SAN JOSE STATE UNIVERSITY
DEPARTMENT OF ELECTRICAL ENGINEERING

CS 154 Formal Languages and Computability Spring 2012

Section 1 Room MH 225 Class 1: 01-25-12

T. Howell

Green sheet
- Topics
- Textbooks
- Prerequisites
  - Discrete math (Math 42)
  - Programming (CS 46B)
- Reaching me
- Grading
- Cheating
- Homework
  - Gradiance
  - Traditional
- Collaboration

Part 1

Basic questions:
- What does it mean for a function to be computable?
- Are there any noncomputable functions?
- How does computational power depend on programming constructs and machine features?

Fundamental concepts:
- State, transition, nondeterminism, reduction, undecidability
- These are important in many areas of computer science, and you will run into them over and over.

Computational models:
- Finite memory: finite automata
- Finite memory with stack: pushdown automata
- Unrestricted: Turing machines, recursive functions, programs in Java, C++, etc.

Languages and grammars
- Right linear grammars, regular expressions
- Context-free grammars
- Unrestricted grammars

Correspondence between computational models and grammars

Historical perspective:

Hilbert, Russell, and formalists 1900 - 1930.
Effective computability – Turing, Post, Kleene 1930 – 1960.
Gödel's incompleteness theorem is "a crowning intellectual achievement of the 20^{th} century."

This course deals in abstract models and their mathematical properties. Results are general and do not depend on the current fashions in programming languages and machines. These results form the foundation of computer science and are a large part of the distinction between computer science and computer programming. Paring computational models down to the bare minimum makes clear what is essential and of fundamental significance.

Part 2

Decision problem is a function whose outputs are "yes" or "no".
We need to know

the set A of all possible inputs.
the set $B \subseteq A$ of "yes" instances.

Examples:

A = all graphs; B = connected graphs
A = positive integers; B = primes

We like decision problems because they are simple, and the difficulty does not come purely due to having to write out a long answer.

Strings
We will use strings of characters from some alphabet as our inputs. Other data types can be encoded as strings.
An alphabet is any finite set of characters. Examples {0, 1, 2, …, 9} for decimal numbers, {0, 1} for binary numbers, ascii characters for text. We usually denote our alphabet by Σ. We call elements of Σ letters, symbols or characters. They can be anything as long as there are finitely many of them.
A string over Σ is any finite length sequence of elements of Σ. Example: if $\Sigma = \{a,b\}$, then abaab is a string over Σ of length 5. We usually use x, y, z to refer to strings.

The length of a string x is denoted |x|. |abaab| = 5.

The string of length 0 is called the null string and is denoted ε. (not $\in$). Thus $|\varepsilon| = 0$.

We write a^n for a string of n a's. Example a^4 = aaaa.
Inductive definition:

$a^0 = \varepsilon$
$a^{n+1} = a^n a$

The set of all strings over alphabet Σ is called Σ^*. For example
$\{a,b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$

Note $\varnothing^* = \{\varepsilon\}$.

Strings and sets:
Strings have order, and repetition matters. Sets are unordered, and repetition doesn't matter. $\{a, b\} = \{b, a\}$, but $ab \neq ba$. $\{a, a, b\} = \{a, b\}$ but $aab \neq ab$.

$\varnothing$, $\{\varepsilon\}$, ε are three different things.

Operations on strings.

Concatenation puts them together end to end. We write xy for the concatenation of x and y. Note $x\varepsilon = x$. $|xy| = |x| + |y|$.

Just as for symbols, we write x^n for the concatenation of n copies of the string x.
Inductive definition:
$x^0 = \varepsilon$
$x^{n+1} = x^n x$

Notation: If $a \in \Sigma$ and $x \in \Sigma^*$, we write #a(x) for the number of a's in x.
Examples #0(000110101011) = 6, and #1(00000) = 0.

A <u>prefix</u> of a string x is an initial substring. So y is a prefix of x if there exists a string z such that x = yz. Is x a substring of x? Yes. Is ε a substring of x? Yes A <u>proper prefix</u> of x is one other than x or ε.

Operations on sets.
|A| is cardinality.
Union
Intersection
Complement
Set concatenation
Powers (by concatenation)

Algebraic properties of these operations:
De Morgan laws:
$\sim(A \cup B) = \sim A \cap \sim B$
$\sim(A \cap B) = \sim A \cup \sim B$

Similar properties are true of the logic needed to prove theorems.
~(A OR B) = ~A AND ~B

~(A AND B) = ~A OR ~B

An important tool we will use to prove things about languages is mathematical induction. As practice we will prove that $1^3 + 2^3 + 3^3 + \ ... \ + n^3 = (1 + 2 + 3 + \cdots + n)^2$.

Proof:
Base case, n = 1. $1^3 = 1^2$.
Inductive hypothesis:
Assume $1^3 + 2^3 + 3^3 + \ ... \ + (n-1)^3 = (1 + 2 + 3 + \cdots + (n-1))^2$
Inductive step: We need to show
$1^3 + 2^3 + 3^3 + \ ... \ + (n-1)^3 + n^3 = (1 + 2 + 3 + \cdots + (n-1) + n)^2 = (1 + 2 + 3 + \cdots + (n-1))^2 + 2n\left(1 + 2 + 3 + \cdots + (n-1)\right) + n^2$.
By the inductive hypothesis,
$1^3 + 2^3 + 3^3 + \ ... \ + (n-1)^3 = (1 + 2 + 3 + \cdots + (n-1))^2$, so we only need to show that $n^3 = n^2 + \ 2n(1 + 2 + 3 + \cdots + (n-1))$.
It is known that $1 + 2 + 3 + \cdots + (n-1) = \frac{n(n-1)}{2}$. (If you don't know this, practice proving it by induction.)
$n^2 + \ 2n(1 + 2 + 3 + \cdots + (n-1) = \ n^2 + n^2(n-1) = \ n^3$
We have now shown that the inductive hypothesis implies
$1^3 + 2^3 + 3^3 + \ ... \ + n^3 = (1 + 2 + 3 + \cdots + n)^2$, and the proof is complete.