ECE 664
Lecture 2
Avi Kak
Purdue Univ.

Computability, Complexity, & Languages
Some Basic Definitions

Sets

- duplicates not allowed
- order of appearance not important
- $a \in S$ membership
- notation $S = \{a_1, a_2, \ldots, a_n\}$
- subset $R \subseteq S$
- ops: $R \cup S$, $R \cap S$, $\bar{R}$
- De Morgan's Law: $\overline{R \cup S} = \bar{R} \cap \bar{S}$ $\quad \overline{R \cap S} = \bar{R} \cup \bar{S}$
- Notation: $\Phi \equiv$ Empty set
  $N = \{0, 1, 2, \ldots\}$ the set of all natural numbers.

Tuples

- ordered sequence of symbols
- duplicates allowed
- n-tuple: $(a_1, a_2, \ldots, a_n)$
- 2-tuple $\equiv$ ordered pair
- 3-tuple $\equiv$ triple
- $(a_1, a_2, \ldots a_n) = (b_1, b_2, \ldots, b_n)$ iff $a_i = b_i$, $i = 1, 2, \ldots n$.

Language

- We start with an alphabet A which is a finite and non-empty set of symbols.
- An n-tuple drawn from A is a word or a string: $u = (a_1, a_2, \ldots, a_n)$
- We prefer to write $u = a_1 a_2 a_3 \ldots a_n$
- Length of u: $|u| = n$
- Unique null word: $\emptyset$
- A^*: the set of all words drawn from A
- Any subset of A^* is a language
- If $u, v \in A^*$, uv is a concatenation of u and v
- $u^{[n]} = \underbrace{uuu \ldots u}_{n \text{ times}}$

Cartesian Product

Given the sets $S_1, S_2, \ldots, S_n$, the C.P is denoted $S_1 \times S_2 \times \cdots \times S_n$. The C.P. is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ with $a_i \in S_i$

- When $S_1 = S_2 = \cdots = S_n \equiv S$, we denote C.P by S^n.

Functions

- A function is a set of ordered pairs
- If $(a,b) \in f$ and $(a,c) \in f$, then $b = c$
- If $(a,b) \in f$, we write $f(a) = b$
- Domain of f: The set of all a such that $f(a) = b$
- Range of f: The set of all b with $f(a) = b$ for every a in the domain of f

Functions Specified by Computer Programs

- We frequently write programs for functions whose arguments must come from certain designated sets. We often do not provide guarantees that our program will work for all elements in those sets. Should the program be invoked for input values that are outside the domain of the function (as actually computed by the program), we expect the program to throw an exception, as opposed to giving us a wrong answer. Throwing an exception amounts to saying that the program output is undefined for the inputs that cause the exception to be thrown.
- Therefore, the actual domain of a function as computed by a program may differ from the set for which the program was written.
- If we know that the actual domain of the function f computed by a program is a subset of S, we say f is a partial function on S.
- If f is a partial function on S and if $a \in S$ is in the actual domain of f, we write $f(a)\downarrow$. Otherwise, we write $f(a)\uparrow$.
- If the domain of a partial function f on S equals S, we say f is a total function on S.

Functions Defined on Cartesian Products

- Given a C.P.: $S_1 \times S_2 \times \cdots \times S_n$ we can define a function f: $((a_1, a_2, \ldots, a_n), b) \in f$ which we may write as $f((a_1, a_2, \ldots, a_n)) = b$ or more compactly as $f(a_1, a_2, \ldots, a_n) = b$
- When $S_1 = S_2 = \cdots = S_n \equiv S$, we have an n-ary function on S.
- 1-ary $\equiv$ unary
  2-ary $\equiv$ binary

n-ary Partial Function

- A partial function f on S^n is called an n-ary partial function on S

Predicates

- A predicate is a Boolean valued function. For 1-ary predicates, for $a \in S$, $P(a) = 1$ or $P(a) = 0$.
- Must be a total function.
- Given $R \subseteq S$, we defined a predicate $P(x) = \begin{cases} 1 & x \in R \\ 0 & x \notin R \end{cases}$ → such a predicate is called a characteristic function

Quantifiers

Predicates on N^m (same as m-ary predicates on N)

$P(t, x_1, x_2, \dots, x_n)$
$(n+1)$-ary pred. on N

$$Q(y, x_1, \dots, x_n) = P(0, x_1, \dots, x_n) \vee P(1, x_1, \dots, x_n) \vee \dots P(y, x_1, \dots, x_n)$$

$$Q(y, x_1, \dots, x_n) \Leftrightarrow (\exists t)_{\le y} P(t, x_1, \dots, x_n)$$

same predicate

bounded existential quantifier

$$(\forall t)_{\le y} P(t, x_1, \dots, x_n) \Leftrightarrow P(0, x_1, \dots, x_n) \,\&\, P(1, x_1, \dots, x_n) \dots$$

bounded universal quantifier

$(\exists t)$, $(\forall t)$

An existential quantifier answers the question whether a predicate is true for any member of a set

→ any() in Python

A universal quantifier answers the question whether a predicate is true for all members of a set.

→ all() in Python

An Example of Proof by Contradiction

Prove that $\left(\frac{m}{n}\right)^2 = 2$ has no sol. for m and n in $N = \{0, 1, 2, \dots\}$.

Proof will be based on the observation that the square of an even num is always even and the square of an odd num is always odd. Try squaring the even $m = 2k$ and the odd $m = 2k+1$.

PROOF: Assume there exists a sol for m and n in N. Then there exists at least one sol in which m and n are not both even (since you can cancel out any common factors of 2 from m and n). But, at the same time, we can show that in every sol both m and n must be even: $\left(\frac{m}{n}\right)^2 = 2$ leads to $m^2 = 2n^2$ implying that m must be even. Now set $m = 2k$ for some k. Since $m^2 = 2n^2$, we have $n^2 = 2k^2$, implying that n must also be even.

NOTE: This proves $\sqrt{2}$ is not rational.

→ You may say that the square-root of an even number – let's say 6 – is NOT even. Note the claim is that if the square-root of an even number exists in N, it must be even. Couple that with the fact that all numbers in N belong to two ~~nuts~~ disjoint subsets: even numbers with even squares and odd numbers with odd squares.

Proof By Contradiction

- ~~Assu~~ Suppose that the assertion we wish to prove is false
- Now logically combine the negative assertion with the initial statements.
- Logically derive a pair of statements that contradict each other.

- Prove that $\left(\frac{m}{n}\right)^2 = 2$ has no solution for $m, n \in N$ (See sol at top right)
- Let $x \in \{a, b\}^*$ such that $xa = ax$, then prove that $x = a^{[n]}$ for some $n \in N$

Mathematical Induction

It is used for proving statements like:

$$(\forall n)\, P(n)$$

where P is a predicate on N.

Proof by mathematical induction consists of proving

$P(0)$

and $(\forall n)(\text{If } \underbrace{P(n)}_{\text{Induction hypothesis}} \text{ then } P(n+1))$

$$(\forall n)\left[\text{If } (\forall m)_{m<n} P(m) \text{ then } P(n)\right]$$