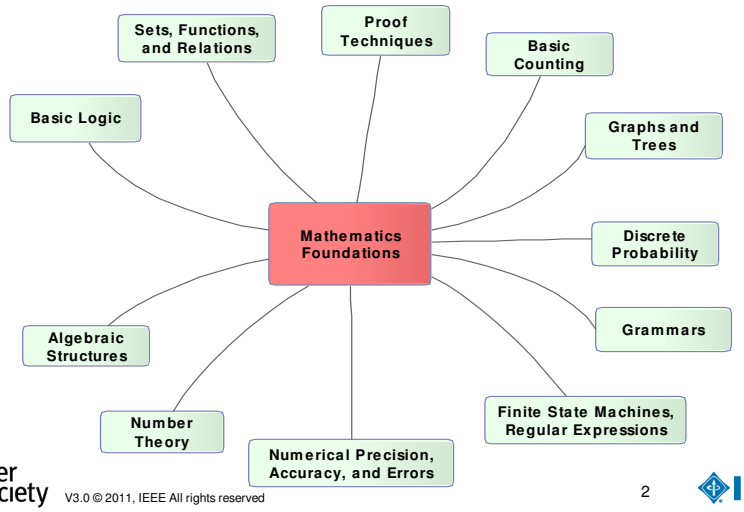
Module – 4 Software Engineering Foundations

Chapter - XIV

Mathematical Foundations

Mathematics Foundations - Roadmap



Discrete Mathematics

- Discrete Mathematics is the application of mathematics to solve distinct, separate (non-continuous) problems in many disciplines
- What calculus is to physics, discrete mathematics is to computer science
- Discrete means "data that consists of distinct elements such as characters, or to physical quantities having a finite number of distinctly recognizable values, as well as to processes and functional units that use those data" (ISO/IEC Std. 2382-1)

Content Area 1

Basic Logic

Logics, Proofs, Propositional Logic

- Logic
  - Formal reasoning principles
  - Criteria for valid inference
- Proof (argument)
  - Correct mathematical argument
  - Is one statement the logical consequence of some other statement(s)?
- Propositions (declarative language)
  - *p*: "Rex fetched the stick."
  - *q*: "5 + 5 = 11"
- Not propositions
  - "Fetch the stick, Rex."
  - "5 + *x* = 11"
- Truth variables and Boolean variables
  - True (T) = 1
  - False (F) = 0

Assignment - 1

- Which of the following are propositions and which are not propositions
  a. "The sun rose in the west today"
  b. "It is doubtful whether the bus will come"
  c. "Look at that flower"
  d. 5 * 5 = 25
  e. x * x = 36

Solution: a, b, d are propositions

Negations & Logical Connectives

Content Area 1: Basic Logic

Propositions		Negation	Conjunction	Disjunction	Exclusive Or	Implication	Biconditional
p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \rightarrow q$	$p \leftrightarrow q$
T	T	F	T	T	F	T	T
T	F	F	F	T	T	F	F
F	T	T	F	T	T	T	F
F	F	T	F	F	F	T	T

Assignment - 2

Content Area 1: Basic Logic

- Let p and q be propositions. Create truth tables for the following expressions –
  - $(p \vee q) \vee (p \wedge q)$
  - $(p \wedge q) \vee q$
  - $(p \vee q) \wedge (p \oplus q)$

Logical Expressions

- A logical expression is any set of propositions joined by logical connectives
- Let p and q be propositions where
  - *p*: Mary opened the door
  - *q*: The alarm was triggered
  - *r*: There is an intruder in the building
- "If Mary did not open the door and the alarm was triggered, then there is an intruder in the building"
  - $(\neg p \wedge q) \rightarrow r$

Translating a Logical Expression

- Let p and q be propositions where
  - *p*: Mary opened the door
  - *q*: The alarm was triggered
  - *r*: There is an intruder in the building
- Translate $\neg(q \leftrightarrow (p \vee r))$
- "It is not the case that the alarm was triggered if and only if Mary opened the door or there is an intruder in the building"

Assignment – 3.1

Content Area 1: Basic Logic

- Let p and q be propositions where
  - *p*: The alarm went off
  - *q*: Jo woke up on time
  - *r*: Jo got to work on time
- Create logical expression for the following English statement
  - "If the alarm did not go off, then Jo did not wake up on time and did not get to work on time"
- Solution: $\neg p \rightarrow (\neg q \wedge \neg r)$

Assignment – 3.2

Content Area 1: Basic Logic

- Let p and q be propositions where
  - *p*: The alarm went off
  - *q*: Jo woke up on time
  - *r*: Jo got to work on time
- Translate the following expression into English

$$\neg(r \leftrightarrow (p \vee q))$$

Solution: "It is not the case that Jo got to work on time if and only if the alarm went off or Jo woke up on time"

Precedence Order of Logical Operators

- ¬ (negation)
- ∧ (conjunction)
- ∨ (disjunction)
- → (implication)
- ↔ (biconditional)

Thus,

$\neg p \wedge q$ *is interpreted as*

$(\neg p) \wedge q$

Tautology & Contradiction

- A *tautology* is any logical expression that is true regardless of the truth values assigned to its propositions, denoted T_0
  - Negation law $p \vee \neg p$ only requires one of the elements to be true for it to be true. So it's a tautology.
- A *contradiction* is any compound statement that is false regardless of truth values assigned to its propositions, denoted F_0
  - Negation law $p \wedge \neg p$ will always evaluate to false. So it's a contradiction.

Propositions		Tautology	Contradiction
p	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
T	F	T	F
F	T	T	F

Logical Equivalence Definition

- If p ↔ q is a tautology, then p and q are said to be *logically equivalent*
  - Denoted as $p \Leftrightarrow q$
  - *p* is true if and only if *q* is true
  - *p* is false if and only if *q* is false
  - *p* and *q* have the same truth table results

Propositions		Implication	Biconditional
p	*q*	$p \rightarrow q$	$p \leftrightarrow q$
T	T	T	T
T	F	F	F
F	T	T	F
F	F	T	T

De Morgan's & Other Laws of Logic

De Morgan's Laws	Law of Double Neg.	Identity Laws	Domination Laws
$\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$ $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$	$\neg\neg p \Leftrightarrow p$	$p \wedge \mathrm{T}_0 \Leftrightarrow p$ $p \vee \mathrm{F}_0 \Leftrightarrow p$	$p \vee \mathrm{T}_0 \Leftrightarrow \mathrm{T}_0$ $p \wedge \mathrm{F}_0 \Leftrightarrow \mathrm{F}_0$
Idempotent Laws	**Commutative Laws**	**Associative Laws**	
$p \vee p \Leftrightarrow p$ $p \wedge p \Leftrightarrow p$	$p \vee q \Leftrightarrow q \vee p$ $p \wedge q \Leftrightarrow q \wedge p$	$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$ $(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$	
Distributive Laws		**Absorption Laws**	**Negation Laws**
$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$		$p \vee (p \wedge q) \Leftrightarrow p$ $p \wedge (p \vee q) \Leftrightarrow p$	$p \vee \neg p \Leftrightarrow \mathrm{T}_0$ $p \wedge \neg p \Leftrightarrow \mathrm{F}_0$

Assignment – 4

- Verify De Morgan's Laws by constructing truth tables

Proof of Consistency

- Proof of consistency means that a set of logical expressions do not contradict one another
- Is useful when establishing the consistency of software and hardware specifications
  - Statements should not contradict themselves
- Let the specifications be $\neg p$, $p \vee q$, and $p \rightarrow q$
  - If $\neg p$ is to be true, p must be false
  - So, if $p \vee q$ is to be true, q must be true
  - Since q is true, $p \rightarrow q$ is true irrespective of p

Inverse, Converse, Contrapositive

If p and q be propositions where $p \rightarrow q$

- Inverse $\neg p \rightarrow \neg q$
- Converse $q \rightarrow p$
- Contrapositive $\neg q \rightarrow \neg p$

Assignment – 5.1

Let proposition p be "Rex pressed the button" and proposition q be "Rex got a treat." The contrapositive of the implication $p \rightarrow q$ is:

a) "If Rex did not press the button, Rex did not get a treat"

b) **"If Rex did not get a treat then Rex did not press the button"**

c) "If Rex got a treat then Rex pressed the button"

Assignment – 5.2

- Verify using truth tables that an implication and its contrapositive are logically equivalent
  - i.e. they have the same value

Limitations of Propositional Logic

- Propositional logic cannot make conclusions from a general proposition to a specific item within a proposition or vice versa
  - Additional information required to make that conclusion may be missing
- Example
  - p: All employees got to work on time
  - q: Jo got to work on time
  - Cannot say p → q since it is not known if Jo is an employee
  - Cannot say q → p even if Jo is an employee since Jo may not be the *only* employee

Predicate Logic

- Predicate logic is an extension of propositional logic that introduces *variables* into logical statements
  - So, can be used to prove/disprove generalizations from specifics or vice-versa
- "Jo got to work on time"
  - "Jo" is the subject
  - "got to work on time" is the predicate
- P(x) = x got to work on time
  - Open statement which is neither true nor false
  - If x is assigned a value, P becomes a proposition
- A predicate with n variables is denoted as $P(x_1, x_2, x_3, .., x_n)$

Quantifiers

- Quantifiers express the extent to which a predicate is true for a set of values
- Universal quantifier, "for all x" $\forall x P(x)$
- Existential quantifier, "there exists an x" $\exists x P(x)$
- Universe of discourse or domain: U
- Universal and existential quantifiers have higher precedence than the other logical connectives

Vacuous and Trivial Proofs

- Vacuous proofs are proofs that can be proven true because the argument is known to be false
  - If p is known to be false in p→q, then by the definition of an implication, p→q must be true
  - If you design a process that deals with some or all employees, the "no employees" case is the vacuous case
- Trivial proofs are proofs that can be proven true because the conclusion is known to be true
  - Once q is proven true, p→q must also be true

Assignment – 6

1) Translate to predicate logic: "If every dog in this house has a flea collar then no dog has fleas."
2) If in the prior statement it is known that no dogs in the house have fleas, then a proof of this statement is a:
3) If in the prior statement it is known that there are no dogs in the house, then a proof of this statement is a:

1) Define the domain as dogs in this house. Let C(x) be "dog x has a flea collar" and F(x) be "dog x has fleas."

$$\forall x C(x) \rightarrow \neg F(x)$$

2) Trivial proof
3) Vacuous proof

Content Area 2

Sets, Functions, & Relations

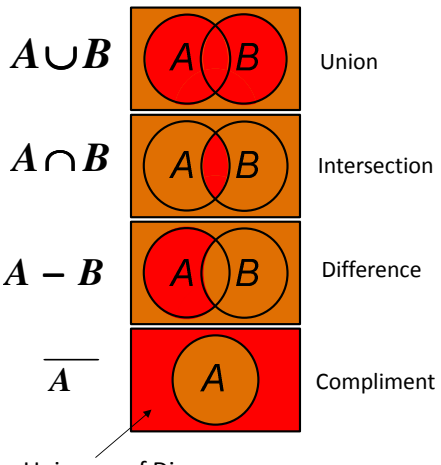
Sets

- Sets are unordered groupings of numbers and other objects, denoted by capital letters
- Naïve set theory: objects in a set are elements of that set. If set X has an element x, $x \in X$
- Sets are unordered and don't repeat identical elements
- Large sets, e.g.,: {a, b, c, …, z}
- Infinite sets, e.g., {1, 2, 3, …}
- Number sets: natural numbers **N**, integers **Z**, rational numbers **Q**, real numbers **R**
- Set builder notation defines set by some logic, e.g., the set of real numbers of all square roots of all positive integers:

$$\{\sqrt{x} \mid x \in \mathbf{Z}^{+}\}$$

Type of Sets and Set Operators

- Empty set $\varnothing$
- Subset
  $A \subseteq B \quad \{\forall x | x \in A \rightarrow x \in B\}$
- Proper subset
  $A \subset B \quad A \subseteq B, but\ A \neq B$
- Set system {{a}, {a, b}}
- Finite set size $|S|$



Power Sets & Cartesian Product

- A power set is the set of all subsets of a set S
  - Denoted P(S)
  - Example: if A = {1, 2}, P(A) = { Ø, {1}, {2}, {1,2}}
  - A set with n elements has 2^n elements in its power set
- The Cartesian product of two sets A and B is the set of all ordered pairs
  - Denoted $A \times B = \{(a,b) | a \in A \wedge b \in B\}$
  - Example: if A=(x,y), B=(1,2), then A x B = {(x,1), (x,2), (y,1), (y,2)}
  - Cartesian products form the foundations of relations and relational databases

Set Identities: Laws of Logic

Content Area 2: Sets, Functions, and Relations

Identity Laws	Domination Laws	De Morgan's Laws	Complementation Law
$A \cup \varnothing = A$ $A \cap \boldsymbol{U} = A$	$A \cup \boldsymbol{U} = \boldsymbol{U}$ $A \cap \varnothing = \varnothing$	$\overline{A \cup B} = \overline{A} \cap \overline{B}$ $\overline{A \cap B} = \overline{A} \cup \overline{B}$	$\overline{(\overline{A})} = A$

Idempotent Laws	Commutative Laws	Associative Laws
$A \cup A = A$ $A \cap A = A$	$A \cup B = B \cup A$ $A \cap B = B \cap A$	$A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$

Distributive Laws	Absorption Laws	Complement Laws
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$	$A \cup \overline{A} = \boldsymbol{U}$ $A \cap \overline{A} = \varnothing$

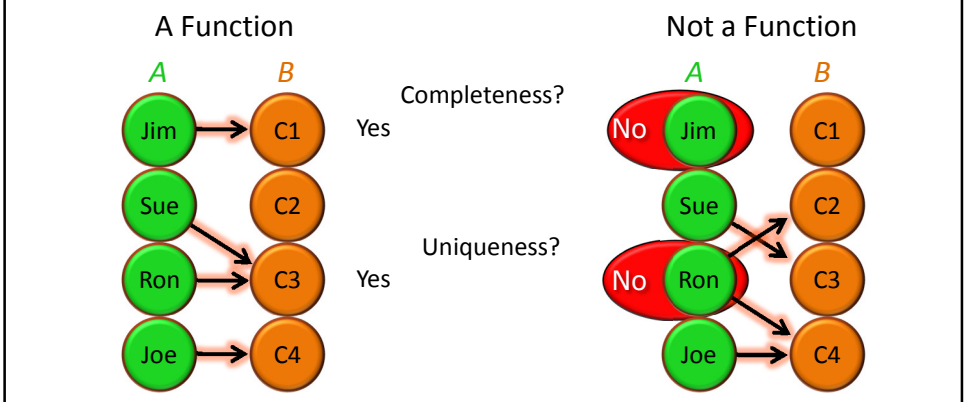
Functions

Content Area 2: Sets, Functions, and Relations

- Let A and B be nonempty sets. A function, denoted f, from A to B assigns **exactly** one element of B to **each** element of A, denoted f: A → B
  - This is read as "f maps A to B"
  - A is the domain of f, B is the codomain of f
- A function f has uniqueness, meaning that each element in A can map to at most one element in B
- A function f has completeness, meaning that each element in A must map to some element in B

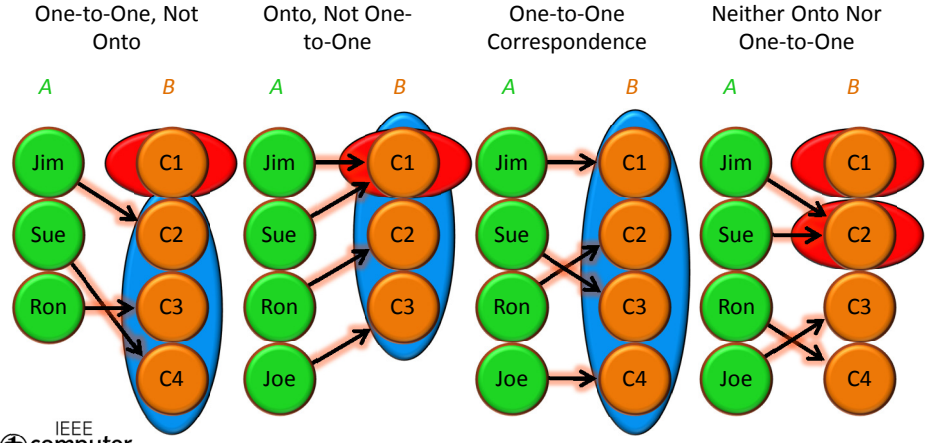
Functions: Example

Content Area 2: Sets, Functions, and Relations



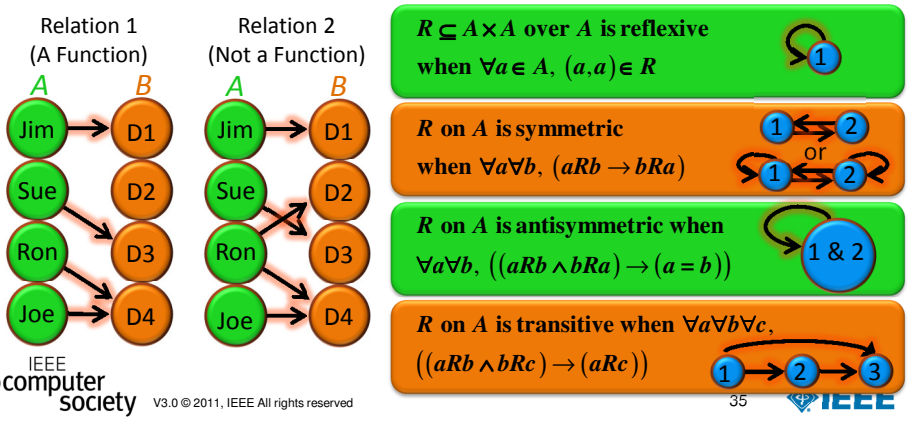
Function Definitions

Content Area 2: Sets, Functions, and Relations



Binary Relations

- Let A and B be nonempty sets. A binary relation R is a subset of the Cartesian product of these two sets.



Equivalence Relations and POSETs

- Equivalence relations are relations that are reflexive, symmetric, and transitive
  - Used for optimization e.g. eliminate redundant states in Finite State Machines, develop optimal test data sets
- Partial orderings are relations that are reflexive, antisymmetric, and transitive
  - A partially ordered set or POSET is a set whose elements have a partial ordering
  - Used for sequencing tasks e.g. PERT charts
- Let P(S) be the power set of set S. Then $(P(S), \subseteq)$ is a poset because the subset relation over P(S) is reflexive, anti-symmetric and transitive

Assignment – 7.1

Content Area 2: Sets, Functions, and Relations

- For every integer, show that the relation "less than or equal to" (≤) is reflexive, antisymmetric, and transitive, or in other words that (Z, ≤) is a partially ordered set (poset):

- Answer: Let x, y, and z be integers.
  - Reflexive: x ≤ x
  - Antisymmetric: If x ≤ y and y ≤ x, then x = y
  - Transitive: If x ≤ y and y ≤ z, then x ≤ z

Assignment – 7.2

Content Area 2: Sets, Functions, and Relations

- For every integer, show that the relation "greater than or equal to" (>=) is reflexive, antisymmetric, and transitive, or in other words that (Z, >=) is a partially ordered set (poset):

- Answer: Let x, y, and z be integers.
  - Reflexive: x >= x
  - Antisymmetric: If x >= y and y >= x, then x = y
  - Transitive: If x >= y and y >= z, then x >= z

Content Area 3

Proof Techniques

Terms related to Proofs

- A ***proof*** is a valid argument that is used to establish the truth of statements called theorems
- ***Theorems*** are statements that can be proven true
- A ***postulate*** or an ***axiom*** is a statement in logic that is seen as self-evident
- ***Conjectures*** are statements that may yet be proven true or false
- A ***fallacy*** is an incorrect proof or false logic
- A ***corollary*** is a statement that can be proven directly from an existing theorem
- A ***lemma*** is a basic theorem used to help break down complex theorems into components so that they can be proven singly

Valid Arguments

- Valid arguments
  - Premise leads to a conclusion

premise_1
premise_2
$\therefore$ conclusion

Modus Ponens

p
$p \rightarrow q$
$\therefore q$

Rule of Contradiction

$\neg p \rightarrow F_0$
$\therefore p$

Conjunction Rule

p
q
$\therefore p \wedge q$

Resolution Rule

$p \vee q$
$\neg p \vee r$
$\therefore q \vee r$

Assignment – 8

- Supply the conclusions for the following statements

Modus Tollens

$\neg q$
$p \rightarrow q$
$\therefore \neg p$

Rule of Cases

$p \rightarrow q$
$\neg p \rightarrow q$
$\therefore q$

Syllogism

$p \rightarrow q$
$q \rightarrow r$
$\therefore p \rightarrow r$

Disjunctive Syllogism

$p \vee q$
$\neg p$
$\therefore q$

Conditional Proof

$p \wedge q$
$p \rightarrow (q \rightarrow r)$
$\therefore r$

Proof by Cases

$p \rightarrow r$
$q \rightarrow r$
$\therefore (p \vee q) \rightarrow r$

Universal Instantiation

$\forall x P(x)$
$\therefore P(c)$

Existential Instantiation

$\exists x P(x)$
$\therefore P(c)$

Universal Generalization

$P(c)$
$\therefore \forall x P(x)$

Existential Generalization

$P(c)$
$\therefore \exists x P(x)$

Direct Proofs

- E.g., prove that the sum of two odd integers must be even:
  - Let x and y be odd integers. By the axiom defining "odd," there exist integers a and b for which x = 2a + 1 and y = 2b + 1
  - x + y = (2a + 1) + (2b + 1) = 2(a + b + 1)
  - Because x + y is twice the product of an integer, by the definition of "even," the sum must be even
- Direct proofs can be quite complex and require multiple valid rules of inference

Indirect Proofs

- **Proof by negation (counterexample)**
  - With universal quantifiers
    - Find single instance that makes P(x) true and Q(x) false

      $\forall x\left(P(x) \rightarrow Q(x)\right)$
  - With nested quantifiers
    - Apply DeMorgan's Laws to move negation from before a quantifier to logical equivalent with negation in predicate
- **Contrapositive**
  - Proves $p \rightarrow q$ by proving $\neg q \rightarrow \neg p$
- **Proof by contradiction**
  - Premises must not be self-contradictory or they would violate the rule of contradiction $(p \wedge \neg p) \rightarrow \mathbf{F_0}$

Proof by Mathematical Induction

Content Area 3: Proof Techniques

Given the predicate $P(n) \equiv 1+2+3+\ldots+n = \frac{n(n+1)}{2}$

Using mathematical induction, show P(n) is true $\forall n > 0,$

1. **Basis step:** Show Predicate P(1) is true $\longrightarrow \frac{1(1+1)}{2} = 1$
2. **Inductive step:** Show for k $\in$ $\mathbf{Z}^+$ that if P(k) is true then P(k+1) is true

$\forall k \geq 1, P(k) \rightarrow P(k+1)$ If $1+2+3+\ldots+n = \frac{n(n+1)}{2}$,

then $1+2+3+\ldots+n+n+1 = \frac{(n+1)(n+1+1)}{2}$

Substitute $\overset{?}{=} \frac{n(n+1)}{2} + n + 1 \overset{?}{=} \frac{(n+1)(n+1+1)}{2}$

× 2 $\overset{?}{=} n(n+1) + 2n + 2 \overset{?}{=} (n+1)(n+1+1)$

Simplify $\overset{?}{=} n^2 + 3n + 2 = n^2 + 3n + 2$

IEEE computer society

IEEE

Assignment – 9

Content Area 3: Proof Techniques

Given the predicate $P(n) = 1^2 + 2^2 + 3^2 + .. + n^2 = n(n+1)(2n+1)/6$

Using mathematical induction, show P(n) is true

IEEE computer society

IEEE

Content Area 4

Basic Counting

Basics of Counting

- Sum

$$\sum_{i=\text{Lower Limit}}^{\text{Upper Limit}} a_i = \sum_{i=m}^{n} a_i = a_m + a_{m+1} + \cdots + a_n \quad \text{e.g.,} \sum_{i=1}^{3} a_i = 1 + 2 + 3$$

- Product

$$\prod_{i=\text{Lower Limit}}^{\text{Upper Limit}} a_i = \prod_{i=m}^{n} a_i = a_m \times a_{m+1} \times \cdots \times a_n \quad \text{e.g.,} \prod_{i=1}^{3} a_i = 1 \times 2 \times 3$$

- Rule of sum
  - One or more tasks (e.g., X and Y) that can be done in one or more distinct ways (elements x and y, where $x \in X$ and $y \in Y$): $|X| + |Y|$
- Rule of product
  - One or more procedures that can be broken down into stages with distinct outcomes per stage: $|X| \times |Y|$

Example: Rule of Sum and Product

- Rule of Sum: If there are 20 different cars that a person can drive to go from point A to point B, and there are 10 different boats that he can use to go from point A to point B, then he has a total of 30 different modes of transportation.
- Rule of Product: If a software system has a light, regular, and deluxe configuration; is available in encrypted or unencrypted versions, and is available for 1, 2, 4, 8, 16 users, then there would be 3 x 2 x 5 = 30 varieties of the system to support.

Assignment – 10

- If a message identifier has *three alphabetical characters* followed by *three numerals*, how many identifiers are possible?
- Solution: Use Rule of Product
  - 26 x 26 x 26 x 10 x 10 x 10

Permutations & Combinations

Content Area 4: Basic Counting

- Permutations and Combinations are methods of finding the number of possibilities
  - Difference is whether ***order*** matters or not
- Consider the following poker hand:
  - 4♦, 8♥, 2♣, Q♠, A♠
- Is the above hand same as:
  - A♠, Q♠, 2♣, 8♥, 4♦
- Does the ***order*** in which the cards are handed out matter?
  - If yes, we are dealing with permutations
  - If no, we are dealing with combinations

Reference: "Handbook of discrete and combinatorial mathematics" by Kenneth Rosen and John Michaels, CRC Press, 2000

Permutations

Content Area 4: Basic Counting

- A permutation is a one-to-one correspondence function of a finite set to itself
  - Represents a re-arrangement of the set elements
- If set X = {1, 2, 3, 4, 5, 6, 7, 8}
  - {7, 5, 1, 4, 6, 8, 3, 2} is a permutation of X
- An r-permutation is an ordered arrangement of r elements of the set
  - P = {1, 3, 4, 2, 5} is a 5-permutation of the set X
- The notation for the number of r-permutations: P(n,r)
  - The set P above is one of P(8,5) permutations

Permutations – Calculation

$$P(n,r) = n(n-1)(n-2)..(n-r+1)$$

$$= \frac{n!}{(n-r)!} = \prod_{i=n-r+1}^{n} i$$

$n!$ is called "n factorial"

$$n! = 1*2*3..n = \prod_{i=1}^{n} i$$

- Example: Set P is one of P(8,5) permutations
  - Number of P(8,5) permutations = 4*5*6*7*8 = 6720
  - Choosing an order for all 8 numbers is P(8,8) = 8!
- Example: The poker hand is one of P(52,5) permutations
  - 4♦, 8♥, 2♣, Q♠, A♠
  - Number of poker hands (5 cards):
    - P(52,5) = 52*51*50*49*48 = 311,875,200

Assignment – 11.1

- If a school has 10 instructors, how many ways are there for 5 instructors to teach a particular class?

- Solution:
  - Order matters!!
  - Therefore, P(10,5)
    = 10*9*8*7*6
    = 30240

Assignment – 11.2

Content Area 4: Basic Counting

- How many permutations of {a, b, c, d, e, f} end with a?
- Solution:
  - Note that the set has 6 elements
  - The last character must be "a"; the rest can be in any order
  - Thus, it is a 5-permutation on the set {b, c, d, e, f}
  - P(5,5) = 5! = 120
- Why is it not P(6,5)?

Distinct Permutations

Content Area 4: Basic Counting

- Finding the number of distinct permutations requires finding the number of linear arrangements of a set of distinguishable objects
- If there are n objects in a set and n_1 denotes the first indistinguishable set, n_2 the second, n_r the r-th indistinguishable set, then the number of linear arrangements of n distinguishable objects is calculated as:

$$\text{If } n = n_1 + n_2 + \cdots + n_r \text{ then}$$

$$\text{linear arrangements of } n \text{ distinguishable objects} = \frac{n!}{n_1!n_2!\cdots n_r!}$$

$$\text{e.g., COLLECT} = \frac{7!}{2!2!1!1!1!} = \frac{(7)(6)(5)(4)(3)(2)(1)}{(2)(1)(2)(1)(1)(1)(1)} = \frac{5{,}040}{4} = 1{,}260$$

Assignment – 12

- Find the number of distinct permutations of letters in the word MISSISSIPPI
- Solution:
  - MISSISSIPPI = 1 M, 4 I, 4 S, 2 P (total 11)
  - MISSISSIPPI = 11!/(4! 4! 2! 1!)
    = 34650

Combinations

- What if ***order*** doesn't matter?
- E.g. in poker, the following two hands are equivalent:
  - 4♦, 8♥, 2♣, Q♠, A♠
  - A♠, Q♠, 2♣, 8♥, 4♦
- The number of r-combinations of a set with n elements, where n is non-negative and 0 ≤ r ≤ n is:

$$C(n,r) = \frac{n!}{r!(n-r)!}$$

Combinations Example

- How many different sets of 5 numbers can be selected from a set of 8 numbers?
  - Order does not matter
  - C(8,5) = 8!/5!3! = 56
- How many different poker hands are there (5 cards)?
  - C(52,5) = 52!/47!5! = 2598960

Assignment – 13.1

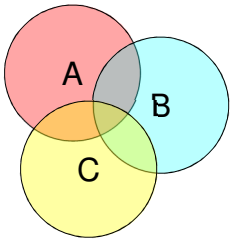
- How many bit strings of length 10 contain exactly four 1's?
- Solution:
  - Find the positions of the four 1's
  - Does the order of these positions matter?
    - No!!
    - Positions 2, 3, 5, 7 is the same as positions 7, 5, 3, 2
  - Thus, the answer is C(10,4) = 10!/6!4! = 210

Assignment – 13.2

- How many bit strings of length 10 contain at most four 1's?
- Solution:
  - Again, order does not matter
  - There can be 0, 1, 2, 3, or 4 occurrences of 1
  - Thus, the answer is:
  - C(10,0) + C(10,1) + C(10,2) + C(10,3) + C(10,4)
  - = 1+10+45+120+210 = 386
- How many bit strings of length 10 contain at least four 1's?

Inclusion Exclusion Principle

- Let A, B, C be three societies in a school. It is possible that some students belong to one or more societies.
- The combined membership of the three societies can be expressed as -



$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

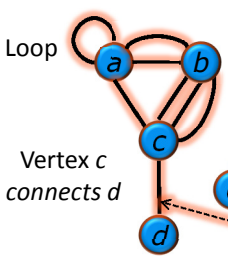
Content Area 5

Graphs and Trees

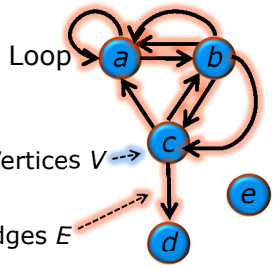
Different Types of Graphs

Graph $G = (V, E)$

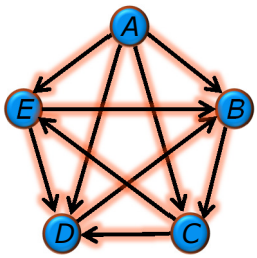
Undirected Multigraph



Directed Multigraph



Simple Directed Compete Graph



Vertex d is the endpoint for this edge

Cycles

A Wheel

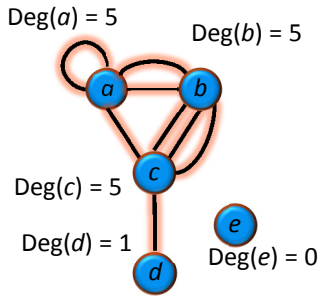
Uses of Graphs

- Graphs are used to create models such as telephone calls in a network or webpage hyperlinks
- Directed graphs are used to model state machines
- Graphs are also used in complexity analysis
- Precedence graphs help computer programs execute some code concurrently and determine what process must be executed before another
  - Similar to a PERT chart
- Graphs can assign *weights* to edges to capture real-world scenarios
  - Weights can be cost, response time, distance etc.

Undirected graph

- For *undirected* graph
  - Degree denoted by deg(v)
    - Loops are counted twice for a vertex
  - *Pendant* vertex – has one edge
  - *Isolated* vertex – has no edge
- Handshaking theorem:
  - sum of degrees on any undirected graph is even (2× the edges)
    - E.g. degree sum = 16 = 2*8(edges)

    => an undirected graph has even number of odd-degree vertices
    - E.g. 4 odd-degree vertices



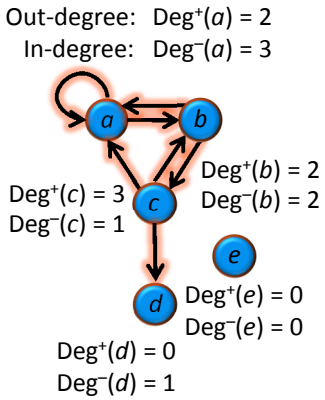
Degree is the number of edges attached to a vertex

Directed graph

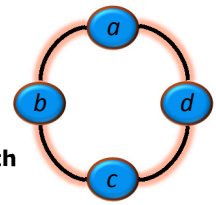
- For *directed* graph
  - Sum of out-degrees and sum of in-degrees is equal

$$E = \sum_{v \in V} \deg^+(v) = \sum_{v \in V} \deg^-(v)$$

  - Example:
    Out-degrees = 2+3+2=7
    In-degrees = 3+1+1+2=7



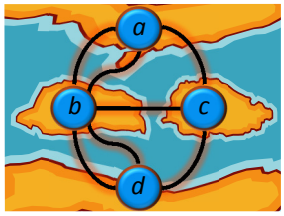
Paths



- A **path (trail, walk)** is one possible route for traveling between vertexes v_0 to v_n in graph G where |n| is the number of vertices visited from starting point v_0 to end point v_n
- The number of vertices traversed in the path is its **length**
- Sum of weights of edges in a path is its **weighted length**
- A path can visit the same vertices and edges multiple times
- A **circuit** is a path in which the starting point equals the end point
- A **simple path** is a path that contains the same edge only ONCE
- **Connected** refers to undirected graphs with a path between every pair of distinct vertices

7 Bridges of Konigsberg

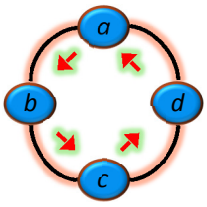
- City of Konigsberg had 7 bridges
- There were 2 islands *b* and *c* and two sides of the river, *a* and *d*
- Euler wanted to see if a person could make a **circuit** of all 7 bridges and end in the **same** place crossing each bridge **exactly once**



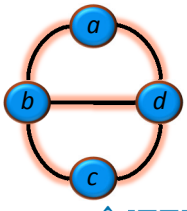
Euler Circuit and Trail

- Let G(V,E) be a undirected graph with no pendant vertices
- G has a **Euler circuit** if there is a circuit that traverses every edge of the graph exactly once
  - *A Euler circuit exists if and only if the graph is connected and each vertex has even degree*
- G has a **Euler trail** if there is a simple path from u to v that traverses each edge exactly once
  - *A Euler trail exists if and only if the graph is connected and has exactly 2 vertices of odd degree*

Euler Circuit

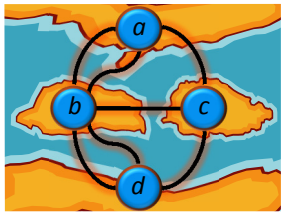


Euler Trail



7 Bridges: Euler Trail or Circuit?

- Vertices for the 7 bridges problems have the following degrees
  - Deg(a) = 3
  - Deg(b) = 5
  - Deg(c) = 3
  - Deg(d) = 3



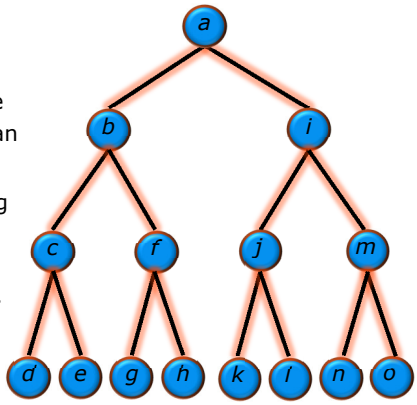
- No vertex is of even degree – so *not* a Euler circuit
- More than 2 odd degree vertices – so *not* a Euler path

Uses of Paths

- In Software Engineering, a path is defined as the sequence of instructions that may be performed in the execution of a program
- In file access, a path is the hierarchical sequence of directory and sub-directory names specifying the location of a file
- Paths are used for a number of problems, e.g.
  - Shortest path algorithm (Dijkstra)
  - Traveling salesman problem

Trees

- A tree is an undirected, connected simple graph with no circuits
- An undirected graph can only be a tree if, and only if, a unique, simple path can be found between any two vertices
- A tree is minimally connected, meaning that if any arc is removed, a disconnected graph results
- For a tree T = (V, E), the total number of vertices equals the total number of edges plus one, or $|V| = |E| + 1$ (Euler's formula)



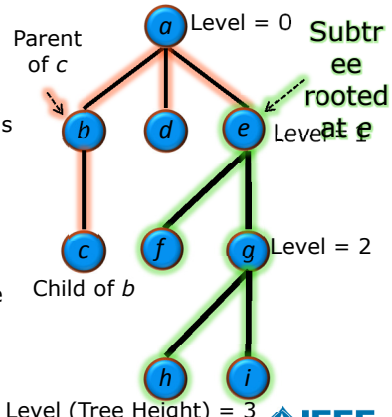
|V| = 15

|E| = 14

Tree Terminology

- **Root** – top of the tree
- **Internal vertex** – a vertex with 1 child or more
- **Leaf** – a vertex with no children
- **Height** – largest level
- **An m-ary tree** – each internal vertex has m or fewer children
- **A full m-ary tree** – an m-ary tree in which each internal vertex has exactly m children
- **Balanced tree** pertains to an m-ary tree of height h in which all children are at levels h or h – 1

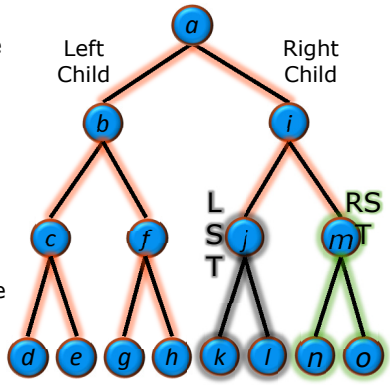
Rooted Tree
(an *m*-ary tree m= 3)



Tree Terminology

- A subtree of T is rooted at an internal vertex and includes all descendants of that vertex
  - The left subtree (LST) is rooted at the left child
  - The right subtree (RST) is rooted at the right child
- A sorted tree is created by sorting logic, such as by setting
  - the root as an initial key and
  - placing items less than or equal to the key in a LST and
  - items greater than the key in a RST

Full *m*-ary Tree m = 2
(Binary Tree)



Tree Theorems

- m-ary tree theorems

  Maximum number of leaves = m^h

  $$h \geq \lceil \log_m l \rceil$$

- Full m-ary tree theorems

  $$n = mi + 1 = \frac{ml - 1}{m - 1}$$

  $$i = \frac{n-1}{m} = \frac{l-1}{m-1}$$

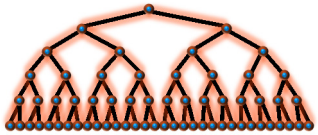
  $$l = i(m-1) + 1$$

  $$h = \lceil \log_m l \rceil$$

h = height of tree
l = number of leaves
n = number of vertices
i = number of internal vertices

Assignment – 14

- A server uses parallel processing arranged as a full binary tree (with a processor at each node, a path of serial processes, as the links, from root to a leaf). If the lowest level alone has 32 processors, then the server will have:
  - 5 serial processes using 63 processors
  - 5 serial processes using 64 processors
  - 6 serial processes using 63 processors
  - 6 serial processes using 64 processors



- Answer: c. Processes equals the height plus 1 (for level 0):

$$h = \lceil \log_m l \rceil = \lceil \log_2 32 \rceil = \lceil 5 \rceil = 5,\ h+1 = 6$$

- Number of vertices: $n = \frac{ml-1}{m-1} = \frac{2(32)-1}{2-1} = 63$

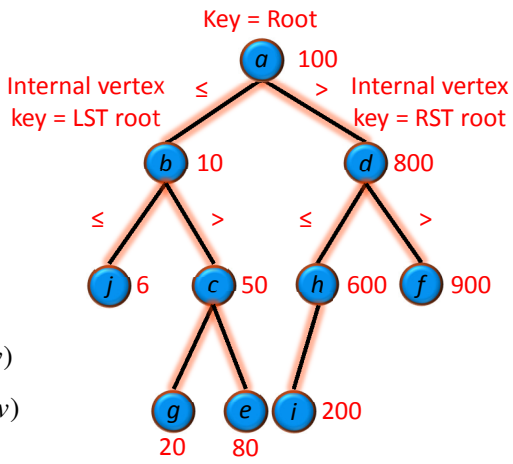
Binary Trees

Binary insertion search tree

- Helps search for items or model a decision tree
- Vertices are labeled with keys
- E.g. {100, 10, 50, 800, 900, 20, 600, 200, 6}
- Invariant property

$$\forall v \in LST(T), L(root) \geq L(v)$$

$$\forall v \in RST(T), L(root) < L(v)$$

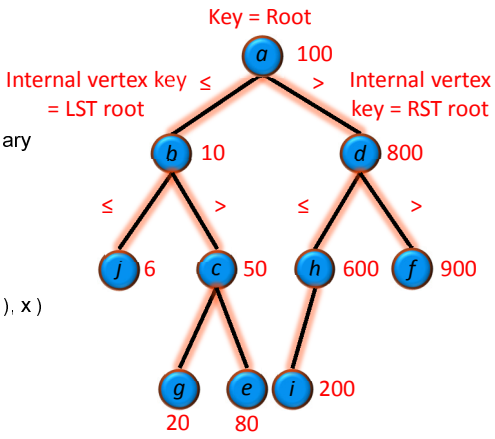


Binary Tree Search Algorithm

- Relies on recursion

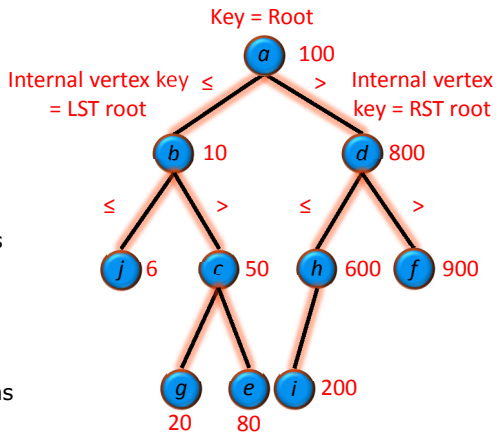
Function BinarySearch(T: ordered binary tree, x: item) : boolean

- if T is empty then return false
- if label(T) = x then return true
- if x < label(T) then BinarySearch(left_subtree(T), x)
- else BinarySearch(right_subtree(T), x)



Traversal Algorithms

- **Inorder** traversal (LNR)
  - Left-Node-Right
  - **j, b, g, c, e, a, i, h, d, f**
  - used for *infix* math notations
- **Preorder** traversal (NLR)
  - Node-Left-Right
  - **a, b, j, c, g, e, d, h, i, f**
  - used for *prefix* math notations
- **Postorder** traversal (LRN)
  - Left-Right-Node
  - **j, g, e, c, b, i, h, f, d ,a**
  - used for *postfix* math notations

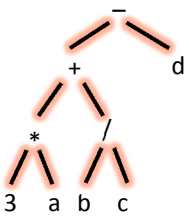


Assignment – 15

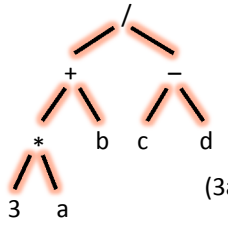
- Which of the following correctly apply infix, prefix or postfix notations?
  a. Infix for F1 and F2 is 3*a+b/c–d
  b. Postfix for F1 is –+*3a/bcd and F2 is /+*3ab–cd
  c. Prefix for F1 is 3a*bc/+d– and F2 is 3a*b+cd–/
  d. Infix for F1 is 3*a+/bc–d and F2 is 3*a+b/–dc

(3a + b/c) – d

Formula 1 (F1)



Formula 2 (F2)



(3a + b)/(c – d)

Content Area 6

Discrete Probability

Discrete Probability

- Discrete probability is the study of a sample space or set of outcomes that are finite
- Sample space S is a random experiment's set of ALL possible outcomes
- If all possible outcomes are equally likely, for *n* possible outcomes, the probability of each outcome is 1/*n*
- Event E is a subset of S and is defined as all possible outcomes that will satisfy the experiment's objective
- Probability of E is total of all positive outcomes divided by all possible outcomes i.e. $E \subseteq S,\ p(E) = \frac{|E|}{|S|}$

Assignment – 16.1

- What is the probability of getting a 1 when a six-sided dice is rolled?

- Solution:
  - |E| = 1
  - |S| = 6
  - Therefore, probability is 1/6 as $E \subseteq S,\ p(E) = \frac{|E|}{|S|}$

Assignment – 16.2

- What is the probability of 3 six-sided dice when rolled adding to the sum of 3?
- Solution:
  - Only way three dice can add up to 3 is if each rolls to a 1. So, successful outcome is {1, 1, 1}
  - Using rule of product, probability is (1/6)*(1/6)*(1/6) = 1/216

Sampling without Replacement

- Sampling without replacement is a calculation of probability in which the total size of the sample space is reduced after each sample
- *Example:* what are the odds of selecting the numbers 12, 2, and 32 from a pool of 40 consecutively numbered balls?
- There is ONLY ONE positive outcome
  - But 40, 39, 38 balls to choose in each drawing
  - Thus, odds are 1/(40*39*38) or 1 in 59,280

Assignment – 17

- What is the probability of winning a lottery where the user needs to correctly choose 5 numbers out of 55 integers and 1 number out of 42 integers?
- Solution:
  - There is ONLY ONE outcome
  - 5 numbers can be chosen from 55 using C(55, 5)
  - 1 number can be chosen from 42 using C(42,1)
  - Probability is 1/[C(55,5)*C(42,1)]
    = 1/146,107,962

Probability Theorems

- Probability of an event not happening (complement)

$$p(\overline{E}) = \left(1 - p(\overline{E})\right) = 1 - \frac{|E|}{|S|} = \left(\frac{|S| - |E|}{|E|}\right)$$

- Union of probabilities of two events E and F

$$p(E \cup F) = p(E) + p(F) - p(E \cap F)$$

- E and F are independent events if and only if

$$p(E \cap F) = p(E)\,p(F)$$

- E and F are mutually exclusive when

$$p(E \cap F) = \Phi$$

Probability

- Probability distribution is the function p where

$$\forall s \in S, 0 \le p(s) \le 1, \ and \ \sum_{s \in S} p(s) = 1$$

- Probability of an event E is $p(E) = \sum_{s \in E} p(s)$
- Example:
  - If E = (rain, snow) and rain=30% chance and snow=20% chance, the probability of rain OR snow is

$$p(E) = \sum_{s \in E} p(s) = p(snow) + p(rain) = \frac{3}{10} + \frac{2}{10} = \frac{5}{10} = 0.5$$

- Conditional probability of event E, given event F is

$$p(E|F) = \frac{p(E \cap F)}{p(F)}$$

Assignment – 18.1

- Let the random variable X represent the number of boys in a family. What would be the probability distribution for a family of two children?
- Solution: (Total possibilities = 4)
  - If first child is a boy
    - If second child is a boy, X=2
    - If second child is a girl, X=1
  - If first child is a girl
    - If second child is a boy, X=1
    - If second child is a girl, X=0

X	P(X)
0	1/4
1	1/2
2	1/4

Assignment – 18.2

Content Area 6: Discrete Probability

- Three coins are tossed. Let X be the number of heads obtained. Construct the probability distribution of X.
- Solution: (Total possibilities = 8)

X	P(X)
0	1/8
1	3/8
2	3/8
3	1/8

Coin 1	Coin 2	Coin 3
H	H	H
H	T	H
H	H	T
H	T	T
T	H	H
T	T	H
T	H	T
T	T	T

Uneven Probability Distribution

Content Area 6: Discrete Probability

- E.g., two 6-sided dice are rolled, both are weighted: p(1) = 3/8 to roll a 1 and p(2 through 6) = 1/8
- To roll a sum of 2:
  - Each dice must roll a 1, with probability 3/8 each
  - By product rule, probability = 3/8 x 3/8 = 9/64 = 14.06%
- If it were normal dice, probability = 1/6 x 1/6 = 2.77%

Value	Prob
1	3/8
2	1/8
3	1/8
4	1/8
5	1/8
6	1/8

Conditional Probability

- E.g., Consider three Boolean (1 or 0) events. If at least one of the 1st two events is a 1, what is the probability that 3rd is a 1?

$$p(E_2|E_1) = \frac{p(E_2 \cap E_1)}{p(E_1)}$$

$$= \frac{\frac{3}{8}}{\frac{6}{8}} = \frac{3}{6} = 50\%$$

- E1 represents at least one of the first two events is a 1
- E2 represents the third event is a 1

E_1	E_2	$E_1 \cap E_2$
111	111	111
110	110	110
101	101	101
100	100	100
001	001	001
010	010	010
011	011	011
000	000	000

Assignment – 19

- From a pool of 80 tennis balls, 10 are orange, 50 are blue and 20 are yellow. If two balls are selected randomly without replacement, what is the probability that a blue ball is selected if the first ball selected is a orange?
- Solution:
  - To calculate P(blue|orange)
  - 2nd ball picked is out of a total of 79 balls that are left
  - There are totally 50 blue balls
  - So P(blue|orange) = 50/79

Content Area 7

Grammars

Languages

- Grammar is the set of rules that describe how a language is to be used
- Syntax – grammatical rules of a language
- Semantics – meaning behind syntax
- Language L uses vocabulary V
  - L over V $\subseteq$ V*.
    V* is the set of all strings over V.
  - Nonterminals N
  - Terminals Σ. V – Σ = N.
  - Productions P. z0→z1, e.g., <sentence>→<subject><predicate>
  - Start symbol S
  - Empty string λ

Grammars

- Grammar G = (V, Σ, S, P)
  - **Concatenation** – appending of 2 strings
    Let A ⊆ V* and B ⊆ V*,
    AB = {ab | a ∈ A and b ∈ B}
  - **Derivation** – deriving one string from another
    If z0→z1 ∈ P and w0 = lz0r and w1 = lz1r are concatenated strings over V, then w1 can be derived directly from w0:
    w0 $\Rightarrow$ w1 or w0 $\overset{*}{\Rightarrow}$ wn
  - **Language** L(G) ⊆ Σ*
    - Denotes the language generated by grammar G

Types of Grammars

$G = (V, \sum, S, P)$ where $V = \{a, b, A, B, S\}$, $\sum = \{a, b\}$;
$P = \{S \rightarrow aABb, A \rightarrow bB, B \rightarrow b, B \rightarrow \lambda\}$
$S \Rightarrow aABb \Rightarrow abBBb \Rightarrow abbBb \Rightarrow abbb$

- **Context-Free Grammars**
  $w1 \rightarrow w2$ if and only if w1 = single symbol $\neq \sum$
- **Regular Grammars** $P = \{S \rightarrow aA, A \rightarrow a, A \rightarrow b\}$ $S \Rightarrow aA \Rightarrow ab$
  $w1 \rightarrow w2$ if and only if w1 = A and w2 = a or aB
- **Context-Sensitive Grammars** $V = \{a, b, c, R, T, U, V, W, S\}$ $\sum = \{a, b, c\}$
  $P = \{S \rightarrow aRc, R \rightarrow aRT, bTc \rightarrow bbcc, bTT \rightarrow bbUT\}$
  $w1 \rightarrow w2$ derivations exist only if w1=lAr and w2=lwr
  l and r must be present; they are the context
  A is a non-terminal; w, l and r can be terminal/non-terminal

Backus-Naur Form

- Backus-Naur Form (BNF) is a shorthand notation for expressing a context-free grammar
- BNF is widely adopted in Java, SQL, XML
- Nonterminal symbols are enclosed in brackets <>
- Productions with the same left-hand nonterminal can be represented as a single unit and are separated using vertical bars (|)
  - Symbol ::= is used for →
- Productions A→ABa, A→AA, A→ab become

  <A>::=<A><B>a|<A><A>|ab

Assignment – 20

- Let G = (V, Σ, S, P), where V = {a, b, A, B, S}, Σ = {a, b}, start symbol S, and P = {S → aABb, A → bB, B → b, B → λ}. This grammar allows derivations of the form w1→w2 if and only if w1 = a single symbol ≠ Σ. This grammar is a _____ and one valid derivation is _____

a. Regular grammar; S $\overset{*}{\Rightarrow}$ bbbbb
b. Context-free grammar; S $\overset{*}{\Rightarrow}$ abbbb
c. Context-sensitive grammar; S $\overset{*}{\Rightarrow}$ ababa
d. Backus-Naur Form; S $\overset{*}{\Rightarrow}$ ba

Solution: (b)

Content Area 8

FSM and Regular Expressions

Finite State Machines

- A Finite State Machine (FSM) is a computational model consisting of a finite number of states and transition between those states, possibly with accompanying actions
  - It is an abstract model for processes and machines that takes inputs but can delay responding until a specific state is achieved
- FSMs are used to model speech recognition, spell-checking, indexing etc.
- FSMs are divided into
  - FSMs with no output (finite state automata)
  - FSMs with output

Finite State Automata (FSA)

- A FSA has no output but helps recognize inputs
  - It either accepts or rejects a input string depending upon whether the last state is a "final" state
- Used in lexical analyzers within compilers to check whether a given set of characters is a valid identifier
- FSA denoted by M = (S, s_0, F, I, f), where

  S = set of states; s_0 = initial state; Final state(s) $\subseteq$ S

  I = finite input alphabet; State transition function f: $S \times I \rightarrow S$
- From s0, transitions are based on input string t $\subseteq$ I*
- To determine the next state: s'' = f(s, i)

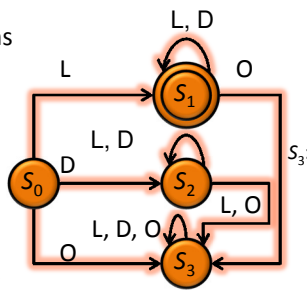
  where s = current state, i = input

FSA Example

- Lexical Analyzer – looks up lexemes (identifiers, keywords, etc.) and tokens and determines if they are valid
  - Return statements indicate actions to be done when input is complete
  - S1 is the final state; thus, valid inputs must begin only with a Letter (L) and must not contain any Other (O) character

Lexical Analyzer Transition Functions

S	Letter (L)	Digit (D)	Other (O)
s_0	s_1	s_2	s_3
s_1	s_1	s_1	s_3
s_2	s_3	s_2	s_3
s_3	s_3	s_3	s_3



Recognition?

S_1: `return(lexeme)`
Yes

S_3: `return(Int_Literal)`
No

S_2: `return(error)`
No

Regular Expressions

- Regular expressions, or patterns, are expressions based upon a regular grammar that provide an unambiguous way of identifying or describing sets of strings without needing to list all the elements
  - Regular expressions can identify particular patterns of characters in text e.g. man* can be used to identify both "man" as well as "manner"
- Some means of expressing alternatives and quantifiers exist for regular expressions:
  - A bar | presents alternatives: *theat(er|re)s matches both theaters and theatres*
  - A question mark ? states that there are none or one of the prior element: *rocks? matches both rock and rocks*
  - An asterisk * states that there are zero or more of the prior element: *bo* matches b, bo, boo, booo, and so forth*
  - A plus sign + states that there is one or more of the prior element: *bo+ matches bo, boo, booo, and so forth*

Kleene's Theorems

- A regular set is the set of all strings represented by a regular expression
- Kleene's Theorems
  - A set is regular if, and only if, a FSA can recognize it
  - If a set is regular, it was created by a regular grammar
- Regular sets
  - If R = $\varnothing$ or, if R = {λ} or,
  - If R = {x}, x $\in$ V or,
  - If X and Y are regular sets, then so are the following:
    - R = XY (concatenation),
    - R = X $\cup$ Y (union),
    - R = X* (Kleene's closure)
- Kleene Closure of X, where X $\subseteq$ V*
  - Denoted as set X*
  - Formed from the concatenation of many strings selected from X
  - Represented as

$$X^* = \bigcup_{k=0}^{\infty} X^k$$

  - Example:

    If X = {xy}, then

    X* = {λ, xy, xyxy, xyxyxy, ..}

Assignment – 21.1

- The regular expression 01* over {0, 1} represents which of the following set of strings
  a. *{0, 01, 010, 0100, 0101, ..}*
  b. *{0, 01, 011, 0111, 0111, ..}*
  c. *{01, 010, 011, 0101, 0110, ..}*
  d. *{0, 1, 01, 10, 011, 111, ..}*

- Solution: (b)

Assignment – 21.2

- The regular expression 1(1+0)* over {0, 1} represents which of the following set of strings
  a. *{1, 01, 010, 0100, 0101, ..}*
  b. *{1, 11, 101, 110, 011, ..}*
  c. *{1, 11, 10, 101, 110, 111, ..}*
  d. *{1, 01, 10, 110, 111, ..}*

- Solution: (c)

FSM with Output

- Finite state machines with output are abstract models for processes or machines that recognize specified inputs but can delay output until a specific state is achieved
  - *Example: A vending machine which advances from state to state as coins or bills are inserted, with a state for each possible combination of currency that can be input. When the required purchase amount is reached or exceeded, the finite state machine activates selection buttons and gives change as needed*
  - FSM translates an input string into an output string. It can also recognize inputs.
- FSM with output denoted by M = (S, s_0, I, O, f, f'), where

  S = set of states; s_0 = initial state;

  I = finite input alphabet; O = finite output alphabet

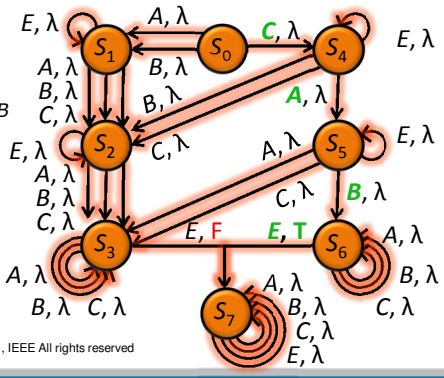
  Input State transition function *f*: $S \times I \rightarrow S$

  Output state transition function f': $S \times I \rightarrow O$

Example: FSM with Output

- A combination lock with buttons A, B, C, Enter (E), that can be pressed in any sequence
- If the correct sequence, CABE is pressed, the desired output message T for True results, and the FSM recognizes this input sequence
- For all other sequence of inputs, FSM does nothing and the output F for False results

3-Digit Combination, *CAB* plus Enter (*E*)



Recognition:
Let $x \in I^*$ be an input string for FSM *M*. *M* recognizes *x* if, and only if, the last output produced by *x* is the desired output (**T** in the e.g.)

Types of Finite State Machines

- Mealy Machines
  - FSM with outputs associated with edges (e.g. combination lock in previous slide)
- Moore Machines
  - FSM with outputs associated with states (vertices)
  - Does not have an output function f'
  - Can be used for translation or recognition
- Pushdown Automaton / Stack Machine
  - An FSA with a stack to provide for unlimited memory
  - Can recognize context-free grammar and languages
- Turing Machine
  - A FSM with a tape for infinite memory
  - Can recognize any kind of grammar including regular and context-free

Content Area 9

Numeric Precision, Accuracy, and Errors

Numerical Precision

- Numerical precision, accuracy and errors are very important when dealing with computer calculations
  - Calculations beyond a certain number of digits are rounded, so precision (reproducibility of a measurement) may be lost
  - Accuracy can be lost when nearly equal values are subtracted
- Significant Digits
  - Start with the leftmost digit NOT equal to 0 and end with the rightmost digit
  - There are 9 significant digits in 00124.223000, if the three rightmost 0s are significant
  - 0.00723 has 3 significant digits.
  - In 0.1234, 1 is the most significant digit and 4 is the least significant digit

Accuracy

- Accuracy is the measure of the deviation of a value from its correct or true value
- Two ways to measure accuracy
  - **Accurate to n significant digits** means that only n significant digits can be trusted; the rest are unreliable
  - **Accurate to n decimal places** means that only n digits starting to the right of the decimal place can be trusted; the rest are unreliable
- When performing calculations, the least accurate value is used to determine the reliability of the entire calculation.
  - For example, assuming all zeros are significant, 12.345 + 10 is accurate to two significant digits, but 12.345 + 10.000 is accurate to five.

Rounding

- Rounding is the reduction of the number of significant digits in a value
- Floating-point arithmetic can be rounded or chopped:
  - Rounded to n digits means that a value x is replaced by an n-digit value that approximates x with the least error
    - Let dn and dn+1 be the nth and (n + 1)st digit of x. Rounding to n digits is based on dn+1: If dn+1 > 5 or (dn+1 = 5 and dn is odd) then replace dn with dn+1; otherwise, don't change dn.
    - For example, 0.923 ≈ 0.92; 0.926 ≈ 0.93; 0.935 ≈ 0.94; 0.945 ≈ 0.94
  - Chopped or truncated to n digits means that a value x is replaced by an n-digit value by discarding all digits after the nth digit
    - For example, 0.923 ≈ 0.92; 0.926 ≈ 0.92; 0.935 ≈ 0.93; 0.946 ≈ 0.94

Assignment – 22

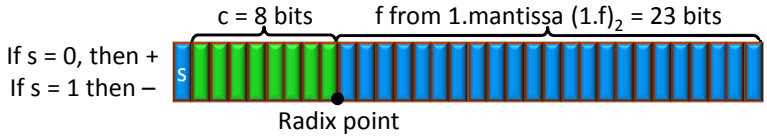
- Which of the following is correct rounding for the value: 0323.325555 if in Scenario 1 (S1) it is accurate to 6 significant digits and should be rounded appropriately and in S2 it is accurate to 5 decimal places and should be chopped appropriately?

a. S1: 323.33; S2: 323.32556
b. S1: 323.325; S2: 323.32555
c. S1: 323.325; S2: 323.32
d. S1: 323.326; S2: 323.32555

Solution: (d)

Single-Precision Floating Point Number

- Normalized floating-point notation (used in computing) represents any real value $x \neq 0$ as $x = \pm 0.d_1d_2d_3\ldots \times 10^n$ where n is an integer, d_i = digits 0 through 9, with $d_1 \neq 0$. Alternately, $x = \pm r \times 10^n$ where r, called the mantissa, is $1/10 \leq r < 1$
- Standard single-precision floating point **machine** numbers: $(-1)^s \times 2^{c-127} \times (1.f)_2$



Absolute and Relative Errors

- Let α be an exact value and β be an approximation of α created by measurement or rounding
  - E.g., α1= 4.54; β1 = 4.55; α2 = 2.02; β2 =2.01
- Error of β = α–β,
  - α1–β1 = 0.01; α2–β2 = –0.01
- Absolute error of β = |α–β|
  - |α1-β1| = 0.01; |α2–β2| = 0.01
- Relative error of β = $\frac{|\alpha-\beta|}{|\alpha|}$

$$\frac{|\alpha_1-\beta_1|}{|\alpha_1|} = \frac{0.01}{4.54}$$

More Significant

$$\frac{|\alpha_2-\beta_2|}{|\alpha_2|} = \frac{0.01}{2.02}$$

Content Area 10

Number Theory

Basic Number Theory

Let x and y be integers with $x \neq 0$. Let $m \in Z^+$

Dividing Integers

If integer z exists so that $x = yz$, then y divides x

Divisional Algorithm

If x = q*d + r, then
Quotient $q = x$ **div** d
Remainder $r = x$ ***mod*** d

Congruence

If m divides $x - y$, then x is congruent to y modulo m:

$x \equiv y \pmod{m}$

Congruence

- 2 integers have same remainder when divided by the same integer m
- $x \equiv y \pmod{m}$ if, and only if, there exists an integer k such that $x = y + km$
- If $x \equiv y \pmod{m}$ and $a \equiv b \pmod{m}$, then $x + a \equiv (y + b) \pmod{m}$ and $xa \equiv yb \pmod{m}$

Uses of Congruence

- Random numbers, Hashing function, Cryptology

Assignment – 23

Content Area10: Number Theory

2	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111
8	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

- Which of the following correctly expands $(4,271)_8$ to base 10 and converts $(351)_{10}$ to base 16?
  a. 2,233 and 15F
  b. 10,257 and F51
- Answer: (a)

$(4271)8 = 4\times8^3+2\times8^2+7\times8^1+1\times8^0 = 2{,}048+128+56+1 = 2{,}233$

$(351)10 = 351/16 = (16)21+15_0$, $21/16 = (16)1+5_1$, $1/16 = (16)0+1_2 = 15F$

Greatest Common Divisor

Content Area10: Number Theory

- GCD (x, y), where x,y>0 is the greatest integer g such that g|x (g divides x) and g|y (g divides y)
  - Use prime factorization to compute GCD
  - Let prime factorization of x be $x = p_1^{x1} p_2^{x2} .. p_n^{xn}$
  - Let prime factorization of y be $y = p_1^{y1} p_2^{y2} .. p_n^{yn}$
  - Then

$$\gcd(x, y) = p_1^{\min(x1,y1)} p_2^{\min(x2,y2)} .. p_n^{\min(xn,yn)}$$

where "min" is the smaller of x or y

- Example: GCD(350,540)
  - $350 = 2(5^2)(7)$; $540 = 2^2(3^3)(5)$
  - $GCD(350,540) = 2^1 5^1 = 10$

Euclidean Algorithm to compute GCD

{Precondition: x, y are positive integers
Procedure: gcd(x,y)
a = x; b = y;
while (b != 0) {
  r = a mod b ; a = b; b = r
}
{Postcondition: a = gcd(x,y)}

Least Common Multiple

- LCM (x, y), where x,y>0 is the smallest positive integer l divisible by both x and y
  - Use prime factorization to compute LCM

$$\text{lcm}(x, y) = p_1^{\max(x1,y1)} p_2^{\max(x2,y2)} .. p_n^{\max(xn,yn)}$$

"max" is larger of x or y

- Example: LCM(350,540)
  - $350 = 2(5^2)(7)$; $540 = 2^2(3^3)(5)$
  - LCM(350,540) = $2^2\ 3^3\ 5^2\ 7 = 18900$

- xy = GCD(x,y) * LCM(x,y)
  - 350 * 540 = 10 * 18900 = 189,000

Content Area 11

Algebraic Structures

Boolean Expressions

- Boolean expression is an expression that evaluates to true (T = 1) or false (F = 0)
- Bit string, e.g., 1100 bitwise XOR 1001 = 0101
- Basic logic concepts apply to Boolean algebra
- Let B = {0, 1} and b∈ B be a Boolean variable
  - 0, 1, b are Boolean expressions
- Recursively, if X and Y are Boolean expressions, then so are $\overline{X}$, (XY), and X+Y

Bits		Complement	Product	Sum
x	*y*	$\overline{x}$	*x* × *y*	*x* + *y*
1	1	0	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	0

Boolean Algebra Identities

De Morgan's Laws	Law of Double Complement	Identity Laws	Domination Laws
$\overline{(xy)} = \overline{x} + \overline{y}$ $\overline{(x+y)} = \overline{x}\,\overline{y}$	$\overline{\overline{x}} = x$	$x+0=x$ $x1=x$	$x+1=1$ $x0=0$
Idempotent Laws	**Commutative Laws**	**Associative Laws**	
$x+x=x$ $xx=x$	$x+y=y+x$ $xy=yx$	$(x+y)+z=x+(y+z)$ $(xy)z=x(yz)$	
Distributive Laws		**Absorption Laws**	**Unit and Zero Property**
$x+(yz)=(x+y)(x+z)$ $x(y+z)=xy+xz$		$x+xy=x$ $x(x+y)=x$	Unit: $x+\overline{x}=1$ Zero: $x\overline{x}=0$

Boolean Algebra

Truth Table for Two Functions

a	b	c	F 1	F 2	F_1 Products	F_2 Products
1	1	1	0	0	–	
1	1	0	1	0	*abc*	–
1	0	1	0	1	+	*abc*
1	0	0	0	0		
0	1	1	0	0	– –	
0	1	0	1	0	*abc*	
0	0	1	0	0		
0	0	0	0	0		

Miniterm

Literals

Sum-of-products

- Literal - a boolean variable or its complement
- Miniterm – product of literals
- Sum of products – sum of miniterms (sum of those miniterms that evaluate to value 1 for F1)

$$F1 = ab\bar{c} + \bar{a}b\bar{c}$$

- Product of sums – product of maxterms (product of those maxterms that evaluate to 1 for F1)

$$F1 = (a+b+\bar{c})(\bar{a}+b+\bar{c})$$

Assignment – 24.1

- Use DeMorgan's theorems to simplify the following expression

$$\overline{(\bar{c}*a*\bar{b})} + \overline{(\bar{a}*b)}$$

- Solution:

$$\overline{(\bar{c}*a*\bar{b})} + \overline{(\bar{a}*b)}$$
$$= \bar{\bar{c}} + \bar{a} + \bar{\bar{b}} + \bar{\bar{a}} + \bar{b}$$
$$= c + (a+\bar{a}) + (b+\bar{b})$$
$$= c + 1 + 1$$
$$= 1$$

Assignment – 24.2

Content Area 11: Algebraic Structures

- Solve the following expression using Boolean algebra

$$(a+\bar{b})(a+b)$$

- Solution:

$$\begin{aligned}&(a+\bar{b})(a+b)\\&=aa+ab+\bar{b}a+\bar{b}b\\&=a+ab+\bar{b}a+0\\&=a+a(b+\bar{b})\\&=a+a(1)=a\end{aligned}$$