CS/ECE-374: Lecture 1

Lecturer: Nickvash Kani
Chat moderator: Samir Khan

January 26, 2021

University of Illinois at Urbana-Champaign

Course Administration

Instructional Staff

- Instructors:
  - Section A: Chandra Chekuri and Patrick Lin
  - Section B: Nickvash Kani and Yi Lu
- 11 Teaching Assistants
- 28 Undergraduate Course Assistants
- Office hours: See course webpage
- Contacting us: Use *private notes* on Piazza to reach course staff. Direct email only for sensitive or confidential information.

Section A vs B

Only lectures different for the sections.

Homework, exams, labs etc will be common.

Homework groups can be across sections.

Online resources

- Webpage: General information, announcements, homeworks, course policies
  `courses.engr.illinois.edu/cs374`
- Gradescope: Written homework submission and grading, regrade requests
- PrairieLearn: Quizzes, short question, autograded assessments
- Piazza: Announcements, online questions and discussion, contacting course staff (via private notes)
- Mediaspace/YouTube: Channels for videos

See course webpage for links

Important: check Piazza/course web page at least once each day

Prereqs and Resources

- Prerequisites: CS 173 (discrete math), CS 225 (data structures)
- Recommended books: (not required)
  - Introduction to Theory of Computation by Sipser
  - Introduction to Automata, Languages and Computation by Hopcroft, Motwani, Ullman
  - Algorithms by Dasgupta, Papadimitriou & Vazirani. Available online for free!
  - Algorithm Design by Kleinberg & Tardos
- Lecture notes/slides/pointers: available on course web-page
- Additional References
  - Lecture notes of Jeff Erickson, Sariel Har-Peled, Mahesh Viswanathan and others
  - Introduction to Algorithms: Cormen, Leiserson, Rivest, Stein.

Grading Policy: Overview

- Quizzes: 4%
- Homeworks: 24%
- Midterm exams: 42% ($2 \times 21\%$)
- Final exam: 30% (covers the full course content)

Midterm exam dates:

- Midterm 1: Mon, March 1, 6.30–9.30pm
- Midterm 2: Mon, April 12 , 6.30–9.30pm

No conflict exam offered unless you have a valid excuse.

Homework

- Quizzes, short self-graded questions on PrarieLearn: Due Monday, 10am.
  - Individually done and submitted.
- Written homework every week: Due on Wednesdays at 10am on *Gradescope*. Assigned at least a week in advance.
  - Written homeworks can be worked on in groups of up to 3 and each group submits *one* written solution (except Homework 0).
- Important: academic integrity policies. See course web page.

More on Homeworks

- No extensions or late homeworks accepted.
- To compensate, six problems in written homework will be dropped (corresponds to two whole home works). And two quizzes will be dropped.
- Important: Read homework faq/instructions on website.

Discussion Sessions/Labs

- 50min problem solving session led by TAs
- Two times a week
- Go to your assigned discussion section
- Bring pen and paper!

Advice

- Attend lectures, please ask plenty of questions.
- Attend discussion sessions.
- Don't skip homework and don't copy homework solutions. Each of you should think about *all* the problems on the home work - do not divide and conquer.
- Start homework early! Your mind needs time to think.
- Study regularly and keep up with the course.
- This is a course on problem solving. Solve as many as you can! Books/notes have plenty.
- This is also a course on providing rigourous proofs of correctness. Refresh your 173 background on proofs.
- Ask for help promptly. Make use of office hours/Piazza.

Homework 0

- HW 0 is posted on the class website. Quiz 0 available on Moodle.
- HW 0 due on Wednesady September 5th at 10am on Gradescope
- HW 0 to be done and submitted *individually*.

Miscellaneous

Please contact instructors if you need special accommodations.

Lectures are being taped. See course webpage.

High-Level Questions

- Computation, formally.
  - Is there a formal definition of a computer?
  - Is there a “universal” computer?
- Algorithms
  - What is an algorithm?
  - What is an *efficient* algorithm?
  - Some fundamental algorithms for basic problems
  - Broadly applicable techniques in algorithm design
- Limits of computation.
  - Are there tasks that our computers cannot do?
  - How do we prove lower bounds?
  - Some canonical hard problems.

Course Structure

Course divided into three parts:

- Basic automata theory: finite state machines, regular languages, hint of context free languages/grammars, Turing Machines
- Algorithms and algorithm design techniques
- Undecidability and NP-Completeness, reductions to prove intractability of problems

Basic

Alg

P/NP

Alg

Comp

Week	Tuesday Lecture	Tues/Wed Lab	Thursday Lecture	Thurs/Fri Lab
Jan 25-29	Administrivia and course goals Introduction and history; strings [Sariel's Videos, Lec 1]	String induction [Jeff's induction notes, Chandra's induction notes] [solutions]	Languages and regular expressions [Sariel's Videos, Lec 2]	Regular expressions [solutions]
Feb 1-5	DFAs: intuition, definitions, closure properties [Automata Tutor, JFLAP, Mahesh's DFA notes, Sariel's Videos, Lec 3]	DFA construction [solutions]	Non-Determinism, NFAs [Sariel's Videos, Lec 4]	DFA product construction [solutions]
Feb 8-12	Equivalence of DFAs, NFAs, and regular expressions [Sariel's Videos, Lec 5]	Regex to NFA to DFA (to Regex) [solutions]	Closure Properties: Language Transformations	Language Transformations [solutions]
Feb 15-19	Fooling Sets and Proving Non-Regularity [Mahesh's DFA notes, Fall 2015 TAs' Fooling Sets Notes, Sariel's Videos, Lec 6]	NO INSTRUCTION (Campus-wide break)	Beyond Regularity: CFGs, PDAs, Turing Machines [Sariel's Videos, Lec 7/8]	Proving Non-Regularity [solutions]
Feb 22-26	Universal Turing machines [Sariel's Videos, Lec 8]	Turing Machines [solutions]	*Optional review for Midterm 1*	*Optional review for Midterm 1*
Midterm 1 — Monday, March 1 18:30-21:30 Conflict exam: Tuesday, March 2 07:30-10:30				
Mar 1- 5	Reductions & Recursion [Sariel's Videos, Lec 10]	Hint: Binary search [solutions]	Divide and conquer: Selection, Karatsuba [Sariel's Videos, Lec 11]	Divide and Conquer [solutions]
Mar 8-12	Backtracking [Sariel's Videos, Lec 12]	Backtracking [solutions]	Dynamic programming [Sariel's Videos, Lec 13]	Dynamic programming [solutions]
Mar 15-19	More Dynamic programming [Sariel's Videos, Lec 14]	More Dynamic programming [solutions]	Graphs, Basic Search [Chandra's Graph notes, Sariel's Videos, Lec 15]	Graph Modeling [solutions] **Drop deadline**
Mar 22-26	Directed Graphs, DFS, DAGs and Topological Sort [Chandra's Graph notes, Sariel's Videos, Lec 16]	NO INSTRUCTION (Campus-wide break)	More Directed Graphs: DFS again, SCCs [Chandra's Graph notes, Sariel's Videos, Lec 16]	More Graph Modeling [solutions]
Mar 29-Apr 2	Shortest Paths: BFS and Dijkstra [Chandra's Graph notes, Sariel's Videos, Lec 17]	Shortest paths [solutions]	Shortest paths: Bellman-Ford, Dynamic Programming on DAGs [Chandra's Graph notes, Sariel's Videos, Lec 18]	More Shortest Paths [solutions]
Apr 5-9	Minimum Spanning Trees [Sariel's Videos, Lec 20]	Minimum Spanning Trees [solutions]	*Optional review for Midterm 2*	*Optional review for Midterm 2*
Midterm 2 — Monday, April 12 18:30-21:30 Conflict exam: Tuesday, April 13 07:30-10:30				
Apr 12-16	NO INSTRUCTION (Campus-wide break)	NO INSTRUCTION (Campus-wide break on Tues)	Reductions [Sariel's Videos, Lec 21]	Reductions [solutions]
Apr 19-23	NP and NP-Hardness [Sariel's Videos, Lec 22-24]	NP-hardness reductions [solutions]	More NP-Hardness [Sariel's Videos, Lec 23-24]	More NP-Hardness [solutions]
Apr 26-30	Undecidability [Sariel's Videos, Lec 9]	Undecidability reductions [solutions]	TBD **ICES Forms**	TBD [solutions] **TA ICES Forms**
May 3-7	Wrap-up, closing remarks *Optional review for Final Exam*	Optional Review for final exam	**Reading Day**	
Final exam — TBD Conflict exam: TBD				

Goals

- Algorithmic thinking
- Learn/remember some basic tricks, algorithms, problems, ideas
- Understand/appreciate limits of computation (intractability)
- Appreciate the importance of algorithms in computer science and beyond (engineering, mathematics, natural sciences, social sciences, ...)

Formal languages and complexity (The Blue Weeks!)

Why Languages?

First 5 weeks devoted to language theory.

Why Languages?

First 5 weeks devoted to language theory.

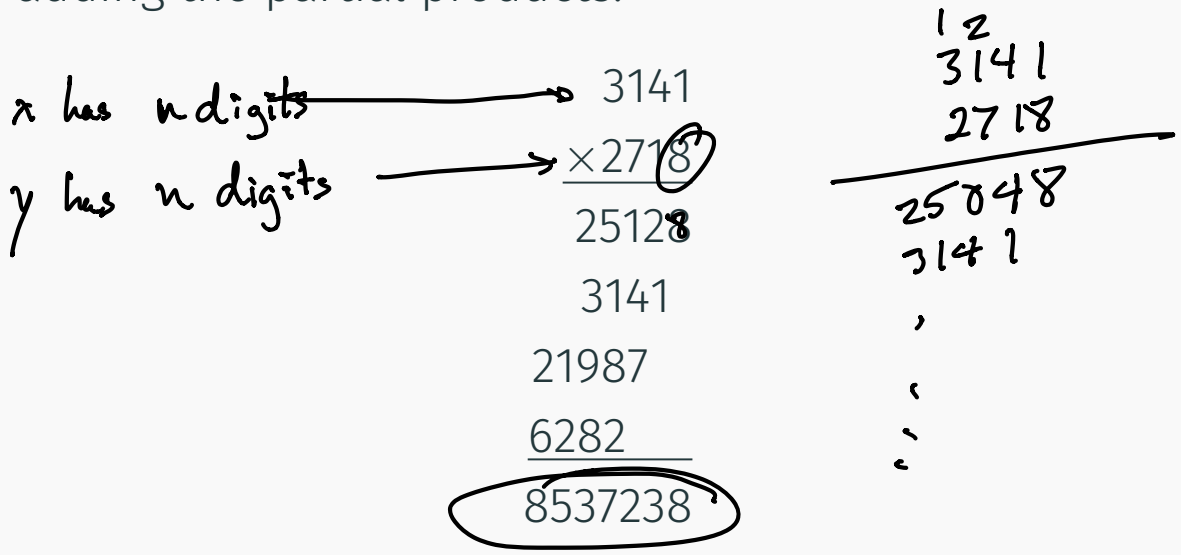
But why study languages?

Multiplying Numbers

Consider the following problem:

Problem Given two n-digit numbers x and y, compute their product.

Grade School Multiplication
Compute "partial product" by multiplying each digit of y with x and adding the partial products.



Time analysis of grade school multiplication

- Each partial product: $\Theta(n)$ time
- Number of partial products: $\leq n$
- Adding partial products: n additions each $\Theta(n)$ (Why?)
- Total time: $\Theta(n^2)$
- Is there a faster way?

Fast Multiplication

- $O(n^{1.58})$ time [Karatsuba 1960] disproving Kolmogorov's belief that $\Omega(n^2)$ is best possible
- $O(n \log n \log\log n)$ [Schonhage-Strassen 1971].
  Conjecture: $O(n \log n)$ time possible
- $O(n \log n \cdot 2^{O(\log^* n)})$ time [Furer 2008]
- $O(n \log n)$ [Harvey-van der Hoeven 2019]

Can we achieve $O(n)$? No lower bound beyond trivial one!

Equivalent Complexity

Does this mean multiplication is as complex as another problem that has a $O(n \log n)$ algorithm like sorting/QuickSort?

Equivalent Complexity

Does this mean multiplication is as complex as another problem that has a $O(n \log n)$ algorithm like sorting/QuickSort? How do we compare? The two problems have:

- Different inputs (two numbers vs n-element array)
- Different outputs (a number vs n-element array)
- Different entropy characteristics (from a information theory perspective)

Equivalent Complexity

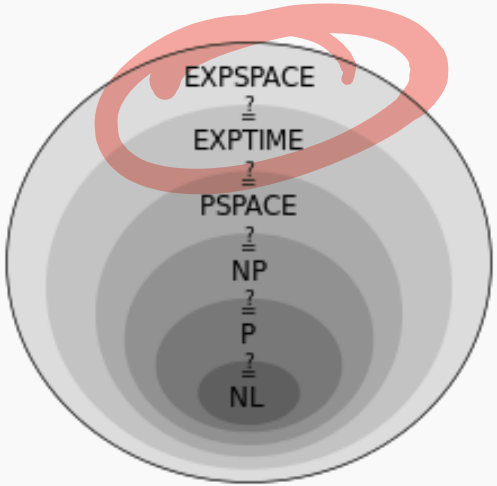
Does this mean multiplication is as complex as another problem that has a $O(n \log n)$ algorithm like sorting/QuickSort? How do we compare? The two problems have:

- Different inputs (two numbers vs n-element array)
- Different outputs (a number vs n-element array)
- Different entropy characteristics (from a information theory perspective)

Since multiplication has a $O(n \log n)$ algorithm, is it as complex as quicksort?

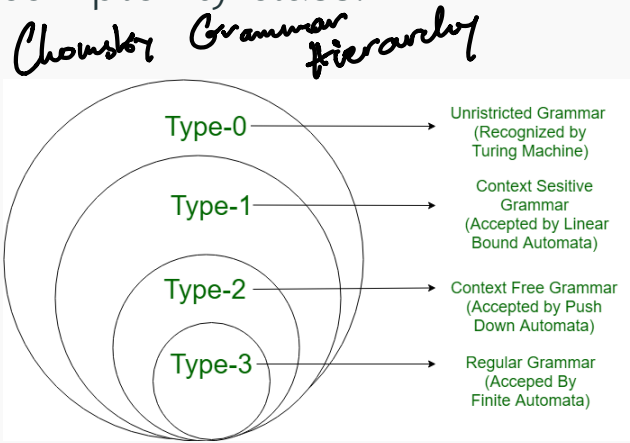
Languages, Problems and Algorithms … oh my! II

An algorithm has a runtime complexity.



Languages, Problems and Algorithms ... oh my! III

A problem has a complexity class!



Problems do not have run-time since a problem $\neq$ the algorithm used to solve it. *Complexity classes are defined differently.*

How do we compare problems? What if we just want to know if a problem is ”computable”.

Algorithms, Problems and Languages ... oh my! I

Definition

1. An algorithm is a step-by-step way to solve a problem.
2. A problem is some question that we'd like answered given some input. It should be a decision problem of the form "Does a given input fulfill property X."
3. A Language is a set of strings. Given a alphabet, Σ a language is a subset of Σ^*
   A language is a formal realization of this problem. For problem X, the corresponding language is:

   L = {w | w is the encoding of an input y to problem X and the answer to input y for a problem X is "YES" }
   A decision problem X is "YES" is the string is in the language.

Language of multiplication

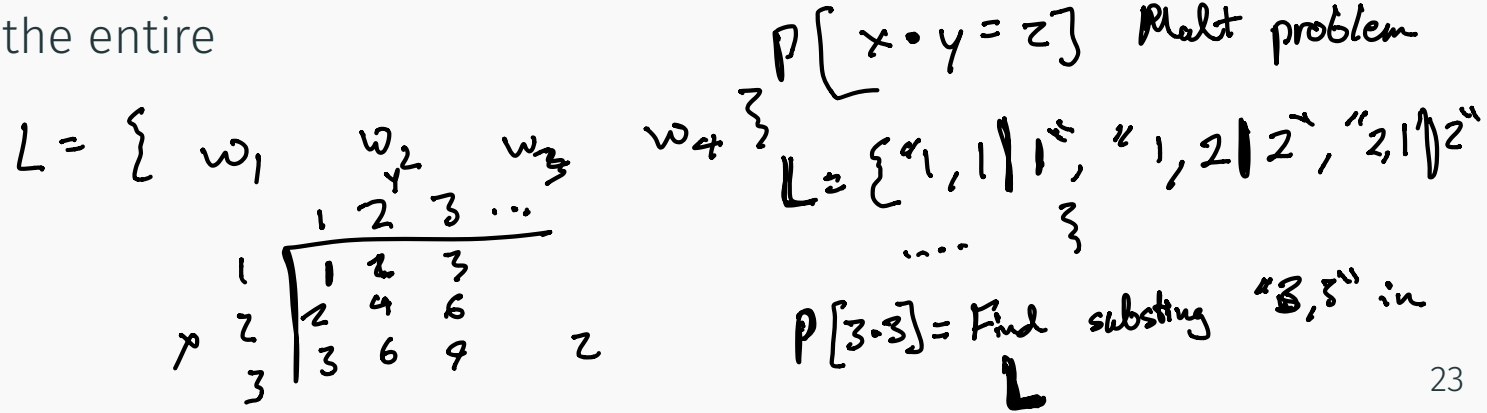
How do we define the multiplication problem as a language?

Define L as language where inputs are separated by comma and output is separated by |.

That way the language has all possible combination of inputs with their outputs.

Machine accepts a x*y=z if "x,y|z" is in L. Rejects otherwise.

Hence, $x \cdot y$ is computable cause you can just search through the entire



Consequences for Computation

Hows does this help?

$L_{MULT \cdot \mathbb{N}} = \{ \text{“}1,1 \| 1\text{”}, \text{“}2,1 \| 2\text{”}, \ldots, \text{“}5,4 \| 20\text{”}, \ldots \}$

$L_{SORT \cdot \mathbb{N}} = \{ \text{“}1,3,2,4 \| 1,2,3,4\text{”}, \ldots, \text{“}6,10,9 \| 6,9,10\text{”}, \ldots \}$

Consequences for Computation

Hows does this help? $L_{C_{prog}} = \{0, 1, 00, 10, 01, 11, 000, \ldots$

- **Question**: How many C programs are there? Countable many
- **Question**: How many languages are there? Uncountable many

Consequences for Computation

Hows does this help?

- **Question**: How many C programs are there?
- **Question**: How many languages are there?
- Hence some (in fact almost all!) languages/boolean functions do not have any C program to recognize them.

Questions:

Consequences for Computation

Hows does this help?

- **Question**: How many C programs are there?
- **Question**: How many languages are there?
- Hence some (in fact almost all!) languages/boolean functions do not have any C program to recognize them.

Questions:

- Maybe interesting languages/functions have C programs and hence computable. Only uninteresting languors uncomputable?
- Why should C programs be the definition of computability?
- Ok, there are difficult problems/languages. what languages are computable and which have efficient algorithms?

Strings

Alphabet

An alphabet is a **finite** set of symbols.

Examples of alphabets:

- $\Sigma = \{0, 1\}$,
- $\Sigma = \{a, b, c, \ldots, z\}$,
- ASCII.
- UTF8.
- $\Sigma = \{\langle \text{moveforward} \rangle, \langle \text{moveback} \rangle, \langle \text{moveleft} \rangle, \langle \text{moveright} \rangle\}$

String Definition

Definition

1. A string/word over Σ is a **finite sequence** of symbols over Σ. For example, '0101001', '*string*', '$\langle \text{moveback} \rangle \langle \text{rotate90} \rangle$'
2. $x \cdot y \equiv xy$ is the concatenation of two strings $abc \cdot cda = abccda$
3. The length of a string w (denoted by $|w|$) is the number of symbols in w. For example, $|101| = 3$, $|\epsilon| = 0$
4. For integer $n \geq 0$, Σ^n is set of all strings over Σ of length n. Σ^* is the set of all strings over Σ. $3^n = 3 \cdot 3 \cdot 3 \ldots$
5. Σ^+ set of all strings of all lengths including empty string.

Question: $\{'a', 'c'\}^* = \{\epsilon, a, c, aa, ac, ca, cc, \ldots\}$ $\{0,1\}^n = \{0,1\} \cdot \{0,1\} \cdot \{0,1\} \ldots$

$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$ $\Sigma^0 = \epsilon$ Σ^1 Σ^2

Emptiness

- ϵ is a string containing no symbols. It is not a set
- $\{\epsilon\}$ is a set containing one string: the empty string. It is a set, not a string.
- $\emptyset$ is the empty set. It contains no strings.

Question: What is $\{\emptyset\}$ = { {} }

Concatenation and properties

- If x and y are strings then xy denotes their concatenation.
- Concatenation defined recursively :
  - $xy = y$ if $x = \epsilon$
  - $xy = a(wy)$ if $x = aw$
- xy sometimes written as $x \bullet y$.
- concatenation is associative: $(uv)w = u(vw)$ hence write $uvw \equiv (uv)w = u(vw)$
- **not** commutative: uv not necessarily equal to vu
- The *identity* element is the empty string ϵ:

$$\epsilon u = u\epsilon = u.$$

Substrings, prefixes, Suffixes

Definition
v is substring of $w \iff$ there exist strings x, y such that $w = xvy$.

- If $x = \epsilon$ then v is a prefix of w
- If $y = \epsilon$ then v is a suffix of w

Substring

sub ubst

ub

Subsequence

A subsequence of a string $w[1...n]$ is either a subsequence of $w[2...n]$ or $w[1]$ followed by a subsequence of $w[2...n]$.

Example
kapa is a supsequence of *knapsack*

Subsequence

A subsequence of a string $w[1...n]$ is either a subsequence of $w[2...n]$ or $w[1]$ followed by a subsequence of $w[2...n]$.

Example
kapa is a supsequence of *knapsack*

Question: How many sub-sequences are there in a string $|w| = 5$? $= 32 = 2^{|w|}$ sacks

if there are no repeating characters

String exponent

$\{0,1\}^2 = 00, 01, 10, 11$

Definition
If w is a string then w^n is defined inductively as follows:

$w^n = \epsilon$ if $n = 0$

$w^n = ww^{n-1}$ if $n > 0$

Question: $(blah)^3 =$. blah blah blah

Set Concatenation

Definition
Given two sets X and Y of strings (over some common alphabet Σ) the concatenation of X and Y is

$$XY = \{xy \mid x \in X, y \in Y\} \tag{1}$$

Question: $X = \{fido, rover, spot\}$, $Y = \{fluffy, tabby\} \implies$
$XY =$ {fidofluffy, fidotabby,
roverfluffy, rovertabby, ...}

Σ^* and languages

Definition

1. Σ^n is the set of all strings of length n. Defined inductively:
   $\Sigma^n = \{\epsilon\}$ if $n = 0$
   $\Sigma^n = \Sigma\Sigma^{n-1}$ if $n > 0$
2. $\Sigma^* = \cup_{n \geq 0} \Sigma^n$ is the set of all finite length strings
3. $\Sigma^+ = \cup_{n \geq 1} \Sigma^n$ is the set of non-empty strings.

excludes $\Sigma^0 = \{\epsilon\}$

Definition
A language L is a set of strings over Σ. In other words $L \subseteq \Sigma^*$.

Question: Does Σ^* have strings of infinite length? No

Rapid-fire questions -strings

Answer the following questions taking $\Sigma = \{0, 1\}$.

1. What is Σ^0? $= \{\epsilon\}$
2. How many elements are there in Σ^n? $= 2^n$
3. If $|u| = 2$ and $|v| = 3$ then what is $|u \cdot v|$? $= 5$
4. Let u be an arbitrary string in Σ^*. What is ϵu? What is $u\epsilon$? $= u$

Induction on strings

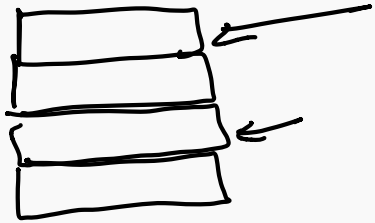
Inductive proofs on strings

Inductive proofs on strings and related problems follow inductive definitions.

Definition
The reverse w^R of a string w is defined as follows:

- $w^R = \epsilon$ if $w = \epsilon$
- $w^R = x^R a$ if $w = ax$ for some $a \in \Sigma$ and string x

If case n is true
then case n+1 must be true



Is top brick next on ground

Inductive proofs on strings

Inductive proofs on strings and related problems follow inductive definitions.

Definition
The reverse w^R of a string w is defined as follows:

- $w^R = \epsilon$ if $w = \epsilon$
- $w^R = x^R a$ if $w = ax$ for some $a \in \Sigma$ and string x

Theorem
Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Example: $(dog \cdot cat)^R = (cat)^R \cdot (dog)^R = tacgod$.

Principle of mathematical induction

Induction is a way to prove statements of the form $\forall n \geq 0, P(n)$ where $P(n)$ is a statement that holds for integer n.

Example: Prove that $\sum_{i=0}^{n} i = n(n+1)/2$ for all n.

Induction template:

- **Base case:** Prove $P(0)$
- **Induction hypothesis:** Let $k > 0$ be an **arbitrary** integer. Assume that $P(n)$ holds for any $n \leq k$.
- **Induction Step:** Prove that $P(n)$ holds, for $n = k + 1$.

Structured induction

- Unlike simple cases we are working with...
- ...induction proofs also work for more complicated "structures".
- Such as strings, tuples of strings, graphs etc.
- See class notes on induction for details.

Proving the theorem

Theorem
Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Proof: by induction.

On what?? $|uv| = |u| + |v|$?

$|u|$?

$|v|$?

What does it mean "induction on $|u|$"?

By induction on $|u|$

Theorem
Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Proof by induction on $|u|$ means that we are proving the following.

Base case: Let u be an arbitrary string of length 0. $u = \epsilon$ since there is only one such string. Then

$$(uv)^R = (\epsilon v)^R = v^R = v^R \epsilon = v^R \epsilon^R = v^R u^R$$

By induction on $|u|$

Theorem
Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Proof by induction on $|u|$ means that we are proving the following.

Base case: Let u be an arbitrary string of length 0. $u = \epsilon$ since there is only one such string. Then

$$(uv)^R = (\epsilon v)^R = v^R = v^R \epsilon = v^R \epsilon^R = v^R u^R$$

Induction hypothesis: $\forall n \geq 0$, for any string u of length n:

For all strings $v \in \Sigma^*$, $(uv)^R = v^R u^R$.

By induction on $|u|$

Theorem
Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Proof by induction on $|u|$ means that we are proving the following.

Base case: Let u be an arbitrary string of length 0. $u = \epsilon$ since there is only one such string. Then

$$(uv)^R = (\epsilon v)^R = v^R = v^R \epsilon = v^R \epsilon^R = v^R u^R$$

Induction hypothesis: $\forall n \geq 0$, for any string u of length n:

For all strings $v \in \Sigma^*$, $(uv)^R = v^R u^R$.

No assumption about v, hence statement holds for all $v \in \Sigma^*$.

Inductive step

- Let u be an arbitrary string of length $n > 0$. Assume inductive hypothesis holds for all strings w of length $< n$.
- Since $|u| = n > 0$ we have $u = ay$ for some string y with $|y| < n$ and $a \in \Sigma$.
- Then

Inductive step

- Let u be an arbitrary string of length $n > 0$. Assume inductive hypothesis holds for all strings w of length $< n$.
- Since $|u| = n > 0$ we have $u = ay$ for some string y with $|y| < n$ and $a \in \Sigma$.
- Then

$$(uv)^R \quad =$$

Inductive step

- Let u be an arbitrary string of length $n > 0$. Assume inductive hypothesis holds for all strings w of length $< n$.
- Since $|u| = n > 0$ we have $u = ay$ for some string y with $|y| < n$ and $a \in \Sigma$.
- Then

$$\begin{aligned}
(uv)^R &= ((ay)v)^R \\
&= (a(yv))^R \\
&= (yv)^R a^R \\
&= (v^R y^R) a^R \\
&= v^R (y^R a^R) \\
&= v^R (ay)^R \\
&= v^R u^R
\end{aligned}$$

Another example!

Theorem
Prove that for any strings x and y, $|xy| = |x| + |y|$

- Base Case: Assume $|x| = 0$, $x = \varepsilon$ i.e. $|x| = |\varepsilon| = 0$ by definition

  therefore $|x| + |y| = 0 + |y| = |\varepsilon y| = |y|$

- Inductive Case
  - Inductive Hypothesis (Strong induction)

    Suppose for $n \geq 0$, IH holds $|x| \leq n$, $|x| + |y| = |xy|$
  - Inductive step: need to prove that hypothesis holds for $|x| = n+1$

    Suppose $|x| = n+1$, goal $|x| + |y| = |xy|$

    Suppose $x = aw$ for some $a \in \Sigma$, $w \in \Sigma^*$: $1 + |w| = |x|$

    $xy = (aw)y = a(wy)$ we know $|wy| = |w| + |y|$ by IH

    $|awy| = 1 + |wy|$ by (1)

    $= 1 + |w| + |y|$ by inductive hypothesis

    $= |x| + |y|$

Languages

Languages

Definition
A language L is a set of strings over Σ. In other words $L \subseteq \Sigma^*$.

Languages

Definition
A language L is a set of strings over Σ. In other words $L \subseteq \Sigma^*$.

Standard set operations apply to languages.

- For languages A, B the concatenation of A, B is $AB = \{xy \mid x \in A, y \in B\}$.
- For languages A, B, their union is $A \cup B$, intersection is $A \cap B$, and difference is $A \setminus B$ (also written as $A - B$).
- For language $A \subseteq \Sigma^*$ the complement of A is $\bar{A} = \Sigma^* \setminus A$.

$A = \{dog\}$ $\bar{A}$ = all words except "dog"

Exponentiation, Kleene star etc

Definition
For a language $L \subseteq \Sigma^*$ and $n \in \mathbb{N}$, define L^n inductively as follows.

$$L^n = \begin{cases} \{\epsilon\} & \text{if } n = 0 \\ L \bullet (L^{n-1}) & \text{if } n > 0 \end{cases}$$

And define $L^* = \cup_{n \geq 0} L^n$, and $L^+ = \cup_{n \geq 1} L^n$

Rapid-Fire questions - Languages

Problem

Consider languages over $\Sigma = \{0, 1\}$.

1. *What is $\emptyset^0$?* $= \{\epsilon\}$
2. *If $|L| = 2$, then what is $|L^4|$?* $= 16$
3. *What is $\emptyset^*$, $\{\epsilon\}^*$, ϵ^*?* $= \{\epsilon\}$
4. *For what L is L^* finite?* $= L = \{\epsilon\}$ or $\emptyset$
5. *What is $\emptyset^+$, $\{\epsilon\}^+$, ϵ^+?*

Σ^0 = all strings of length 0 $= \{\epsilon\}$

$\emptyset^* = \emptyset^0 \cup \emptyset^1 \cup \emptyset^2 \cup \ldots$ $\quad \{\epsilon\} \quad \{\} \quad \{\}$

$\emptyset^+ = \emptyset^1 \cup \emptyset^2 \cup \ldots = \{\}$

$\{\epsilon\}^+ = \{\epsilon\}^1 \cup \{\epsilon\}^2 \cup \ldots = \{\epsilon\}$

$\epsilon^+ = \epsilon^1 \cup \epsilon^2 \cup \ldots = \{\epsilon\}$

$* = 1, 2, 3, 4 \ldots$ $\{\}$

$\epsilon^* = \bigcup_{n=0}^{\infty} \epsilon = \{\epsilon\}^0 \cup \{\epsilon\}^1 \cup \{\epsilon\}^2 \cup \epsilon^3 \cup \ldots = \{\epsilon\}$

$|L^2| = |L \cdot L| = |\{\text{"dog"}, \text{"eat"}\} \cdot \{\text{"dog"}, \text{"eat"}\}|$

$L = \{dog, eat\}$

Languages: easiest, easy, hard, really hard, really^n hard

- Regular languages.
  - Regular expressions.
  - DFA: Deterministic finite automata.
  - NFA: Non-deterministic finite automata.
  - Languages that are not regular.
- Context free languages (stack).
- Turing machines: Decidable languages.
- TM Undecidable/unrecognizable languages (halting theorem).