Preliminaries

COMP 455 – 002, Spring 2019

Strings

Letters, digits, \$, *etc.*

- **String**: A *finite* sequence of symbols

This is important!

- Let w denote a string
  - $|w|$ = the length of string w
    - If w = abcd, then $|w| = 4$
- Let ε denote the empty string
  - $|\varepsilon| = 0$

Strings

Given a string w:

- **Prefix**: The first 0 up to $|w|$ characters in w
  - If $w = \text{abc}$, then w's prefixes are $\boxed{\varepsilon, \text{a}, \text{ab},}$ and abc

"Proper prefixes"

Strings

- **Concatenation**: If w and x are strings, then wx is a string.
  - Example: If $w = \text{abc}$ and $x = \text{def}$, then $wx = \text{abcdef}$
  - $\varepsilon w = w\varepsilon = w$

Alphabets

- **Alphabet**: A *finite* set of symbols

Languages

- **Language**: A set of strings over some alphabet
  - Example: L is the language of strings containing an equal number of 0s and 1s
    - L is defined for the *alphabet* $\{0, 1\}$
    - The strings ε, 01, and 0110, *etc.* are in L
    - 1, 0, 11, and 101, *etc.* are not in L

More Language Examples

- ∅ is the language consisting of no strings
- {ε} is the language consisting only of ε
- {ε, 0, 1, 010} (a *finite* language)
- {ε, 0, 00, 000, 0000, …} (an *infinite* language)
- All files denoting legal C programs
- All legal English sentences
  - This is *really* hard to formally define. Example: Is "My car ate my shoe." a valid sentence?

"*" Notation

- If Σ is an alphabet, then Σ^* is the set of all strings over Σ.
  - Example: If $\Sigma = \{0\}$, then $\Sigma^* = \{\varepsilon, 0, 00, 000, \ldots\}$
  - If $\Sigma = \{0, 1\}$, then $\Sigma^* = \{\varepsilon, 0, 1, 10, 11, 100, \ldots\}$

The Big Picture

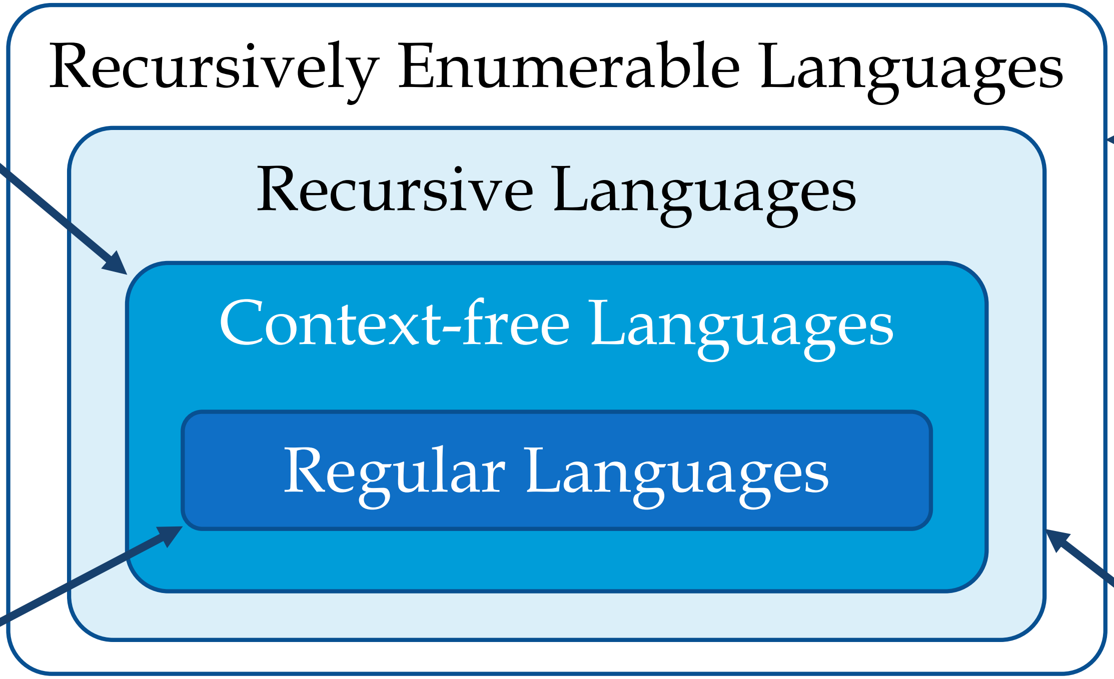
- In this class we will study *classes* of languages

...But what does that have to do with *computing*?

Language Classes We Cover

Useful for parsing.
- Pushdown automata
- Context-free grammars
- Chapters 5-7

Useful for pattern matching.
- Finite automata
- Regular expressions
- Chapters 2-4



Turing machines.
- "Computable functions"
- Chapter 8

Turing machines that always halt.
- "Algorithms"
- "Decision problems"
- Chapter 9

Languages and Computing
(a preview)

- Imagine a program takes a file as an input and outputs *success* or *fail* depending on the file's contents.
- The file's contents can be considered a *string*.
  - All 8-bit bytes can be considered an *alphabet*.
- The set of strings (files) for which the program outputs *success* can be considered a *language*.
- **Turing machine**: A computer program that takes a string and outputs *success* or *fail* (for now…)
- The terms *language*, *function*, and *problem* often blur together in this context.

Undecidability and Intractability

- **Undecidable problems**: Not solvable by a Turing machine that always halts
  - Also known as *non-recursive* problems (using the terminology from before)
  - It is *impossible* for *any* algorithm to solve such a problem!

Undecidability and Intractability

- **Intractable problems**: Problems that can't be solved *efficiently*.
  - These are formally called "NP-hard" problems
  - We will discuss some of these when we discuss *recursive languages*.
  - This course only touches on this topic.

Formal Proofs

- Read Sections 1.2, 1.3 , and 1.4 in the textbook for more information, I assume familiarity with basic formal logic.

Quantifiers

- $\forall$: "For all"
  - Example: $\left(\forall x : x \geq 1 \ :: P(x)\right)$
    - "For all values of x where $x \geq 1$, $P(x)$ is true."
- $\exists$: "There exists"
  - Example: $\left(\exists x \ :: P(x)\right)$
    - "There exists some value of x where $P(x)$ is true."

Formal Proof Basics

- **Implication**: $A \Rightarrow B$
  - "A implies B."
  - Equivalent to $\neg A \vee B$.
- **Contrapositive**: The *contrapositive* of $A \Rightarrow B$ is $\neg B \Rightarrow \neg A$.
  - $(A \Rightarrow B) = (\neg B \Rightarrow \neg A)$.
  - Sometimes it's easier to prove an assertion by proving its contrapositive.

Formal Proof Basics

- **Contradiction**
  - To prove assertion A by contradiction, prove $\neg A \Rightarrow \mathit{false}$.
- **Counterexample**
  - You only need a single counterexample to disprove an assertion.
  - Example: Disprove $(\forall x : x \geq 0 \ :: x^2 = 2x)$ by counterexample.

Inductive Proofs Over Integers

- We want to prove assertion $S(n)$ is true for all integer values of n where $n \geq 0$.
- Two steps for an inductive proof:
  - **Basis**: Prove $\boxed{S(0)}$ is true
  - **Induction**: Assuming $S(n-1)$ is true, prove $S(n)$.

You can start with values other than 0 if necessary.

The "inductive hypothesis"

Example Proof by Induction

Example: Prove $\sum_{i=0}^{n} a^i = \frac{1-a^{n+1}}{1-a}$ by induction on n.

Basis: (Start with $n = 0$)

$$\sum_{i=0}^{0} a^i = a^0 = 1 = \frac{1 - a^{0+1}}{1 - a}$$

Example Proof by Induction (cont.)

Inductive step: Assume $\sum_{i=0}^{n-1} a^i = \frac{1-a^n}{1-a}$ is true.
(This is the original statement for values up to $n-1$)

Proof:

$$\sum_{i=0}^{n} a^i = \boxed{\sum_{i=0}^{n-1} a^i} + a^n$$

$$= \boxed{\frac{1-a^n}{1-a}} + a^n$$

$$= \frac{1-a^n+(1-a)a^n}{1-a}$$

$$= \frac{1-a^{n+1}}{1-a}$$

Our assumption in the inductive step lets us make this substitution.

Other Notes on Induction

- You may sometimes need to assume that the assertion holds for multiple prior values in the inductive step.
- You may sometimes need to prove multiple base cases.

Other Types of Induction

- **Structural Induction**: A fancy name for induction over the size of some structure.
  - Example: Prove a complete binary tree of height h has 2^h leaf nodes.
- **Mutual Induction**: A fancy name for needing to prove that multiple assertions continue to hold.
  - This includes proving invariants about state machines (we will do this *a lot*).

Sets

- **Notation**: $\{x \mid P(x)\}$
  - "The set of all values x such that $P(x)$ is true."
- $A = \{x \mid x \text{ is even}\}$
  - A is the set of all even numbers

Small sets can also be explicitly written:

- $A = \{1, 2, 3, 4\}$
  - A is the set containing the numbers 1, 2, 3 and 4.

More Set Notation

If A and B are sets:

- $A \subseteq B \equiv$ "A is a subset of B."
- $A = B \equiv A \subseteq B$ and $B \subseteq A$
- $A \subset B \equiv A \subseteq B$ and $A \neq B$ (strict subset)
- $A \cup B \equiv \{x \mid x \in A \text{ or } x \in B\}$
- $A \cap B \equiv \{x \mid x \in A \text{ and } x \in B\}$
- $A - B \equiv \{x \mid x \in A \text{ and } x \notin B\}$
- $A \times B \equiv \{(a, b) \mid a \in A \text{ and } b \in B\}$

Cardinality

- Sets A and B have the same *cardinality* if a one-to-one mapping of A onto B is possible.
- This can be counterintuitive for infinite sets.
  - For example, the set of all even integers has the same cardinality as the set of all integers.
    - Use the mapping $f(i) = 2i$.
- Sets are *countably infinite* if they have the same cardinality as the set of all integers.

Real Numbers Are Uncountable

The set of real numbers is *uncountably infinite*. We'll prove this by contradiction:

- Assume a mapping $f(i) = x_i$ exists.
  - i is the i^{th} integer
  - x_i is a real number
- Define a real number, y, such that the i^{th} digit after the decimal in y is not equal to the i^{th} digit after the decimal of x_i.

Real Numbers Are Uncountable

The set of real numbers is *uncountably infinite*. We'll prove this by contradiction:

- Assume a mapping $f(i) = x_i$ exists.
  - i is the i^{th} integer
  - x_i is a real number
- Define a real number, y, such that the i^{th} digit after the decimal in y is not equal to the i^{th} digit after the decimal of x_i.
- This construction makes it impossible for $f(i)$ to equal y for any i, contradicting the assumption.

Real Numbers Are Uncountable

- This is a *diagonalization* argument.
- Imagine putting real numbers in a table:

i	x_i
0	$.\underline{3}467\ldots$
1	$.1\underline{2}89\ldots$
2	$.99\underline{6}3\ldots$
3	$.000\underline{0}\ldots$
4	$.1122\ldots$
$\vdots$	

$y = .4371\ldots$

Add 1 to the digits along the "diagonal" to construct y's digits.

- We will use this kind of argument to show that noncomputable functions exist.