \neq 0$)
- ▶ Straightforward to implement as a probabilistic Turing machine

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Bad news: there are composite numbers, known as the Carmichael numbers, that also pass “Fermat’s test”:

$$\{m : m \text{ is a composite and } a^m \equiv a \pmod{m} \text{ for all integers } a\}.$$

Definition 93 (Square root with modular arithmetic)

x is a **square root** of a modulo p if $x^2 \equiv a \pmod{p}$.

Useful result for primality testing:

1. If p is prime, then 1 has exactly two roots: 1 and -1
2. If p is a Carmichael number, then 1 has at least four roots

This gives raise to “Carmichael test” that tightens the “sieve”

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

TM PRIME(p):

1. accept if $p = 2$ or $p = 3$; Otherwise reject if p is even
2. For k random a in $\{2, \dots, p - 2\}$
 - ▶ (Fermat's test): reject if $a^{p-1} \not\equiv 1 \pmod{p}$
 - ▶ (Carmichael test):
Let $p - 1 = s \cdot 2^\ell$, where s is odd
Compute $a^s, a^{2s}, a^{2^2s}, \dots, a^{2^\ell s}$ modulo p
If some element is not one, reject if the last such element is not -1
3. accept (all k tests passed)

A probabilistic Turing machine for a practical problem.

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Monte Carlo integration

Consider an n -dimensional integral

$$\int_V f(\mathbf{r}) d^n \mathbf{r},$$

where V is some subset of \mathbb{R}^n with volume $|V|$. For example, $f(\mathbf{r})$ could be an indicator function (is a condition met, or not),

$$f(\mathbf{r}) = \begin{cases} 1, & \text{system works} \\ 0, & \text{failure} \end{cases}$$

Numerical solution: take i.i.d. samples from V and compute their mean multiplied by $|V|$:

$$\int_V f(\mathbf{r}) d^n \mathbf{r} \approx |V| \cdot \frac{1}{n} \sum_{i=1}^n f(\mathbf{x}_i)$$

This is known as Monte Carlo integration

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

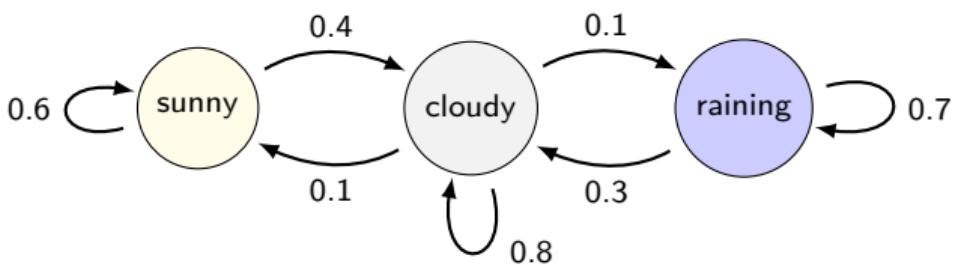
11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Example: Markov chains

Consider Markov chain model for weather forecasts:¹²



- ▶ 3 states and labels on edges are *transition probabilities*
- ▶ Suppose time step is 1 hour

Problem: It is raining at 9am. What is the probability that it is sunny at 1pm?

Construct a probabilistic TM that gives an answer?

¹²More about Markov processes in REI503M *Performance Analysis of Computer Systems*.

Example

Example 111

Consider language

$$A_\pi = \{w \mid w = a/b \text{ and } a/b < \pi\}.$$

where w is a string describing a fractional number a/b , and a and b are given, e.g., in binary. String $w \in A_\pi$ if $a/b < \pi$.

- ▶ Note that π is not a fractional number
- ▶ What kind of TM can decide on this?
- ▶ ... what if we do not mind of occasional errors?

(What kind of language is $B_\pi = \{w \in A_\pi \mid (a+1)/b \notin A_\pi\}$?)

0 Introduction
1 DFA
2 NFA
3 Regexp
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

0	Introduction
1	DFA
2	NFA
3	Regexp
4	Nonregular L
5	Context-Free
6	Pushdown
7	Turing
8	Decidability
9	Reducibility
10	Time
11	Complete
12	Brute Force, SPACE and Probabilities
13	Review

1. SPACE complexity and PSPACE class

- ▶ Similar to time complexity
- ▶ Characterize the memory requirement as a function of input length
- ▶ Relationship with other complexity classes:

$$P \subseteq NP \subseteq \text{PSPACE} = \text{NPSPACE}$$

2. Probabilistic TMs

- ▶ Each step involves also a coin flip
- ▶ Probabilistic algorithms (Monte Carlo simulation)
- ▶ Profound open question:

Is $P = BPP$?

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

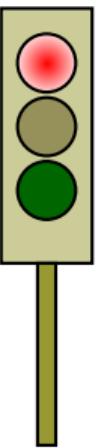
9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review



The End

Thank you!

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Lecture 13

Course review

Course summary: Part one

Part One: Automata and Languages

1. Prerequisites:

- ▶ Sets and their notation (\cup , \cap , \subseteq , \in)
- ▶ Symbols, strings, languages and classes of languages
- ▶ Mathematical tools:
 - ▶ Proof by construction
 - ▶ Proof by contradiction
 - ▶ Proof by *induction*

2. Finite automata: *DFA*, *NFA* and *REs*

- ▶ Class of regular languages
- ▶ Definitions and closures
- ▶ *Pumping lemma*

3. *Context-free Grammar (CFG)* and *Pushdown Automata (PDA)*

- ▶ Class of Context-free Languages (CFL)
- ▶ Definitions, stack, and pumping lemma

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Part Two: Computability theory

1. Turing machines and

- ▶ Infinite memory in form of *tape*
- ▶ Special halting states accept and reject
- ▶ Different variants of Turing machines
- ▶ *Church-Turing thesis:*

“Computable by an algorithm \Leftrightarrow Computable by a TM”

2. High level, implementation level and formal descriptions

3. Decidability

- ▶ Decidable or Undecidable (cf. A_{TM} and HALT_{TM})
- ▶ Recognizable or Unrecognizable

4. Reducibility

- ▶ Mapping reducibility, $A \leq_m B$, and computable functions

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Decidability results

TÖL301G

E. Hyttiä

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

		DFA	CFG	TM
Accept	A_x	✓	✓	✗
Empty	E_x	✓	✓	✗
Equivalent	EQ_x	✓	✗	✗
Halting	HALT_{TM}			✗
Regular L.	$\text{REGULAR}_{\text{TM}}$			✗

Table: Decidability results for DFAs, PDAs, and TMs

Part Three: Time Complexity

1. *Big-O* and *small-o* notation
2. *Class P and Class NP*
 - ▶ *Verifier* and certificates
3. *NP-completeness*
 - ▶ *Polynomial time reducibility*, $A \leq_p B$
 - ▶ NP-complete problems
 - ▶ How to proof that a language is NP-complete?

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Problem	context
SAT and 3SAT	boolean expressions
3COLOR	graph node coloring
CLIQUE	size k cliques in a graph
HAMPATH	Hamiltonian path from s to d
SUBSET-SUM	list of integers and their subsets

Table: Some NP-complete problems.

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force, SPACE and Probabilities
- 13 Review

1. SAT solvers
2. Space complexity
 - ▶ Similarly time complexity
 - ▶ $\text{PSPACE} = \cup_k \text{SPACE}(n^k)$
3. Probabilistic Turing Machines (PTMs)
 - ▶ Like NTMs, but single branch followed (in random)
 - ▶ Random choices referred to as “coin-flips”
 - ▶ Time complexity according to the longest branch
 - ▶ Accept and reject given as probabilities
 - ▶ Amplification lemmas

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force, SPACE and Probabilities
13 Review

Want more?

https://complexityzoo.uwaterloo.ca/Complexity_Zoo

Thank you and see you in the Exam!

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

INDEX

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

$0^n 1^n$, 330, 334, 337

\mathbb{Z}_p , 24

3COLOR problem, 405, 407

3SAT problem, 396, 402

3cnf, 392

4-color theorem, 429

A

... CFG, 253

... DFA, 244

... NFA, 246

... RE, 246

... TM, 260, 274

algorithm

... Church-Turing thesis, 232

... level of detail, 236

ALL

... CFG, 300

alphabet, 30

amplification lemma, 443

Big-O notation, 328

bijection, 27

Boolean expression, 392

Boolean Pythagorean Triples problem, 425

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

Boolean Schur Triple problem, 418

cardinality of a set, 262

Carmichael numbers, 446

certificate, 368

... examples, 365, 366

CFG, 138

... ambiguity, 152

... Chomsky normal form, 154

Chomsky normal form, 154, 253

Church-Turing thesis, 232

class BPP, 441

class NP-complete, 399

class NP-hard, 408

class NTIME, 372

class P, 347

class PSPACE, 436

class RP, 442

class TIME, 333, 347

clauses, 392

clique, 374, 403

CLIQUE problem, 374, 396

cnf, 392

coloring

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

... 3COLOR, 404
... 4-color theorem, 429
... graph, 404

complement, 21

COMPOSITES problem, 366

computable function, 301

... in polynomial time, 387

computation

... with DFA, 38

... with NFA, 55

... with PDA, 168

... with Turing machine, 209

concatenation

... of languages, 39

congruence classes, 24

context-free grammar, 138

... decidability, 252

Cook-Levin theorem, 402

countable set, 263

decidability, 218

... context-free grammar, 252

... with probabilistic TM, 441

Deterministic Finite Automata (DFA), 36

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

... computation, 38
diagonalization, 268
diophantine equation, 213
DPLL algorithm, 421

E (empty language)
... CFG, 254
... DFA, 247
... LBA, 299
... TM, 290
encoding, 237, 348
... unary, 308
enumerator, 228
EQ (equivalency)
... CFG, 255
... DFA, 249
... TM, 294
equivalence relation, 27
Euclidean algorithm, 359

Fermat's last theorem, 417
Fermat's little theorem, 445
finite automata
... deterministic, 36
four-color theorem, 429

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force,
SPACE and
Probabilities
13 Review

graph

- ... 3-coloring, 404
- ... 4-color theorem, 429
- ... clique, 374, 403
- ... Hamiltonian path, 365
- ... vertex coloring, 404
- ... vertex cover, 403

Halting problem HALT, 287

HAMCYCLE problem, 389

Hamiltonian path, 365

HAMPATH problem, 365, 389

iff, 85

induction proof, 110

inductive definition, 79, 80

injection, 27

intersection, 21

Knapsack problem, 376

languages

- ... classification, 257
- ... decidable, 218
- ... definition, 33
- ... equivalency, 294

E. Hyttiä

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

... regular, 38
... undecidable, 274, 287, 294
lexicographic order, 32
Linear Bounded Automaton (LBA), 295
... empty language, 299
literals, 392
... pure, 420
mapping reducibility, 303
Markov chain, 449
Mathematica, 419
maximum clique, 374
modular arithmetics, 24
modulo, 24
Monte Carlo integration, 448
multitape Turing machine, 223, 337, 339
natural numbers, 80
NFA, 53
nondeterministic
... finite automaton, 53
... Turing machine, 225
nonregular language
... pumping lemma, 116
NP-complete, 399

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force,
SPACE and
Probabilities
13 Review

... list of, 405, 407
... SAT, 402
NP-hard, 408
NTM, 225
number of Turing machines, 285
number system, 97
Pascal's triangle, 80
PATH problem, 349
PDA
... simplified, 179
pigeonhole principle, 115
polynomial time
... class P, 347
... computable function, 387
... reducible, 387
polynomially verifiable, 366, 368
precedence order, 76
primality test, 447
prime numbers, 354, 357
... relative primes, 358
... unique factorization theorem, 354
PRIMES problem, 357
probabilistic Turing machine, 438

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexp
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

proof

- ... by construction, 29
- ... by contradiction, 29
- ... by induction, 29, 110

proper set, 20

PTM, 438

public key cryptography, 357

Pumping lemma, 116, 193

pure literals, 420

pushdown automata (PDA), 167

Pythagorean equation, 416

Pythagorean Triples problem, 425

random walk, 351

reasonable encoding, 348

recursive definition, 80

reducibility, 286

... in polynomial time, 387

... mapping, 303

REGULAR, 292

regular expressions, 73

... definition, 77

... precedence order, 76

regular language, 38

E. Hyttiä

0 Introduction

1 DFA

2 NFA

3 Regexp

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review

regular operations, 39
... closures, 59
RELPRIME problem, 358
remainder, 23
reverse string, 32
Rice coding, 239
RSA algorithm, 355
Satisfiability problem
... 3SAT, 396
... Cook-Levin theorem, 402
... DPLL algorithm, 421
... NP-completeness, 402
... SAT, 395
Savitch's theorem, 435
Schur Triple problem, 418
sequence, 25
set
... proper set, 20
set minus, 21
simulator
... Turing machine, 210
small-o notation, 329
space complexity, 432

0 Introduction
1 DFA
2 NFA
3 Regexps
4 Nonregular L
5 Context-Free
6 Pushdown
7 Turing
8 Decidability
9 Reducibility
10 Time
11 Complete
12 Brute Force,
SPACE and
Probabilities
13 Review

stack, 167
star
... of languages, 39
string, 30
SUBSET-SUM problem, 376
sum of squares, 111
sum of three cubes, 213
surjection, 27
ternary numbers, 97
TIME, 333
TM
... decider, 218
tuple, 26
Turing complete, 302
Turing machine, 206
... computing, 209
... multitape, 223, 337, 339
... nondeterministic, 225
... number of, 285
... probabilistic, 438
... simulator, 210
... universal, 260, 261
Turing-decidable language, 218

- 0 Introduction
- 1 DFA
- 2 NFA
- 3 Regexps
- 4 Nonregular L
- 5 Context-Free
- 6 Pushdown
- 7 Turing
- 8 Decidability
- 9 Reducibility
- 10 Time
- 11 Complete
- 12 Brute Force,
SPACE and
Probabilities
- 13 Review

Turing-recognizable language, 218

unary encoding, 308

uncountable set, 268

undecidability, 288

undecidable language, 274, 287, 290, 293, 294

union, 21

... of languages, 39

unique factorization theorem, 354

unit propagation, 420

UTM, see Turing machine 261

verifier, 368

vertex coloring, 404, 429

vertex cover, 403

worst-case analysis, 322

0 Introduction

1 DFA

2 NFA

3 Regexps

4 Nonregular L

5 Context-Free

6 Pushdown

7 Turing

8 Decidability

9 Reducibility

10 Time

11 Complete

12 Brute Force,
SPACE and
Probabilities

13 Review