ECE-374-B: Lecture 1 - Logistics and Strings/Languages

Lecturer: Nickvash Kani

August 22, 2022

University of Illinois at Urbana Champaign

Course Administration

Instructional Staff

- Instructor:
  - **Nickvash Kani**
  - Andrew Miller
- Teaching Assistants:
  - Nitin Balachandran
  - Nicholas Bampton
  - Jinghan Huang
  - Sung Woo Jeon
  - Haochen Shen
  - Jason Zhu
- 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

This semester, the two sections will be run completely independently.

- Different lectures.
- Different homeworks, quizzes, exams.
- Different grading policies.

Section A vs B

This semester, the two sections will be run completely independently.

- Different lectures.
- Different homeworks, quizzes, exams.
- Different grading policies.

Section B will be in-person only.

Online resources

- Webpage: General information, announcements, homeworks, quizzes, course policies
  `https://canvas.illinois.edu/courses/30574`
- Gradescope: Written homework submission and grading, regrade requests
- Piazza: Announcements, online questions and discussion, contacting course staff (via private notes)

See course webpage for links

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

Grading Policy

Grading Policy: Overview

- Quizzes: 0%

- In contrast to previous semesters, quizzes will be ungraded. Use only for practice
- Approximately 20 quizzes (1 quiz/lecture)

Grading Policy: Overview

- Homeworks: 25%

- There will be approximately 9 HWs with 3 questions each.
- Hoemworks need to be submitted on Gradescope.
- Only the top 21 question grades will be considered for final grade calculation

Grading Policy: Overview

- Homeworks: 25%
- Midterm/Final exams: 75% ($3 \times 25\%$)

Exam dates:

- Midterm 1: Thurs, Sep 22, 12:30pm–1:45pm
- Midterm 2: Tues, Nov 1, 12:30pm–1:45pm
- Midterm 3: Thurs, Dec 1, 12:30pm–1:45pm
- Final: TBD

One exam will be dropped Drop policies should eliminate need for conflict exams.

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 <u>all</u> 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.

Miscellaneous

Please contact instructors if you need special accommodations.

Lectures are being taped (hopefully). The issue is that these recording systems are prone to failure. While I make no promises, I will try my best to record the lectures.

See course webpage for additional information.

Over-arching course questions

High-Level Questions

This course introduces three distinct fields of computer science research:

- Computational complexity.
  - Given infinite time and a certain machine, is it possible to solve a given problem.
- Algorithms
  - Given a deterministic Turing machine, how fast can we solve certain problems.
- Limits of computation.
  - Are there tasks that our computers cannot do and how do we identify these problems?

Why not just focus on Algorithms?

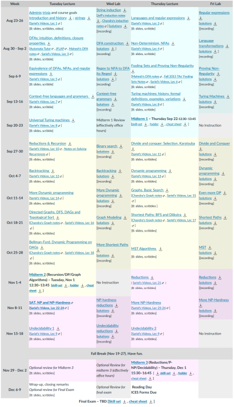
When someone asks you, ”How fast can you compute problem X”, they are actually asking:

- Is X solvable using the deterministic Turing machines we have at our disposal?
- If it is solvable, can we find the solution efficiently (in poly-time)?
- If it is solvable but we don’t have a poly time solution, what problem(s) is it most similar too?

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



Week	Tuesday Lecture	Wed Lab	Thursday Lecture	Fri Lab
Aug 23-26	Adminis trivia and course goals Introduction and history ; strings [Sariel's Videos, Lec 1] [B: slides, scribbles, video]	String induction [Jeff's induction notes , Chandra's induction notes] [solutions][recording]	Languages and regular expressions [Sariel's Videos, Lec 2] [B: slides, scribble]	Regular expressions [solutions][recording]
Aug 30 - Sep 2	DFAs: intuition, definitions, closure properties [Automata Tutor , JFLAP , Mahesh's DFA notes , Sariel's Videos, Lec 3] [B: slides, scribble]	DFA construction [solutions] [recording]	Non-Determinism, NFAs [Sariel's Videos, Lec 4] [B: slides, scribble]	Language transformations [solutions] [recording]
Sep 6-9	Equivalence of DFAs, NFAs, and regular expressions [Sariel's Videos, Lec 5] [B: slides, scribble]	Regex to NFA to DFA (to Regex) [solutions] [recording]	Fooling Sets and Proving Non-Regularity [Mahesh's DFA notes , Fall 2015 TAs' Fooling Sets Notes , Sariel's Videos, Lec 6] [B: slides, scribble]	Proving Non-Regularity [solutions][recording]
Sep 13-16	Context-free languages and grammars [Sariel's Videos, Lec 7] [B: slides, scribble]	Context-free grammars [solutions][recording]	Turing machines: history, formal definitions, examples, variations [Sariel's Videos, Lec 8] [B: slides, scribble]	Turing Machines [solutions][recording]
Sep 20-23	Universal Turing machines [Sariel's Videos, Lec 8] [B: slides, scribble]	Midterm 1 Review (effectively office hours)	**Midterm 1 – Thursday Sep 22** 12:30–13:45 [skill-set , fodder , cheat sheet]	No Instruction
Sep 27-30	Reductions & Recursion [Sariel's Videos, Lec 10 , Notes on Solving Recurrences] [B: slides, scribbles]	Binary search [solutions][recording]	Divide and conquer: Selection, Karatsuba [Sariel's Videos, Lec 11] [B: slides, scribbles]	Divide and Conquer [solutions][recording]
Oct 4-7	Backtracking [Sariel's Videos, Lec 12] [B: slides, scribbles]	Backtracking [solutions][recording]	Dynamic programming [Sariel's Videos, Lec 13] [B: slides, scribbles]	Dynamic programming [solutions][recording]
Oct 11-14	More Dynamic programming [Sariel's Videos, Lec 14] [B: slides, scribbles]	More Dynamic programming [solutions][recording]	Graphs, Basic Search [Chandra's Graph notes , Sariel's Videos, Lec 15] [B: slides, scribbles]	Even more DP [solutions][recording]
Oct 18-21	Directed Graphs, DFS, DAGs and Topological Sort [Chandra's Graph notes , Sariel's Videos, Lec 16] [B: slides, scribbles]	Graph Modeling [solutions][recording]	Shortest Paths: BFS and Dijkstra [Chandra's Graph notes , Sariel's Videos, Lec 17] [B: slides, scribbles]	Shortest Paths [solutions][recording]
Oct 25-28	Bellman-Ford, Dynamic Programming on DAGs [Chandra's Graph notes , Sariel's Videos, Lec 18] [B: slides, scribbles]	More Shortest Paths [solutions][recording]	MST Algorithms [B: slides, scribbles]	MST [solutions][recording]
Nov 1-4	**Midterm 2 (Recursion/DP/Graph Algorithms) – Tuesday, Nov 1 12:30–13:45** [skill-set , fodder , cheat sheet]	No Instruction	Reductions [Sariel's Videos, Lec 21] [B: slides, scribbles]	Reductions [solutions][recording]
Nov 8-11	SAT, NP and NP-Hardness [Sariel's Videos, Lec 22-24] [B: slides, scribbles]	NP-hardness reductions [solutions][recording]	More NP-Hardness [Sariel's Videos, Lec 23-24] [B: slides, scribbles]	More NP-Hardness [solutions][recording]
Nov 15-18	Undecidability 1 [Sariel's Videos, Lec 9] [B: slides, scribbles]	Undecidability reductions [solutions][recording]	Undecidability 2 [Sariel's Videos, Lec 9] [B: slides, scribbles]	No instruction
	Fall Break (Nov 19-27). Have fun.			
Nov 29 - Dec 2	*Optional review for Midterm 3* [B: slides, scribbles]	*Optional Review for midterm 3 (effectively office hours)*	**Midterm 3 (Reductions/P-NP/Decidability) – Thursday, Dec 1 15:30–16:45** [skill-set , fodder , cheat sheet]	
Dec 6-9	Wrap-up, closing remarks *Optional review for Final Exam* [B: slides, scribbles]	*Optional Review for final exam*	**Reading Day ICES Forms Due**	

Final Exam – TBD [Skill set , cheat sheet]

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?

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.

$$
\begin{array}{r}
3141 \\
\underline{\times 2718} \\
25128 \\
3141 \\
21987 \\
\underline{6282} \\
8537238
\end{array}
$$

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?

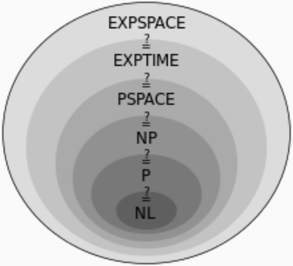
Equivalent Complexity

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)

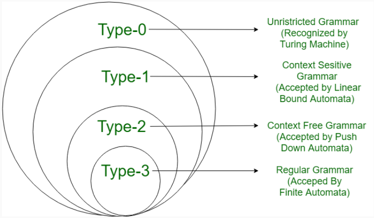
Languages, Problems and Algorithms … oh my! II

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^*

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 |.

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

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 |.

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

$$L_{MULT2} = \left\{ \begin{matrix} 1 \times 1|1, & 1 \times 2|2, & 1 \times 3|3, \ldots \\ 2 \times 1|2, & 2 \times 2|4, & 2 \times 3|6, \ldots \\ \vdots & \vdots & \vdots \\ n \times 1|n, & n \times 2|2n, & n \times 3|3n, \ldots \end{matrix} \right\} \tag{1}$$

Language of sorting

We do the same thing for sorting.

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

Machine accepts a $[i_1, i_2, \ldots] = sort(\{i_1, i_2, \ldots\})$ if "x[]|z[]" is in L. Rejects otherwise.

Language of sorting

We do the same thing for sorting.

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

Machine accepts a $[i_1, i_2, \ldots] = sort(\{i_1, i_2, \ldots\})$ if "x[]|z[]" is in L. Rejects otherwise.

$$L_{Sort2} = \begin{Bmatrix} 1,1|1,1 & 1,2|1,2 & 1,3|1,3,\ldots \\ 2,1|1,2, & 2,2|2,2, & 2,3|2,3,\ldots \\ \vdots & \vdots & \vdots \\ n,1|1,n, & n,2|2,n, & n,3|3,n,\ldots \end{Bmatrix} \tag{2}$$

Language of sorting

We do the same thing for sorting.

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

Machine accepts a $[i_1, i_2, \ldots] = sort(\{i_1, i_2, \ldots\})$ if "x[]|z[]" is in L. Rejects otherwise.

$$L_{Sort2} = \begin{Bmatrix} 1,1|1,1 & 1,2|1,2 & 1,3|1,3,\ldots \\ 2,1|1,2, & 2,2|2,2, & 2,3|2,3,\ldots \\ \vdots & \vdots & \vdots \\ n,1|1,n, & n,2|2,n, & n,3|3,n,\ldots \end{Bmatrix} \tag{2}$$

If the same type of machine can recognize both languages, then that gives us an upperbound top their hardness.

How do we formulate languages?

Strings

Alphabet

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

Examples of alphabets:

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

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
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.
5. Σ^* set of all strings of all lengths including empty string.

Question: $\{'a','c'\}^* =$

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

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 $|w| = 5$?

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: $(blah)^3 =$.

Rapid-fire questions -strings

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

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

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 $\bar{A} = \Sigma^* \setminus A$.

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

$$XY = \{xy \mid x \in X, y \in Y\} \tag{3}$$

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

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.

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?

Rapid-Fire questions - Languages

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

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

Terminology Review

Let's review what we learned.

- A character(a, b, c, x) is a unit of information represented by a symbol: (letters, digits, whitespace)
- A alphabet(Σ) is a set of characters
- A string(w) is a sequence of characters
- A language(A, B, C, L) is a set of strings

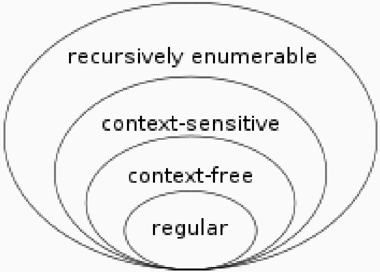
Terminology Review

Let's review what we learned.

- A character(a, b, c, x) is a unit of information represented by a symbol: (letters, digits, whitespace)
- A alphabet(Σ) is a set of characters
- A string(w) is a sequence of characters
- A language(A, B, C, L) is a set of strings
- A grammar(G) is a set of rules that defines the strings that belong to a language

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).



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

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

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

$|xy| = |\epsilon y| = |y| = |y| + |\epsilon| = |y| + |x| = |x| + |y|$

Another example!

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

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

$|xy| = |\epsilon y| = |y| = |y| + |\epsilon| = |y| + |x| = |x| + |y|$

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

For all strings $y \in \Sigma^*$, $|xy| = |x| + |y|$.

Another example!

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

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

$|xy| = |\epsilon y| = |y| = |y| + |\epsilon| = |y| + |x| = |x| + |y|$

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

For all strings $y \in \Sigma^*$, $|xy| = |x| + |y|$.

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

Another Example: Inductive step

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

Another Example: Inductive step

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

$$
\begin{aligned}
|xy| &= |(az)y| \\
&= |a(zy)| \\
&= 1 + |zy| && \text{recursive def of string length} \\
&= 1 + |z| + |y| && \text{inductive hypothesis} \\
&= (1 + |z|) + |y| \\
&= |az| + |y| && \text{recursive def of string length} \\
&= |x| + |y|
\end{aligned}
$$