CSE547T Class 1

Jeremy Buhler

January 18, 2017

`http://classes.engineering.wustl.edu/cse547/`

1 Welcome and Administrivia

Welcome to CSE 547T!

- This is a course about the theory of computation.
- We'll study what computers are capable of doing (for a couple of different definitions of "computer") and how fast.
- "Active learning" – no good unless you engage with the material!!!

How is this course structured?

- **Class Meetings**: fundamental theorems and ideas
- **Practice Problems**: ungraded problems with solutions to reinforce understanding
- **Homeworks** (50%): graded problems without solutions to see if you can work with the material creatively. Probably about four of them.
- **Exams** (50%): final opportunity to demonstrate that you've mastered the material. Probably one midterm in class, plus one final during finals period.

What do you need to know right now?

- The course web site is your friend. All assignments and practice problems will appear there.
- Course discussions and communication with instructor and TAs is conducted through our Piazza board.
- Homeworks are submitted electronically via Blackboard. See the Electronic Homework Submission Guide on the website for detailed instructions and help with formatting.
- Read the course overview! It has all the grody administrative details.
- There is a **collaboration policy** for homeworks (not practice probs) – tries to balance fun of collective problem solving with promotion of individual skill building and assessment.

  - Please limit group problem solving to at most 3 people at once
  - Nothing written brought to discussion or carried away
  - "Iron Chef Rule" – one hour between discussion and writeups
  - Report (in the comment box of the homework submission interface) any assistance you receive.

Who is here to help you, and how do you contact us?

- TAs: Elliott Battle, Seth Ebner
- To contact myself or TAs, use Piazza.
- Our office hours are all posted on Piazza.

What do I expect you to know already?

- First: how to write a very basic formal proof.
- But we'll also learn proof strategies specific to the material.
- Second: asymptotic complexity analysis (for few algorithmic components)
- To warm up on formal proofs, there is an ungraded "Homework 0" on the web site. You should be able to do all the problems found there and produce suitable proofs.
- You can also use Homework 0 to make sure you know how to submit homework problems through Blackboard.
- If you have doubts about Homework 0 or want me to look at it, please submit it no later than January 25 and let me know (via Piazza) that we should check your solutions.

OK, on to the fun stuff...

2 Why Are We Here?

What's so important about automata theory?

- **Philosophy**: what can computers do? What can't they do?
- To answer these questions, we build highly simplified models that capture the essence of real computers. "Abstract machines"
- Essential features? (Ask) memory, logic (ability to branch), I/O
- Given an abstract machine with a certain set of features, can it do any computation? If not, what can it do?
- *Another question*: what is an algorithm? "Effective procedure" for solving all instances of a given problem.

- That's pretty fuzzy. What does "effective" mean? Probably needs to be specifiable on a computer, but with what capabilities?
- Do all problems have "effective procedures" to solve them? If not, can we prove that some problems cannot be solved?
- **Utility**: some of the abstract machines we study are practically very useful.
- Often, they can be implemented efficiently in hardware or software!
- *Finite Automata*: string munching, text search and pattern matching, regular expressions, state machines
- (*Push-Down Automata*: efficient parsing of programming languages, but we won't discuss that in here – take 431.)
- Knowing about these abstractions is very handy for your "bag of tricks" if you ever need to deal with data that looks like a string.
  - text documents
  - security logs
  - programs
  - network packets
  - DNA, RNA, and protein

OK, one more important item...

- Just like "effective procedures," we need "effective arguments" that certain computations are/are not possible on a given abstract machine.
- The mathematical standard of effective argument is a formal proof.
- I expect you to learn to write proofs that are complete, correct, and *can be understood by your peers in the class.*
- If you have taken CSE 441/541 or any upper-level math course, you know how to do this.
- *Warning*: it is very difficult to teach someone how to do good proofs, except by example, trial, and error.
- Volunteering to grade will help. Showing me your proofs will help.

3 On to Strings!

There are lots of different notions of computing abstraction.

- If you follow Kurt Gödel, you compute with numbers (arithmetics).
- If you follow Alonzo Church, you compute with functions (recursive function theory).
- In 547, we follow Alan Turing, so we compute with *strings.*

OK, time for definitions and notation!

- We use low-valued letters to denote characters: $a, b, c, \ldots$
- An **alphabet** Σ is a set of characters.
- *Examples*: $\{a, b, c\}$, $\{0, 1\}$, etc.
- A *string* over Σ is a (finite) ordered list of zero or more characters from Σ
- *Examples*: *ababcac*, 0101, etc
- We use high-valued letters to denote strings $u, v, w, x, y, z, \ldots$
- The length of a string x is denoted $|x|$
- For any alphabet, the special string consisting of 0 characters is denoted ε, called the **null** or **empty string**
- **Notation**: we denote by Σ^* the set of all strings (*including* ε) that can be generated from an alphabet Σ

So far, nothing interesting. But wait!

- We are typically interested in *subsets* of Σ^*
- *Example*: the set of all valid C programs (a subset of all ASCII strings)
- *Example*: the set of all prime numbers (a subset of all digit strings)
- *Example*: the set of all valid HTML web pages containing the name "Donald Trump"
- **Defn**: a **language** L over an alphabet Σ is a subset of Σ^*.
- Trivial examples for $\Sigma = \{a, b\}$: ϕ, $\{\varepsilon\}$, $\{a, b\}^*$, $\{a, b, aa, bb, aba\}$, $\{\varepsilon, a, ab\}$.

Normally, we are interested in more interesting and less trivial languages. We can describe L implicitly by one or more tests that qualify a string for membership in it.

- *Example*: $\{x \in \{a, b\}^* \mid |x| \text{ is even}\}$
- *Example*: $\{x \in \{a, b\}^* \mid x \text{ has more a's than b's}\}$
- *Example*: $\{x \in \{a, b\}^* \mid x \text{ does not contain "bbb"}\}$

We can also build up languages constructively from *operators*: union, concatenation.

- The **concatenation** of languages $L_1 \cdot L_2$ is the set of all strings xy, such that $x \in L_1$ and $y \in L_2$.
- *Example*: if $L_1 = \{a, b\}$ and $L_2 = \{c, d\}$, then $L_1 \cdot L_2 = \{ac, ad, bc, bd\}$.
- For simplicity, we may omit the dot and write $L_1 L_2$.
- Note that for any string x, $x\varepsilon = \varepsilon x = x$. Hence, $L\{\varepsilon\} = \{\varepsilon\}L = L$.

- **Notation**: for a string x, x^k means "k copies of x concatenated together." $x^0 = \varepsilon$.
- Same notation applies to languages: $L^k = LLLLL \ldots L$.
- Note that $L^0 = \{\varepsilon\}$ (the set of all strings formed by concatenating 0 copies of the strings in L).
- The **union** of languages $L_1 \cup L_2$ is simply the union of all their strings.

Finally, here are two more bits of notation that are quite common.

- The **Kleene closure** of a language L, denoted L^*, is the language containing all possible concatenated finite lists of of strings from L.

$$L^* = \bigcup_{k=0}^{\infty} L^k.$$

- The language containing all possible concatenated finite *nonempty* lists of strings from L is denoted L^+.

$$L^+ = \bigcup_{k=1}^{\infty} L^k.$$

4 A Few Words About Languages

When trying to prove a property of a language or combo of languages, it is usually a good idea to prove it for all underlying strings. The following example highlights some of the typical characteristics of language proofs.

- Give me a description in words for the language $L = \{a, ab\}^*$.
- My fave: set of strings over $\{a, b\}$ such that every b is preceded by an a.
- **Problem**: prove that this characterization is correct.
- *Standard approach*: a string x is in L iff it fits our description. **Two directions to prove!!!**
- One direction is usually easy. Suppose $x \in L$. If $x = \varepsilon$, it satisfies our description. Else, x must start with an a, since both a and ab do. Moreover, since the only b's in x can arise from occurrences of the substring ab, each b is indeed preceded by an a.
- The other direction is usually harder. Let's do it carefully with an inductive proof (on $|x|$). Suppose x satisfies our description; we claim that $x \in L$.
- **Base**: if $|x| = 0$, $x = \varepsilon$, which is in L because a Kleene closure always includes ε.
- **Ind**: suppose all strings of length $< n$ satisfy the claim. Consider a string x of length $n > 0$. x ends with either a or b.
- If x ends with a, decompose x as ya. If our description holds for x, *it also holds for every prefix of* x and so in particular for y. By induction, $y \in L$, so ya must also be in L (since it is the Kleene closure of a set that includes a).

- If x ends with b, decompose x as yab. This decomposition is always possible because every b in x is preceded by an a. Again, our description holds for y, and so by induction, $y \in L$. Conclude that yab must also be in L (since it is the Kleene closure of a set that includes ab). QED

OK, now for something more fun.

- How few strings can be in a language? Zero (the empty language).
- How many strings can be in a language? Infinitely many.
- How many *languages* can you generate from a (nonempty) alphabet Σ? All subsets of Σ^*. How big is that?
- *Claim*: any nonempty alphabet generates *uncountably* many languages!
- ("Uncountable" means too large a set to be put into 1:1 correspondence with natural numbers; an example is the reals)
- **Proof**: suppose not; i.e. suppose for some nonempty Σ, we can label all languages $L \subseteq \Sigma^*$ with an integer label, i.e. $L_1, L_2, L_3, \ldots$.
- I claim I can construct an infinite language $\hat{L}$ different from all labeled languages.
- First, observe that we can enumerate all strings in Σ^*. (E.g. order by size, and within same size lexicographically). Call them $x_1, x_2, x_3, \ldots$.
- Construct $\hat{L}$ as follows: for each label i,
  - If L_i contains x_i, $\hat{L}$ does not contain x_i.
  - Otherwise, $\hat{L}$ contains x_i.
- Above construction ensures that $\hat{L} \neq L_i$ for any i.
- Conclude that our labeling does not include all $L \subseteq \Sigma^*$. Hence, the set of languages in Σ^* is uncountable! QED

Interesting corollary that we'll use later: there is no way to put the set of languages in Σ^* in 1:1 correspondence with any single language. If we can encode algorithm descriptions (e.g. code, but also abstract machine descriptions) as finite-length strings over some finite alphabet, then there are languages with no corresponding algorithm to generate / recognize them! We'll see the importance later...