Let's get started with...

Logic!

Fall 2002 CMSC 203 - Discrete Structures 1

Logic

- Crucial for mathematical reasoning
- Used for designing electronic circuitry
- Logic is a system based on propositions.
- A proposition is a statement that is either true or false (not both).
- We say that the truth value of a proposition is either true (T) or false (F).
- Corresponds to 1 and 0 in digital circuits

Fall 2002 CMSC 203 - Discrete Structures 2

The Statement/Proposition Game

"Elephants are bigger than mice."

Is this a statement? yes

Is this a proposition? yes

What is the truth value of the proposition? true

Fall 2002 CMSC 203 - Discrete Structures 3

The Statement/Proposition Game

"520 < 111"

Is this a statement? yes

Is this a proposition? yes

What is the truth value of the proposition? false

The Statement/Proposition Game

"y > 5"

Is this a statement? yes

Is this a proposition? no

Its truth value depends on the value of y, but this value is not specified.

We call this type of statement a propositional function or open sentence.

The Statement/Proposition Game

"Today is January 1 and 99 < 5."

Is this a statement? yes

Is this a proposition? yes

What is the truth value of the proposition? false

The Statement/Proposition Game

"Please do not fall asleep."

Is this a statement? no

It's a request.

Is this a proposition? no

Only statements can be propositions.

The Statement/Proposition Game

"If elephants were red,
they could hide in cherry trees."

Is this a statement? yes

Is this a proposition? yes

What is the truth value of the proposition? probably false

The Statement/Proposition Game

"$x < y$ if and only if $y > x$."

Is this a statement? yes

Is this a proposition? yes

...because its truth value does not depend on specific values of x and y.

What is the truth value of the proposition? true

Combining Propositions

As we have seen in the previous examples, one or more propositions can be combined to form a single compound proposition.

We formalize this by denoting propositions with letters such as p, q, r, s, and introducing several logical operators.

Logical Operators (Connectives)

We will examine the following logical operators:

- Negation (NOT)
- Conjunction (AND)
- Disjunction (OR)
- Exclusive or (XOR)
- Implication (if – then)
- Biconditional (if and only if)

Truth tables can be used to show how these operators can combine propositions to compound propositions.

Negation (NOT)

Unary Operator, Symbol: $\neg$

P	$\neg$P
true (T)	false (F)
false (F)	true (T)

Conjunction (AND)

Binary Operator, Symbol: $\wedge$

P	Q	P$\wedge$Q
T	T	T
T	F	F
F	T	F
F	F	F

Disjunction (OR)

Binary Operator, Symbol: $\vee$

P	Q	P$\vee$Q
T	T	T
T	F	T
F	T	T
F	F	F

Exclusive Or (XOR)

Binary Operator, Symbol: $\oplus$

P	Q	P$\oplus$Q
T	T	F
T	F	T
F	T	T
F	F	F

Implication (if - then)

Binary Operator, Symbol: $\rightarrow$

P	Q	P→Q
T	T	T
T	F	F
F	T	T
F	F	T

Biconditional (if and only if)

Binary Operator, Symbol: $\leftrightarrow$

P	Q	P↔Q
T	T	T
T	F	F
F	T	F
F	F	T

Statements and Operators

Statements and operators can be combined in any way to form new statements.

P	Q	¬P	¬Q	(¬P)∨(¬Q)
T	T	F	F	F
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

Statements and Operations

Statements and operators can be combined in any way to form new statements.

P	Q	$P\land Q$	$\neg(P\land Q)$	$(\neg P)\lor(\neg Q)$
T	T	T	F	F
T	F	F	T	T
F	T	F	T	T
F	F	F	T	T

Equivalent Statements

P	Q	$\neg(P\land Q)$	$(\neg P)\lor(\neg Q)$	$\neg(P\land Q)\leftrightarrow(\neg P)\lor(\neg Q)$
T	T	F	F	T
T	F	T	T	T
F	T	T	T	T
F	F	T	T	T

The statements $\neg(P\land Q)$ and $(\neg P) \lor (\neg Q)$ are logically equivalent, since $\neg(P\land Q) \leftrightarrow (\neg P) \lor (\neg Q)$ is always true.

Tautologies and Contradictions

A tautology is a statement that is always true.

Examples:

- $R\lor(\neg R)$
- $\neg(P\land Q)\leftrightarrow(\neg P)\lor(\neg Q)$

If $S\rightarrow T$ is a tautology, we write $S\Rightarrow T$.
If $S\leftrightarrow T$ is a tautology, we write $S\Leftrightarrow T$.

Tautologies and Contradictions

A contradiction is a statement that is always false.

Examples:
- $R \wedge (\neg R)$
- $\neg(\neg(P \wedge Q) \leftrightarrow (\neg P) \vee (\neg Q))$

The negation of any tautology is a contra-diction, and the negation of any contradiction is a tautology.

Exercises

We already know the following tautology:

$\neg(P \wedge Q) \Leftrightarrow (\neg P) \vee (\neg Q)$

Nice home exercise:

Show that $\neg(P \vee Q) \Leftrightarrow (\neg P) \wedge (\neg Q)$.

These two tautologies are known as De Morgan's laws.

Table 5 in Section 1.2 shows many useful laws.

Exercises 1 and 7 in Section 1.2 may help you get used to propositions and operators.

Let's Talk About Logic

- Logic is a system based on propositions.
- A proposition is a statement that is either true or false (not both).
- We say that the truth value of a proposition is either true (T) or false (F).
- Corresponds to 1 and 0 in digital circuits

Logical Operators (Connectives)

- Negation (NOT)
- Conjunction (AND)
- Disjunction (OR)
- Exclusive or (XOR)
- Implication (if – then)
- Biconditional (if and only if)

Truth tables can be used to show how these operators can combine propositions to compound propositions.

Tautologies and Contradictions

A tautology is a statement that is always true.

Examples:

- $R\vee(\neg R)$
- $\neg(P\wedge Q)\leftrightarrow(\neg P)\vee(\neg Q)$

If $S\rightarrow T$ is a tautology, we write $S\Rightarrow T$.
If $S\leftrightarrow T$ is a tautology, we write $S\Leftrightarrow T$.

Tautologies and Contradictions

A contradiction is a statement that is always false.

Examples:

- $R\wedge(\neg R)$
- $\neg(\neg(P\wedge Q)\leftrightarrow(\neg P)\vee(\neg Q))$

The negation of any tautology is a contradiction, and the negation of any contradiction is a tautology.

Propositional Functions

Propositional function (open sentence):
statement involving one or more variables,

e.g.: x-3 > 5.

Let us call this propositional function P(x), where P is the predicate and x is the variable.

What is the truth value of P(2) ? false

What is the truth value of P(8) ? false

What is the truth value of P(9) ? true

Propositional Functions

Let us consider the propositional function Q(x, y, z) defined as:

x + y = z.

Here, Q is the predicate and x, y, and z are the variables.

What is the truth value of Q(2, 3, 5) ? true

What is the truth value of Q(0, 1, 2) ? false

What is the truth value of Q(9, -9, 0) ? true

Universal Quantification

Let P(x) be a propositional function.

Universally quantified sentence:
For all x in the universe of discourse P(x) is true.

Using the universal quantifier ∀:
∀x P(x) "for all x P(x)" or "for every x P(x)"

(Note: ∀x P(x) is either true or false, so it is a proposition, not a propositional function.)

Universal Quantification

Example:
S(x): x is a UMBC student.
G(x): x is a genius.

What does $\forall x\ (S(x) \rightarrow G(x))$ mean ?

"If x is a UMBC student, then x is a genius."
or
"All UMBC students are geniuses."

Existential Quantification

Existentially quantified sentence:
There exists an x in the universe of discourse for which P(x) is true.

Using the existential quantifier $\exists$:
$\exists x\ P(x)$ "There is an x such that P(x)."
"There is at least one x such that P(x)."

(Note: $\exists x\ P(x)$ is either true or false, so it is a proposition, but no propositional function.)

Existential Quantification

Example:
P(x): x is a UMBC professor.
G(x): x is a genius.

What does $\exists x\ (P(x) \wedge G(x))$ mean ?

"There is an x such that x is a UMBC professor and x is a genius."
or
"At least one UMBC professor is a genius."

Quantification

Another example:
Let the universe of discourse be the real numbers.

What does $\forall x \exists y\ (x + y = 320)$ mean?

"For every x there exists a y so that $x + y = 320$."

Is it true? yes

Is it true for the natural numbers? no

Disproof by Counterexample

A counterexample to $\forall x\ P(x)$ is an object c so that P(c) is false.

Statements such as $\forall x\ (P(x) \rightarrow Q(x))$ can be disproved by simply providing a counterexample.

Statement: "All birds can fly."
Disproved by counterexample: Penguin.

Negation

$\neg(\forall x\ P(x))$ is logically equivalent to $\exists x\ (\neg P(x))$.

$\neg(\exists x\ P(x))$ is logically equivalent to $\forall x\ (\neg P(x))$.

See Table 3 in Section 1.3.

I recommend exercises 5 and 9 in Section 1.3.

...and now for something completely different...

Set Theory

Actually, you will see that logic and set theory are very closely related.

Set Theory

- Set: Collection of objects ("elements")
- $a \in A$ "a is an element of A"
  "a is a member of A"
- $a \notin A$ "a is not an element of A"
- $A = \{a_1, a_2, .., a_n\}$ "A contains..."
- Order of elements is meaningless
- It does not matter how often the same element is listed.

Set Equality

Sets A and B are equal if and only if they contain exactly the same elements.

Examples:

- $A = \{9, 2, 7, -3\}$, $B = \{7, 9, -3, 2\}$: $A = B$
- A = {dog, cat, horse},
  B = {cat, horse, squirrel, dog} : $A \neq B$
- A = {dog, cat, horse},
  B = {cat, horse, dog, dog} : $A = B$

Examples for Sets

"Standard" Sets:

- Natural numbers $\mathbf{N} = \{0, 1, 2, 3, ..\}$
- Integers $\mathbf{Z} = \{.., -2, -1, 0, 1, 2, ..\}$
- Positive Integers $\mathbf{Z}^+ = \{1, 2, 3, 4, ..\}$
- Real Numbers $\mathbf{R} = \{47.3, -12, \pi, ..\}$
- Rational Numbers $\mathbf{Q} = \{1.5, 2.6, -3.8, 15, ..\}$ (correct definition will follow)

Examples for Sets

- $A = \emptyset$ "empty set/null set"
- $A = \{z\}$ Note: $z \in A$, but $z \neq \{z\}$
- $A = \{\{b, c\}, \{c, x, d\}\}$
- $A = \{\{x, y\}\}$
  Note: $\{x, y\} \in A$, but $\{x, y\} \neq \{\{x, y\}\}$
- $A = \{x \mid P(x)\}$
  "set of all x such that P(x)"
- $A = \{x \mid x \in \mathbf{N} \wedge x > 7\} = \{8, 9, 10, ..\}$
  "set builder notation"

Examples for Sets

We are now able to define the set of rational numbers Q:

$\mathbf{Q} = \{a/b \mid a \in \mathbf{Z} \wedge b \in \mathbf{Z}^+\}$

or

$\mathbf{Q} = \{a/b \mid a \in \mathbf{Z} \wedge b \in \mathbf{Z} \wedge b \neq 0\}$

And how about the set of real numbers R?

$\mathbf{R} = \{r \mid r \text{ is a real number}\}$
That is the best we can do.

Subsets

$A \subseteq B$ "A is a subset of B"

$A \subseteq B$ if and only if every element of A is also an element of B.

We can completely formalize this:

$A \subseteq B \Leftrightarrow \forall x\ (x \in A \rightarrow x \in B)$

Examples:

$A = \{3, 9\}, B = \{5, 9, 1, 3\}$, $A \subseteq B$? true

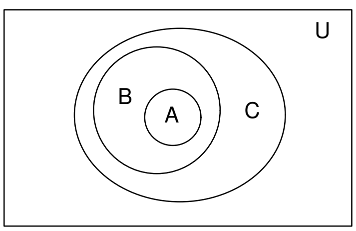
$A = \{3, 3, 3, 9\}, B = \{5, 9, 1, 3\}$, $A \subseteq B$? true

$A = \{1, 2, 3\}, B = \{2, 3, 4\}$, $A \subseteq B$? false

Subsets

Useful rules:

- $A = B \Leftrightarrow (A \subseteq B) \wedge (B \subseteq A)$
- $(A \subseteq B) \wedge (B \subseteq C) \Rightarrow A \subseteq C$ (see Venn Diagram)



Subsets

Useful rules:

- $\emptyset \subseteq A$ for any set A
- $A \subseteq A$ for any set A

Proper subsets:

$A \subset B$ "A is a proper subset of B"

$A \subset B \Leftrightarrow \forall x\ (x \in A \rightarrow x \in B) \wedge \exists x\ (x \in B \wedge x \notin A)$

or

$A \subset B \Leftrightarrow \forall x\ (x \in A \rightarrow x \in B) \wedge \neg \forall x\ (x \in B \rightarrow x \in A)$

Cardinality of Sets

If a set S contains n distinct elements, $n \in \mathbf{N}$, we call S a finite set with cardinality n.

Examples:

A = {Mercedes, BMW, Porsche},	$|A| = 3$
B = {1, {2, 3}, {4, 5}, 6}	$|B| = 4$
$C = \emptyset$	$|C| = 0$
$D = \{ x \in \mathbf{N} \mid x \leq 7000 \}$	$|D| = 7001$
$E = \{ x \in \mathbf{N} \mid x \geq 7000 \}$	E is infinite!

The Power Set

P(A) "power set of A"

$P(A) = \{B \mid B \subseteq A\}$ (contains all subsets of A)

Examples:

A = {x, y, z}

$P(A) = \{\emptyset, \{x\}, \{y\}, \{z\}, \{x, y\}, \{x, z\}, \{y, z\}, \{x, y, z\}\}$

$A = \emptyset$

$P(A) = \{\emptyset\}$

Note: $|A| = 0$, $|P(A)| = 1$

The Power Set

Cardinality of power sets:

$|P(A)| = 2^{|A|}$

- Imagine each element in A has an "on/off" switch
- Each possible switch configuration in A corresponds to one element in 2^A

A	1	2	3	4	5	6	7	8
x	x	x	x	x	x	x	x	x
y	y	y	y	y	y	y	y	y
z	z	z	z	z	z	z	z	z

- For 3 elements in A, there are $2 \times 2 \times 2 = 8$ elements in P(A)

Cartesian Product

The ordered n-tuple $(a_1, a_2, a_3, .., a_n)$ is an ordered collection of objects.

Two ordered n-tuples $(a_1, a_2, a_3, .., a_n)$ and $(b_1, b_2, b_3, .., b_n)$ are equal if and only if they contain exactly the same elements in the same order, i.e. $a_i = b_i$ for $1 \le i \le n$.

The Cartesian product of two sets is defined as:

$A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$

Example: $A = \{x, y\}$, $B = \{a, b, c\}$

$A \times B = \{(x, a), (x, b), (x, c), (y, a), (y, b), (y, c)\}$

Cartesian Product

The Cartesian product of two sets is defined as:

$A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$

Example:

A = {good, bad}, B = {student, prof}

A×B = {(good, student), (good, prof), (bad, student), (bad, prof)}

B×A = {(student, good), (prof, good), (student, bad), (prof, bad)}

Cartesian Product

Note that:

- $A \times \emptyset = \emptyset$
- $\emptyset \times A = \emptyset$
- For non-empty sets A and B: $A \ne B \Leftrightarrow A \times B \ne B \times A$
- $|A \times B| = |A| \cdot |B|$

The Cartesian product of two or more sets is defined as:

$A_1 \times A_2 \times .. \times A_n = \{(a_1, a_2, .., a_n) \mid a_i \in A \text{ for } 1 \le i \le n\}$

Set Operations

Union: $A \cup B = \{x \mid x \in A \vee x \in B\}$

Example: $A = \{a, b\}$, $B = \{b, c, d\}$
$A \cup B = \{a, b, c, d\}$

Intersection: $A \cap B = \{x \mid x \in A \wedge x \in B\}$

Example: $A = \{a, b\}$, $B = \{b, c, d\}$
$A \cap B = \{b\}$

Set Operations

Two sets are called disjoint if their intersection is empty, that is, they share no elements:
$A \cap B = \emptyset$

The difference between two sets A and B contains exactly those elements of A that are not in B:
$A - B = \{x \mid x \in A \wedge x \notin B\}$
Example: $A = \{a, b\}$, $B = \{b, c, d\}$, $A - B = \{a\}$

Set Operations

The complement of a set A contains exactly those elements under consideration that are not in A:
$A^c = U - A$

Example: $U = \mathbf{N}$, $B = \{250, 251, 252, ..\}$
$B^c = \{0, 1, 2, .., 248, 249\}$

Set Operations

Table 1 in Section 1.5 shows many useful equations.
How can we prove $A\cup(B\cap C) = (A\cup B)\cap(A\cup C)$?

Method I:

$x\in A\cup(B\cap C)$
$\Leftrightarrow x\in A \lor x\in(B\cap C)$
$\Leftrightarrow x\in A \lor (x\in B \land x\in C)$
$\Leftrightarrow (x\in A \lor x\in B) \land (x\in A \lor x\in C)$
(distributive law for logical expressions)
$\Leftrightarrow x\in(A\cup B) \land x\in(A\cup C)$
$\Leftrightarrow x\in(A\cup B)\cap(A\cup C)$

Fall 2002 CMSC 203 - Discrete Structures 55

Set Operations

Method II: Membership table

1 means "x is an element of this set"
0 means "x is not an element of this set"

A B C	$B\cap C$	$A\cup(B\cap C)$	$A\cup B$	$A\cup C$	$(A\cup B)\cap(A\cup C)$
0 0 0	0	0	0	0	0
0 0 1	0	0	0	1	0
0 1 0	0	0	1	0	0
0 1 1	1	1	1	1	1
1 0 0	0	1	1	1	1
1 0 1	0	1	1	1	1
1 1 0	0	1	1	1	1
1 1 1	1	1	1	1	1

Fall 2002 CMSC 203 - Discrete Structures 56

Set Operations

Every logical expression can be transformed into an equivalent expression in set theory and vice versa.

You could work on Exercises 9 and 19 in Section 1.5 to get some practice.

Fall 2002 CMSC 203 - Discrete Structures 57

…and the following mathematical appetizer is about…

Functions

Functions

A function f from a set A to a set B is an assignment of exactly one element of B to each element of A.

We write

$f(a) = b$

if b is the unique element of B assigned by the function f to the element a of A.

If f is a function from A to B, we write

$f: A \to B$

(note: Here, "$\to$" has nothing to do with if…then)

Functions

If $f:A \to B$, we say that A is the domain of f and B is the codomain of f.

If $f(a) = b$, we say that b is the image of a and a is the pre-image of b.

The range of $f:A \to B$ is the set of all images of elements of A.

We say that $f:A \to B$ maps A to B.

Functions

Let us take a look at the function $f: P \rightarrow C$ with
P = {Linda, Max, Kathy, Peter}
C = {Boston, New York, Hong Kong, Moscow}

f(Linda) = Moscow
f(Max) = Boston
f(Kathy) = Hong Kong
f(Peter) = New York

Here, the range of f is C.

Functions

Let us re-specify f as follows:

f(Linda) = Moscow
f(Max) = Boston
f(Kathy) = Hong Kong
f(Peter) = Boston

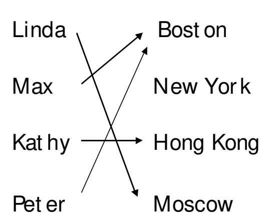
Is f still a function? yes

What is its range? {Moscow, Boston, Hong Kong}

Functions

Other ways to represent f:

x	f(x)
Linda	Moscow
Max	Boston
Kathy	Hong Kong
Peter	Boston



Functions

If the domain of our function f is large, it is convenient to specify f with a formula, e.g.:

$f: \mathbf{R} \to \mathbf{R}$
$f(x) = 2x$

This leads to:
$f(1) = 2$
$f(3) = 6$
$f(-3) = -6$
…

Functions

Let f_1 and f_2 be functions from A to **R**.
Then the sum and the product of f_1 and f_2 are also functions from A to **R** defined by:
$(f_1 + f_2)(x) = f_1(x) + f_2(x)$
$(f_1 f_2)(x) = f_1(x) f_2(x)$

Example:
$f_1(x) = 3x, \; f_2(x) = x + 5$
$(f_1 + f_2)(x) = f_1(x) + f_2(x) = 3x + x + 5 = 4x + 5$
$(f_1 f_2)(x) = f_1(x) f_2(x) = 3x\,(x + 5) = 3x^2 + 15x$

Functions

We already know that the range of a function $f: A \to B$ is the set of all images of elements $a \in A$.

If we only regard a subset $S \subseteq A$, the set of all images of elements $s \in S$ is called the image of S.

We denote the image of S by f(S):

$f(S) = \{f(s) \mid s \in S\}$

Functions

Let us look at the following well-known function:
f(Linda) = Moscow
f(Max) = Boston
f(Kathy) = Hong Kong
f(Peter) = Boston

What is the image of S = {Linda, Max} ?
f(S) = {Moscow, Boston}

What is the image of S = {Max, Peter} ?
f(S) = {Boston}

Properties of Functions

A function $f:A \rightarrow B$ is said to be one-to-one (or injective), if and only if

$\forall x, y \in A\ (f(x) = f(y) \rightarrow x = y)$

In other words: f is one-to-one if and only if it does not map two distinct elements of A onto the same element of B.

Properties of Functions

And again…

f(Linda) = Moscow	g(Linda) = Moscow
f(Max) = Boston	g(Max) = Boston
f(Kathy) = Hong Kong	g(Kathy) = Hong Kong
f(Peter) = Boston	g(Peter) = New York
Is f one-to-one?	Is g one-to-one?
No, Max and Peter are mapped onto the same element of the image.	Yes, each element is assigned a unique element of the image.

Properties of Functions

How can we prove that a function f is one-to-one?

Whenever you want to prove something, first take a look at the relevant definition(s):

$\forall x, y \in A\ (f(x) = f(y) \rightarrow x = y)$

Example:

$f: \mathbf{R} \rightarrow \mathbf{R}$

$f(x) = x^2$

Disproof by counterexample:

$f(3) = f(-3)$, but $3 \neq -3$, so f is not one-to-one.

Properties of Functions

…and yet another example:

$f: \mathbf{R} \rightarrow \mathbf{R}$

$f(x) = 3x$

One-to-one: $\forall x, y \in A\ (f(x) = f(y) \rightarrow x = y)$

To show: $f(x) \neq f(y)$ whenever $x \neq y$

$x \neq y$

$\Leftrightarrow 3x \neq 3y$

$\Leftrightarrow f(x) \neq f(y)$,

so if $x \neq y$, then $f(x) \neq f(y)$, that is, f is one-to-one.

Properties of Functions

A function $f: A \rightarrow B$ with $A, B \subseteq R$ is called strictly increasing, if

$\forall x, y \in A\ (x < y \rightarrow f(x) < f(y))$,

and strictly decreasing, if

$\forall x, y \in A\ (x < y \rightarrow f(x) > f(y))$.

Obviously, a function that is either strictly increasing or strictly decreasing is one-to-one.

Properties of Functions

A function $f:A \to B$ is called onto, or surjective, if and only if for every element $b \in B$ there is an element $a \in A$ with $f(a) = b$.

In other words, f is onto if and only if its range is its entire codomain.

A function $f: A \to B$ is a one-to-one correspondence, or a bijection, if and only if it is both one-to-one and onto.

Obviously, if f is a bijection and A and B are finite sets, then $|A| = |B|$.

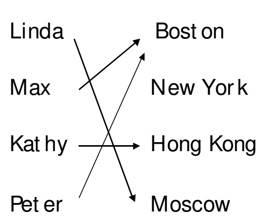
Properties of Functions

Examples:

In the following examples, we use the arrow representation to illustrate functions $f:A \to B$.

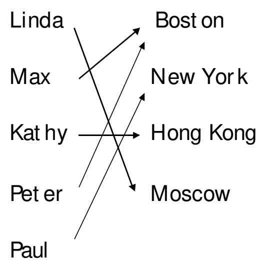
In each example, the complete sets A and B are shown.

Properties of Functions



Is f injective?
No.
Is f surjective?
No.
Is f bijective?
No.

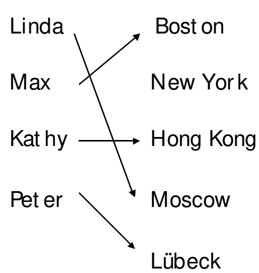
Properties of Functions



Is f injective?
No.
Is f surjective?
Yes.
Is f bijective?
No.

Fall 2002 CMSC 203 - Discrete Structures 76

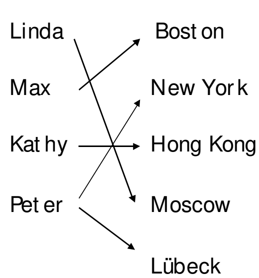
Properties of Functions



Is f injective?
Yes.
Is f surjective?
No.
Is f bijective?
No.

Fall 2002 CMSC 203 - Discrete Structures 77

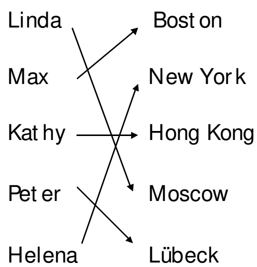
Properties of Functions



Is f injective?
No! f is not even a function!

Fall 2002 CMSC 203 - Discrete Structures 78

Properties of Functions



Is f injective?
Yes.
Is f surjective?
Yes.
Is f bijective?
Yes.

Inversion

An interesting property of bijections is that they have an **inverse function**.

The **inverse function** of the bijection $f:A \rightarrow B$ is the function $f^{-1}:B \rightarrow A$ with
$f^{-1}(b) = a$ whenever $f(a) = b$.

Inversion

Example:

f(Linda) = Moscow
f(Max) = Boston
f(Kathy) = Hong Kong
f(Peter) = Lübeck
f(Helena) = New York

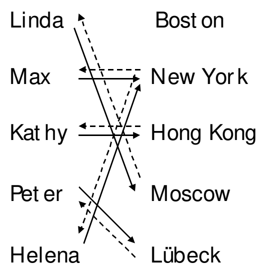
Clearly, f is bijective.

The inverse function f^{-1} is given by:

f^{-1}(Moscow) = Linda
f^{-1}(Boston) = Max
f^{-1}(Hong Kong) = Kathy
f^{-1}(Lübeck) = Peter
f^{-1}(New York) = Helena

Inversion is only possible for bijections (= invertible functions)

Inversion



$f \longrightarrow$

$f^{-1} \dashrightarrow$

f^{-1}:C→P is no function, because it is not defined for all elements of C and assigns two images to the pre-image New York.

Composition

The **composition** of two functions g:A→B and f:B→C, denoted by f∘g, is defined by

(f∘g)(a) = f(g(a))

This means that

- **first**, function g is applied to element a∈A, mapping it onto an element of B,
- **then**, function f is applied to this element of B, mapping it onto an element of C.
- **Therefore**, the composite function maps from A to C.

Composition

Example:

f(x) = 7x – 4, g(x) = 3x,
f:**R**→**R**, g:**R**→**R**

(f∘g)(5) = f(g(5)) = f(15) = 105 – 4 = 101

(f∘g)(x) = f(g(x)) = f(3x) = 21x - 4

Composition

Composition of a function and its inverse:

$(f^{-1} \circ f)(x) = f^{-1}(f(x)) = x$

The composition of a function and its inverse is the **identity function** $i(x) = x$.

Graphs

The **graph** of a function $f:A \rightarrow B$ is the set of ordered pairs $\{(a, b) \mid a \in A \text{ and } f(a) = b\}$.

The graph is a subset of $A \times B$ that can be used to visualize f in a two-dimensional coordinate system.

Floor and Ceiling Functions

The **floor** and **ceiling** functions map the real numbers onto the integers ($\mathbf{R} \rightarrow \mathbf{Z}$).

The **floor** function assigns to $r \in \mathbf{R}$ the largest $z \in \mathbf{Z}$ with $z \leq r$, denoted by $\lfloor r \rfloor$.

Examples: $\lfloor 2.3 \rfloor = 2$, $\lfloor 2 \rfloor = 2$, $\lfloor 0.5 \rfloor = 0$, $\lfloor -3.5 \rfloor = -4$

The **ceiling** function assigns to $r \in \mathbf{R}$ the smallest $z \in \mathbf{Z}$ with $z \geq r$, denoted by $\lceil r \rceil$.

Examples: $\lceil 2.3 \rceil = 3$, $\lceil 2 \rceil = 2$, $\lceil 0.5 \rceil = 1$, $\lceil -3.5 \rceil = -3$

Exercises

I recommend Exercises 1 and 15 in Section 1.6.

It may also be useful to study the graph displays in that section.

Another question: What do all graph displays for any function $f:\mathbf{R}\rightarrow\mathbf{R}$ have in common?

…and now for…

Sequences

Sequences

Sequences represent **ordered lists** of elements.

A **sequence** is defined as a function from a subset of **N** to a set S. We use the notation a_n to denote the image of the integer n. We call a_n a term of the sequence.

Example:

subset of **N**: 1 2 3 4 5 …

$\downarrow \; \downarrow \; \downarrow \; \downarrow \; \downarrow$

S: 2 4 6 8 10 …

Sequences

We use the notation $\{a_n\}$ to describe a sequence.

Important: Do not confuse this with the {} used in set notation.

It is convenient to describe a sequence with a **formula**.

For example, the sequence on the previous slide can be specified as $\{a_n\}$, where $a_n = 2n$.

The Formula Game

What are the formulas that describe the following sequences $a_1, a_2, a_3, \ldots$?

1, 3, 5, 7, 9, …	$a_n = 2n - 1$
-1, 1, -1, 1, -1, …	$a_n = (-1)^n$
2, 5, 10, 17, 26, …	$a_n = n^2 + 1$
0.25, 0.5, 0.75, 1, 1.25 …	$a_n = 0.25n$
3, 9, 27, 81, 243, …	$a_n = 3^n$

Strings

Finite sequences are also called **strings**, denoted by $a_1a_2a_3\ldots a_n$.

The **length** of a string S is the number of terms that it consists of.

The **empty string** contains no terms at all. It has length zero.

Summations

What does $\sum_{j=m}^{n} a_j$ stand for?

It represents the sum $a_m + a_{m+1} + a_{m+2} + \ldots + a_n$.

The variable j is called the **index of summation**, running from its **lower limit** m to its **upper limit** n. We could as well have used any other letter to denote this index.

Summations

How can we express the sum of the first 1000 terms of the sequence $\{a_n\}$ with $a_n=n^2$ for n = 1, 2, 3, …?

We write it as $\sum_{j=1}^{1000} j^2$.

What is the value of $\sum_{j=1}^{6} j$?

It is 1 + 2 + 3 + 4 + 5 + 6 = 21.

What is the value of $\sum_{j=1}^{100} j$?

It is so much work to calculate this…

Summations

It is said that Friedrich Gauss came up with the following formula:

$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

When you have such a formula, the result of any summation can be calculated much more easily, for example:

$$\sum_{j=1}^{100} j = \frac{100(100+1)}{2} = \frac{10100}{2} = 5050$$

Arithemetic Series

How does: $\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$???

Observe that:

1 + 2 + 3 +...+ n/2 + (n/2 + 1) +...+ (n - 2) + (n - 1) + n

= [1 + n] + [2 + (n - 1)] + [3 + (n - 2)] +...+ [n/2 + (n/2 + 1)]

= (n + 1) + (n + 1) + (n + 1) + ...+ (n + 1) *(with n/2 terms)*

= n(n + 1)/2.

Geometric Series

How does: $\sum_{j=0}^{n} a^j = \frac{a^{(n+1)} - 1}{(a-1)}$???

Observe that:

$S = 1 + a + a^2 + a^3 + \ldots + a^n$

$aS = a + a^2 + a^3 + \ldots + a^n + a^{(n+1)}$

so, $(aS - S) = (a - 1)S = a^{(n+1)} - 1$

Therefore, $1 + a + a^2 + \ldots + a^n = (a^{(n+1)} - 1) / (a - 1)$.
For example: 1 + 2 + 4 + 8 +...+ 1024 = 2047.

Useful Series

1. $\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$
2. $\sum_{j=0}^{n} a^j = \frac{a^{(n+1)} - 1}{(a-1)}$
3. $\sum_{j=1}^{n} j^2 = \frac{n(n+1)(2n+1)}{6}$
4. $\sum_{j=1}^{n} j^3 = \frac{n^2(n+1)^2}{4}$

Double Summations

Corresponding to nested loops in C or Java, there is also double (or triple etc.) summation:

Example:

$$\sum_{i=1}^{5}\sum_{j=1}^{2} ij$$

$$= \sum_{i=1}^{5}(i+2i)$$

$$= \sum_{i=1}^{5} 3i$$

$$= 3+6+9+12+15 = 45$$

Double Summations

Table 2 in Section 1.7 contains some very useful formulas for calculating sums.

Exercises 15 and 17 make a nice homework.

Enough Mathematical Appetizers!

Let us look at something more interesting:

Algorithms

Algorithms

What is an algorithm?

An algorithm is a finite set of precise instructions for performing a computation or for solving a problem.

This is a rather vague definition. You will get to know a more precise and mathematically useful definition when you attend CS420.

But this one is good enough for now…

Algorithms

Properties of algorithms:

- **Input** from a specified set,
- **Output** from a specified set (solution),
- **Definiteness** of every step in the computation,
- **Correctness** of output for every possible input,
- **Finiteness** of the number of calculation steps,
- **Effectiveness** of each calculation step and
- **Generality** for a class of problems.

Algorithm Examples

We will use a pseudocode to specify algorithms, which slightly reminds us of Basic and Pascal.

Example: an algorithm that finds the maximum element in a finite sequence

procedure max(a_1, a_2, .., a_n: integers)
max := a_1
for i := 2 **to** n
    if max < a_i **then** max := a_i
{max is the largest element}

Algorithm Examples

Another example: a linear search algorithm, that is, an algorithm that linearly searches a sequence for a particular element.

procedure linear_search(x: integer; $a_1, a_2, .., a_n$: integers)
i := 1
while ($i \leq n$ and $x \neq a_i$)
    i := i + 1
if $i \leq n$ **then** location := i
else location := 0
{location is the subscript of the term that equals x, or is zero if x is not found}

Algorithm Examples

If the terms in a sequence are ordered, a binary search algorithm is more efficient than linear search.

The binary search algorithm iteratively restricts the relevant search interval until it closes in on the position of the element to be located.

Algorithm Examples

binary search for the letter 'j'

search interval

a c d f g h j l m o p r s u v x z

center element

Algorithm Examples

binary search for the letter 'j'

search interval

a c d f g h j l m o p r s u v x z

center element

Algorithm Examples

binary search for the letter 'j'

search interval

a c d f g h j l m o p r s u v x z

center element

Algorithm Examples

binary search for the letter 'j'

search interval

a c d f g h j l m o p r s u v x z

center element

Algorithm Examples

binary search for the letter 'j'

search interval

a c d f g h j l m o p r s u v x z

center element

found !

Algorithm Examples

procedure binary_search(x: integer; a_1, a_2, .., a_n: integers)
i := 1 {i is left endpoint of search interval}
j := n {j is right endpoint of search interval}
while (i < j)
begin
    m := $\lfloor (i + j)/2 \rfloor$
    if x > a_m **then** i := m + 1
    else j := m
end
if x = a_i **then** location := i
else location := 0
{location is the subscript of the term that equals x, or is zero if x is not found}

Complexity

In general, we are not so much interested in the time and space complexity for small inputs.

For example, while the difference in time complexity between linear and binary search is meaningless for a sequence with n = 10, it is gigantic for $n = 2^{30}$.

Complexity

For example, let us assume two algorithms A and B that solve the same class of problems.

The time complexity of A is 5,000n, the one for B is $\lceil 1.1^n \rceil$ for an input with n elements.

For n = 10, A requires 50,000 steps, but B only 3, so B seems to be superior to A.

For n = 1000, however, A requires 5,000,000 steps, while B requires $2.5 \cdot 10^{41}$ steps.

Complexity

This means that algorithm B cannot be used for large inputs, while algorithm A is still feasible.

So what is important is the **growth** of the complexity functions.

The growth of time and space complexity with increasing input size n is a suitable measure for the comparison of algorithms.

Complexity

Comparison: time complexity of algorithms A and B

Input Size	Algorithm A	Algorithm B
n	5,000n	$\lceil 1.1^n \rceil$
10	50,000	3
100	500,000	13,781
1,000	5,000,000	$2.5 \cdot 10^{41}$
1,000,000	$5 \cdot 10^9$	$4.8 \cdot 10^{41392}$

Complexity

This means that algorithm B cannot be used for large inputs, while running algorithm A is still feasible.

So what is important is the **growth** of the complexity functions.

The growth of time and space complexity with increasing input size n is a suitable measure for the comparison of algorithms.

The Growth of Functions

The growth of functions is usually described using the **big-O notation**.

Definition: Let f and g be functions from the integers or the real numbers to the real numbers. We say that f(x) is O(g(x)) if there are constants C and k such that

$|f(x)| \leq C|g(x)|$

whenever $x > k$.

The Growth of Functions

When we analyze the growth of **complexity functions**, f(x) and g(x) are always positive.

Therefore, we can simplify the big-O requirement to

$f(x) \leq Cg(x)$ whenever $x > k$.

If we want to show that f(x) is O(g(x)), we only need to find **one** pair (C, k) (which is never unique).

The Growth of Functions

The idea behind the big-O notation is to establish an **upper boundary** for the growth of a function f(x) for large x.

This boundary is specified by a function g(x) that is usually much **simpler** than f(x).

We accept the constant C in the requirement

$f(x) \leq C \cdot g(x)$ whenever $x > k$,

because **C does not grow with x.**

We are only interested in large x, so it is OK if $f(x) > C \cdot g(x)$ for $x \leq k$.

The Growth of Functions

Example:

Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.

For $x > 1$ we have:

$x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2$

$\Rightarrow x^2 + 2x + 1 \leq 4x^2$

Therefore, for $C = 4$ and $k = 1$:

$f(x) \leq Cx^2$ whenever $x > k$.

$\Rightarrow f(x)$ is $O(x^2)$.

The Growth of Functions

Question: If f(x) is $O(x^2)$, is it also $O(x^3)$?

Yes. x^3 grows faster than x^2, so x^3 grows also faster than f(x).

Therefore, we always have to find the **smallest** simple function g(x) for which f(x) is O(g(x)).

The Growth of Functions

"Popular" functions g(n) are
n log n, 1, 2^n, n^2, n!, n, n^3, log n

Listed from slowest to fastest growth:

- 1
- log n
- n
- n log n
- n^2
- n^3
- 2^n
- n!

The Growth of Functions

A problem that can be solved with polynomial worst-case complexity is called **tractable**.

Problems of higher complexity are called **intractable.**

Problems that no algorithm can solve are called **unsolvable**.

You will find out more about this in CS420.

Useful Rules for Big-O

For any **polynomial** $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$, where $a_0, a_1, \ldots, a_n$ are real numbers, $f(x)$ is $O(x^n)$.

If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1 + f_2)(x)$ is $O(\max(g_1(x), g_2(x)))$

If $f_1(x)$ is $O(g(x))$ and $f_2(x)$ is $O(g(x))$, then $(f_1 + f_2)(x)$ is $O(g(x))$.

If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1 f_2)(x)$ is $O(g_1(x)\, g_2(x))$.

Complexity Examples

What does the following algorithm compute?

procedure who_knows($a_1, a_2, .., a_n$: integers)
m := 0
for i := 1 to n-1
  for j := i + 1 to n
    if $|a_i - a_j| > m$ **then** m := $|a_i - a_j|$
{m is the maximum difference between any two numbers in the input sequence}

Comparisons: n-1 + n-2 + n-3 + ... + 1

$= (n - 1)n/2 = 0.5n^2 - 0.5n$

Time complexity is $O(n^2)$.

Complexity Examples

Another algorithm solving the same problem:

procedure max_diff($a_1, a_2, .., a_n$: integers)
min := a1
max := a1
for i := 2 to n
  if a_i < min **then** min := a_i
  else if a_i > max **then** max := a_i
m := max - min

Comparisons: 2n - 2

Time complexity is O(n).

Let us get into...

Number Theory

Introduction to Number Theory

Number theory is about **integers** and their properties.

We will start with the basic principles of
- divisibility,
- greatest common divisors,
- least common multiples, and
- modular arithmetic

and look at some relevant algorithms.

Division

If a and b are integers with a ≠ 0, we say that a **divides** b if there is an integer c so that b = ac.

When a divides b we say that a is a **factor** of b and that b is a **multiple** of a.

The notation **a | b** means that a divides b.

We write **a X b** when a does not divide b (see book for correct symbol).

Divisibility Theorems

For integers a, b, and c it is true that

- if a | b and a | c, then a | (b + c)
  Example: 3 | 6 and 3 | 9, so 3 | 15.
- if a | b, then a | bc for all integers c
  Example: 5 | 10, so 5 | 20, 5 | 30, 5 | 40, …
- if a | b and b | c, then a | c
  Example: 4 | 8 and 8 | 24, so 4 | 24.

Primes

A positive integer p greater than 1 is called prime if the only positive factors of p are 1 and p.

A positive integer that is greater than 1 and is not prime is called composite.

The fundamental theorem of arithmetic:

Every positive integer can be written **uniquely** as the **product of primes**, where the prime factors are written in order of increasing size.

Primes

Examples:

15 =	$3 \cdot 5$
48 =	$2 \cdot 2 \cdot 2 \cdot 2 \cdot 3 = 2^4 \cdot 3$
17 =	17
100 =	$2 \cdot 2 \cdot 5 \cdot 5 = 2^2 \cdot 5^2$
512 =	$2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 2^9$
515 =	$5 \cdot 103$
28 =	$2 \cdot 2 \cdot 7$

Primes

If n is a composite integer, then n has a prime divisor less than or equal $\sqrt{n}$.

This is easy to see: if n is a composite integer, it must have two prime divisors p_1 and p_2 such that $p_1 \cdot p_2 = n$.

p_1 and p_2 cannot both be greater than $\sqrt{n}$, because then $p_1 \cdot p_2 > n$.

The Division Algorithm

Let **a** be an integer and **d** a positive integer. Then there are unique integers **q** and **r**, with **$0 \le r < d$**, such that **$a = dq + r$**.

In the above equation,
- **d** is called the divisor,
- **a** is called the dividend,
- **q** is called the quotient, and
- **r** is called the remainder.

The Division Algorithm

Example:

When we divide 17 by 5, we have

$17 = 5 \cdot 3 + 2.$

- 17 is the dividend,
- 5 is the divisor,
- 3 is called the quotient, and
- 2 is called the remainder.

The Division Algorithm

Another example:

What happens when we divide -11 by 3 ?

Note that the remainder cannot be negative.

$-11 = 3 \cdot (-4) + 1.$

- -11 is the dividend,
- 3 is the divisor,
- -4 is called the quotient, and
- 1 is called the remainder.

Greatest Common Divisors

Let a and b be integers, not both zero.
The largest integer d such that d | a and d | b is called the **greatest common divisor** of a and b.

The greatest common divisor of a and b is denoted by gcd(a, b).

Example 1: What is gcd(48, 72) ?

The positive common divisors of 48 and 72 are 1, 2, 3, 4, 6, 8, 12, 16, and 24, so gcd(48, 72) = 24.

Example 2: What is gcd(19, 72) ?

The only positive common divisor of 19 and 72 is 1, so gcd(19, 72) = 1.

Greatest Common Divisors

Using prime factorizations:

$a = p_1^{a_1} p_2^{a_2} \ldots p_n^{a_n}$, $b = p_1^{b_1} p_2^{b_2} \ldots p_n^{b_n}$,
where $p_1 < p_2 < \ldots < p_n$ and $a_i, b_i \in \mathbf{N}$ for $1 \le i \le n$

$\gcd(a, b) = p_1^{\min(a_1, b_1)} p_2^{\min(a_2, b_2)} \ldots p_n^{\min(a_n, b_n)}$

Example:

$a = 60 = 2^2\ 3^1\ 5^1$

$b = 54 = 2^1\ 3^3\ 5^0$

$\gcd(a, b) = 2^1\ 3^1\ 5^0 = 6$

Relatively Prime Integers

Definition:

Two integers a and b are **relatively prime** if gcd(a, b) = 1.

Examples:

Are 15 and 28 relatively prime?
Yes, gcd(15, 28) = 1.
Are 55 and 28 relatively prime?
Yes, gcd(55, 28) = 1.
Are 35 and 28 relatively prime?
No, gcd(35, 28) = 7.

Relatively Prime Integers

Definition:

The integers $a_1, a_2, .., a_n$ are **pairwise relatively prime** if $\gcd(a_i, a_j) = 1$ whenever $1 \leq i < j \leq n$.

Examples:

Are 15, 17, and 27 pairwise relatively prime?
No, because gcd(15, 27) = 3.

Are 15, 17, and 28 pairwise relatively prime?
Yes, because gcd(15, 17) = 1, gcd(15, 28) = 1 and gcd(17, 28) = 1.

Least Common Multiples

Definition:

The **least common multiple** of the positive integers a and b is the smallest positive integer that is divisible by both a and b.

We denote the least common multiple of a and b by lcm(a, b).

Examples:

lcm(3, 7) = 21

lcm(4, 6) = 12

lcm(5, 10) = 10

Least Common Multiples

Using prime factorizations:

$a = p_1^{a_1} p_2^{a_2} \ldots p_n^{a_n}$, $b = p_1^{b_1} p_2^{b_2} \ldots p_n^{b_n}$,
where $p_1 < p_2 < \ldots < p_n$ and $a_i, b_i \in \mathbf{N}$ for $1 \leq i \leq n$

$\mathrm{lcm}(a, b) = p_1^{\max(a_1, b_1)} p_2^{\max(a_2, b_2)} \ldots p_n^{\max(a_n, b_n)}$

Example:

$a = 60 = 2^2\, 3^1\, 5^1$

$b = 54 = 2^1\, 3^3\, 5^0$

$\mathrm{lcm}(a, b) = 2^2\, 3^3\, 5^1 = 4 \cdot 27 \cdot 5 = 540$

GCD and LCM

$a = 60 = 2^2\ 3^1\ 5^1$

$b = 54 = 2^1\ 3^3\ 5^0$

$\gcd(a, b) = 2^1\ 3^1\ 5^0 = 6$

$\operatorname{lcm}(a, b) = 2^2\ 3^3\ 5^1 = 540$

Theorem: $a \cdot b = \gcd(a, b) \cdot \operatorname{lcm}(a, b)$

Modular Arithmetic

Let a be an integer and m be a positive integer. We denote by **a mod m** the remainder when a is divided by m.

Examples:

9 mod 4 = 1

9 mod 3 = 0

9 mod 10 = 9

-13 mod 4 = 3

Congruences

Let a and b be integers and m be a positive integer. We say that **a is congruent to b modulo m** if m divides a – b.

We use the notation **a ≡ b (mod m)** to indicate that a is congruent to b modulo m.

In other words:
a ≡ b (mod m) if and only if **a mod m = b mod m**.

Congruences

Examples:

Is it true that $46 \equiv 68 \pmod{11}$?
Yes, because $11 \mid (46 - 68)$.

Is it true that $46 \equiv 68 \pmod{22}$?
Yes, because $22 \mid (46 - 68)$.

For which integers z is it true that $z \equiv 12 \pmod{10}$?
It is true for any $z \in \{..,-28, -18, -8, 2, 12, 22, 32, ..\}$

Theorem: Let m be a positive integer. The integers a and b are congruent modulo m if and only if there is an integer k such that $a = b + km$.

Congruences

Theorem: Let m be a positive integer.
If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then
$a + c \equiv b + d \pmod{m}$ and $ac \equiv bd \pmod{m}$.

Proof:
We know that $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$ implies that there are integers s and t with
$b = a + sm$ and $d = c + tm$.

Therefore,
$b + d = (a + sm) + (c + tm) = (a + c) + m(s + t)$ and
$bd = (a + sm)(c + tm) = ac + m(at + cs + stm)$.

Hence, $a + c \equiv b + d \pmod{m}$ and $ac \equiv bd \pmod{m}$.

The Euclidean Algorithm

The **Euclidean Algorithm** finds the **greatest common divisor** of two integers a and b.

For example, if we want to find gcd(287, 91), we **divide** 287 by 91:

$287 = 91 \cdot 3 + 14$

We know that for integers a, b and c,
if $a \mid b$ and $a \mid c$, then $a \mid (b + c)$.

Therefore, any divisor of 287 and 91 must also be a divisor of $287 - 91 \cdot 3 = 14$.

Consequently, gcd(287, 91) = gcd(14, 91).

The Euclidean Algorithm

In the next step, we divide 91 by 14:

$91 = 14 \cdot 6 + 7$

This means that gcd(14, 91) = gcd(14, 7).

So we divide 14 by 7:

$14 = 7 \cdot 2 + 0$

We find that 7 | 14, and thus gcd(14, 7) = 7.

Therefore, gcd(287, 91) = 7.

The Euclidean Algorithm

In **pseudocode**, the algorithm can be implemented as follows:

```
procedure gcd(a, b: positive integers)
x := a
y := b
while y ≠ 0
begin
    r := x mod y
    x := y
    y := r
end {x is gcd(a, b)}
```


Representations of Integers

Let b be a positive integer greater than 1.
Then if n is a positive integer, it can be expressed **uniquely** in the form:

$n = a_k b^k + a_{k-1} b^{k-1} + \ldots + a_1 b + a_0$,

where k is a nonnegative integer,
$a_0, a_1, \ldots, a_k$ are nonnegative integers less than b,
and $a_k \neq 0$.

Example for b=10:

$859 = 8 \cdot 10^2 + 5 \cdot 10^1 + 9 \cdot 10^0$

Representations of Integers

Example for b=2 (binary expansion):

$(10110)_2 = 1\cdot2^4 + 1\cdot2^2 + 1\cdot2^1 = (22)_{10}$

Example for b=16 (hexadecimal expansion):

(we use letters A to F to indicate numbers 10 to 15)

$(3A0F)_{16} = 3\cdot16^3 + 10\cdot16^2 + 15\cdot16^0 = (14863)_{10}$

Representations of Integers

How can we construct the base b expansion of an integer n?

First, divide n by b to obtain a quotient q_0 and remainder a_0, that is,

$n = bq_0 + a_0$, where $0 \le a_0 < b$.

The remainder a_0 is the rightmost digit in the base b expansion of n.

Next, divide q_0 by b to obtain:

$q_0 = bq_1 + a_1$, where $0 \le a_1 < b$.

a_1 is the second digit from the right in the base b expansion of n. Continue this process until you obtain a quotient equal to zero.

Representations of Integers

Example:

What is the base 8 expansion of $(12345)_{10}$?

First, divide 12345 by 8:

$12345 = 8\cdot1543 + 1$

$1543 = 8\cdot192 + 7$

$192 = 8\cdot24 + 0$

$24 = 8\cdot3 + 0$

$3 = 8\cdot0 + 3$

The result is: $(12345)_{10} = (30071)_8$.

Representations of Integers

procedure base_b_expansion(n, b: positive integers)
q := n
k := 0
while $q \neq 0$
begin
    a_k := q mod b
    q := $\lfloor q/b \rfloor$
    k := k + 1
end
{the base b expansion of n is $(a_{k-1} \ldots a_1 a_0)_b$ }

Addition of Integers

Let $a = (a_{n-1}a_{n-2}..a_1a_0)_2$, $b = (b_{n-1}b_{n-2}..b_1b_0)_2$.

How can we add these two binary numbers?

First, add their rightmost bits:

$a_0 + b_0 = c_0 \cdot 2 + s_0$,

where s_0 is the **rightmost bit** in the binary expansion of a + b, and c_0 is the **carry**.

Then, add the next pair of bits and the carry:

$a_1 + b_1 + c_0 = c_1 \cdot 2 + s_1$,

where s_1 is the **next bit** in the binary expansion of a + b, and c_1 is the carry.

Addition of Integers

Continue this process until you obtain c_{n-1}.

The leading bit of the sum is $s_n = c_{n-1}$.

The result is:

$a + b = (s_n s_{n-1}..s_1 s_0)_2$

Addition of Integers

Example:
Add $a = (1110)_2$ and $b = (1011)_2$.

$a_0 + b_0 = 0 + 1 = 0 \cdot 2 + 1$, so that $c_0 = 0$ and $s_0 = 1$.
$a_1 + b_1 + c_0 = 1 + 1 + 0 = 1 \cdot 2 + 0$, so $c_1 = 1$ and $s_1 = 0$.
$a_2 + b_2 + c_1 = 1 + 0 + 1 = 1 \cdot 2 + 0$, so $c_2 = 1$ and $s_2 = 0$.
$a_3 + b_3 + c_2 = 1 + 1 + 1 = 1 \cdot 2 + 1$, so $c_3 = 1$ and $s_3 = 1$.
$s_4 = c_3 = 1$.

Therefore, $s = a + b = (11001)_2$.

Addition of Integers

How do we (humans) add two integers?

Example:

$$\begin{array}{rl} 111 & \text{carry} \\ 7583 & \\ +4932 & \\ \hline 12515 & \end{array}$$

Binary expansions:

$$\begin{array}{rl} 1\;1 & \text{carry} \\ (1011)_2 & \\ +\,(1010)_2 & \\ \hline (10101)_2 & \end{array}$$

Addition of Integers

Let $a = (a_{n-1}a_{n-2}\ldots a_1a_0)_2$, $b = (b_{n-1}b_{n-2}\ldots b_1b_0)_2$.

How can we **algorithmically** add these two binary numbers?

First, add their rightmost bits:

$a_0 + b_0 = c_0 \cdot 2 + s_0$,
where s_0 is the **rightmost bit** in the binary expansion of $a + b$, and c_0 is the **carry**.

Then, add the next pair of bits and the carry:

$a_1 + b_1 + c_0 = c_1 \cdot 2 + s_1$,
where s_1 is the **next bit** in the binary expansion of $a + b$, and c_1 is the carry.

Addition of Integers

Continue this process until you obtain c_{n-1}.

The leading bit of the sum is $s_n = c_{n-1}$.

The result is:
$a + b = (s_n s_{n-1} .. s_1 s_0)_2$

Addition of Integers

Example:
Add $a = (1110)_2$ and $b = (1011)_2$.

$a_0 + b_0 = 0 + 1 = 0\cdot 2 + 1$, so that $c_0 = 0$ and $s_0 = 1$.
$a_1 + b_1 + c_0 = 1 + 1 + 0 = 1\cdot 2 + 0$, so $c_1 = 1$ and $s_1 = 0$.
$a_2 + b_2 + c_1 = 1 + 0 + 1 = 1\cdot 2 + 0$, so $c_2 = 1$ and $s_2 = 0$.
$a_3 + b_3 + c_2 = 1 + 1 + 1 = 1\cdot 2 + 1$, so $c_3 = 1$ and $s_3 = 1$.
$s_4 = c_3 = 1$.

Therefore, $s = a + b = (11001)_2$.

Addition of Integers

```
procedure add(a, b: positive integers)
c := 0
for j := 0 to n-1
begin
        d := ⌊(a_j + b_j + c)/2⌋
        s_j := a_j + b_j + c – 2d
        c := d
end
s_n := c
{the binary expansion of the sum is (s_n s_{n-1}..s_1 s_0)_2}
```

Let's proceed to…

Mathematical Reasoning

Mathematical Reasoning

We need **mathematical reasoning** to

- determine whether a mathematical argument is correct or incorrect and
- construct mathematical arguments.

Mathematical reasoning is not only important for conducting **proofs** and **program verification**, but also for **artificial intelligence** systems (drawing inferences).

Terminology

An **axiom** is a basic assumption about mathematical structured that needs no proof.

We can use a **proof** to demonstrate that a particular statement is true. A proof consists of a sequence of statements that form an argument.

The steps that connect the statements in such a sequence are the **rules of inference**.

Cases of incorrect reasoning are called **fallacies**.

A **theorem** is a statement that can be shown to be true.

Terminology

A **lemma** is a simple theorem used as an intermediate result in the proof of another theorem.

A **corollary** is a proposition that follows directly from a theorem that has been proved.

A **conjecture** is a statement whose truth value is unknown. Once it is proven, it becomes a theorem.

Rules of Inference

Rules of inference provide the justification of the steps used in a proof.

One important rule is called **modus ponens** or the **law of detachment**. It is based on the tautology $(p \wedge (p \rightarrow q)) \rightarrow q$. We write it in the following way:

$$\begin{array}{l} p \\ p \rightarrow q \\ \hline \therefore q \end{array}$$

The two **hypotheses** p and $p \rightarrow q$ are written in a column, and the **conclusion** below a bar, where $\therefore$ means "therefore".

Rules of Inference

The general form of a rule of inference is:

$$\begin{array}{l} p_1 \\ p_2 \\ \vdots \\ p_n \\ \hline \therefore q \end{array}$$

The rule states that if p_1 **and** p_2 **and** … **and** p_n are all true, then q is true as well.

These rules of inference can be used in any mathematical argument and do not require any proof.

Rules of Inference

p ———— $\therefore p \lor q$	Addition	$\neg q$ $p \to q$ ———— $\therefore \neg p$	Modus tollens
$p \land q$ ———— $\therefore p$	Simplification	$p \to q$ $q \to r$ ———— $\therefore p \to r$	Hypothetical syllogism
p q ———— $\therefore p \land q$	Conjunction	$p \lor q$ $\neg p$ ———— $\therefore q$	Disjunctive syllogism

Arguments

Just like a rule of inference, an **argument** consists of one or more hypotheses and a conclusion.

We say that an argument is **valid**, if whenever all its hypotheses are true, its conclusion is also true.

However, if any hypothesis is false, even a valid argument can lead to an incorrect conclusion.

Arguments

Example:

"If 101 is divisible by 3, then 101^2 is divisible by 9. 101 is divisible by 3. Consequently, 101^2 is divisible by 9."

Although the argument is **valid**, its conclusion is **incorrect**, because one of the hypotheses is false ("101 is divisible by 3.").

If in the above argument we replace 101 with 102, we could correctly conclude that 102^2 is divisible by 9.

Arguments

Which rule of inference was used in the last argument?

p: “101 is divisible by 3.”

q: “101^2 is divisible by 9.”

$$\begin{array}{l} p \\ p \rightarrow q \\ \hline \therefore q \end{array}$$

Modus ponens

Unfortunately, one of the hypotheses (p) is false. Therefore, the conclusion q is incorrect.

Arguments

Another example:

“If it rains today, then we will not have a barbeque today. If we do not have a barbeque today, then we will have a barbeque tomorrow. Therefore, if it rains today, then we will have a barbeque tomorrow.”

This is a **valid** argument: If its hypotheses are true, then its conclusion is also true.

Arguments

Let us formalize the previous argument:

p: “It is raining today.”

q: “We will not have a barbecue today.”

r: “We will have a barbecue tomorrow.”

So the argument is of the following form:

$$\begin{array}{l} p \rightarrow q \\ q \rightarrow r \\ \hline \therefore p \rightarrow r \end{array}$$

Hypothetical syllogism

Arguments

Another example:

Gary is either intelligent or a good actor.
If Gary is intelligent, then he can count from 1 to 10.
Gary can only count from 1 to 2.
Therefore, Gary is a good actor.

i: "Gary is intelligent."
a: "Gary is a good actor."
c: "Gary can count from 1 to 10."

Fall 2002 CMSC 203 - Discrete Structures 178

Arguments

i: "Gary is intelligent."
a: "Gary is a good actor."
c: "Gary can count from 1 to 10."

Step 1:	$\neg c$	Hypothesis
Step 2:	$i \to c$	Hypothesis
Step 3:	$\neg i$	Modus tollens Steps 1 & 2
Step 4:	$a \lor i$	Hypothesis
Step 5:	a	Disjunctive Syllogism Steps 3 & 4

Conclusion: **a** ("Gary is a good actor.")

Fall 2002 CMSC 203 - Discrete Structures 179

Arguments

Yet another example:

If you listen to me, you will pass CS 320.
You passed CS 320.
Therefore, you have listened to me.

Is this argument valid?

No, it assumes $((p \to q) \land q) \to p$.
This statement is not a tautology. It is false if p is false and q is true.

Fall 2002 CMSC 203 - Discrete Structures 180

Rules of Inference for Quantified Statements

$\dfrac{\forall x\ P(x)}{\therefore P(c) \text{ if } c \in U}$	Universal instantiation
$\dfrac{P(c) \text{ for an arbitrary } c \in U}{\therefore \forall x\ P(x)}$	Universal generalization
$\dfrac{\exists x\ P(x)}{\therefore P(c) \text{ for some element } c \in U}$	Existential instantiation
$\dfrac{P(c) \text{ for some element } c \in U}{\therefore \exists x\ P(x)}$	Existential generalization

Rules of Inference for Quantified Statements

Example:

Every UMB student is a genius.
George is a UMB student.
Therefore, George is a genius.

U(x): “x is a UMB student.”
G(x): “x is a genius.”

Rules of Inference for Quantified Statements

The following steps are used in the argument:

Step 1: $\forall x\ (U(x) \rightarrow G(x))$	Hypothesis
Step 2: $U(George) \rightarrow G(George)$	Univ. instantiation using Step 1
Step 3: $U(George)$	Hypothesis
Step 4: $G(George)$	Modus ponens using Steps 2 & 3

$\dfrac{\forall x\ P(x)}{\therefore P(c) \text{ if } c \in U}$ Universal instantiation

Proving Theorems

Direct proof:

An implication $p \rightarrow q$ can be proved by showing that if p is true, then q is also true.

Example: Give a direct proof of the theorem "If n is odd, then n^2 is odd."

Idea: Assume that the hypothesis of this implication is true (n is odd). Then use rules of inference and known theorems to show that q must also be true (n^2 is odd).

Proving Theorems

n is odd.

Then $n = 2k + 1$, where k is an integer.

Consequently, $n^2 = (2k + 1)^2$.

$$= 4k^2 + 4k + 1$$
$$= 2(2k^2 + 2k) + 1$$

Since n^2 can be written in this form, it is odd.

Proving Theorems

Indirect proof:

An implication $p \rightarrow q$ is equivalent to its **contra-positive** $\neg q \rightarrow \neg p$. Therefore, we can prove $p \rightarrow q$ by showing that whenever q is false, then p is also false.

Example: Give an indirect proof of the theorem "If $3n + 2$ is odd, then n is odd."

Idea: Assume that the conclusion of this implication is false (n is even). Then use rules of inference and known theorems to show that p must also be false ($3n + 2$ is even).

Proving Theorems

n is even.

Then n = 2k, where k is an integer.

It follows that $3n + 2 = 3(2k) + 2$
$= 6k + 2$
$= 2(3k + 1)$

Therefore, 3n + 2 is even.

We have shown that the contrapositive of the implication is true, so the implication itself is also true (If 2n + 3 is odd, then n is odd).

Follow me for a walk through...

Mathematical Induction

Induction

The **principle of mathematical induction** is a useful tool for proving that a certain predicate is true for **all natural numbers**.

It cannot be used to discover theorems, but only to prove them.

Induction

If we have a propositional function P(n), and we want to prove that P(n) is true for any natural number n, we do the following:

- Show that P(0) is true.
  (basis step)
- Show that if P(n) then P(n + 1) for any $n \in N$.
  (inductive step)
- Then P(n) must be true for any $n \in N$.
  (conclusion)

Induction

Example:

Show that $n < 2^n$ for all positive integers n.

Let P(n) be the proposition "$n < 2^n$."

1. Show that P(1) is true.
   (basis step)

P(1) is true, because $1 < 2^1 = 2$.

Induction

2. Show that if P(n) is true, then P(n + 1) is true.
   (inductive step)

Assume that $n < 2^n$ is true.
We need to show that P(n + 1) is true, i.e.
$n + 1 < 2^{n+1}$

We start from $n < 2^n$:
$n + 1 < 2^n + 1 \leq 2^n + 2^n = 2^{n+1}$
Therefore, if $n < 2^n$ then $n + 1 < 2^{n+1}$

Induction

- Then P(n) must be true for any positive integer.
  (conclusion)

$n < 2^n$ is true for any positive integer.

End of proof.

Induction

Another Example ("Gauss"):

$1 + 2 + \ldots + n = n (n + 1)/2$

- Show that P(0) is true.
  (basis step)

For $n = 0$ we get $0 = 0$. True.

Induction

- Show that if P(n) then P(n + 1) for any $n \in N$.
  (inductive step)

$1 + 2 + \ldots + n = n (n + 1)/2$

$1 + 2 + \ldots + n + (n + 1) = n (n + 1)/2 + (n + 1)$

$= (2n + 2 + n (n + 1))/2$

$= (2n + 2 + n^2 + n)/2$

$= (2 + 3n + n^2)/2$

$= (n + 1) (n + 2)/2$

$= (n + 1) ((n + 1) + 1)/2$

Induction

- Then P(n) must be true for any $n \in N$. (conclusion)

$1 + 2 + \ldots + n = n (n + 1)/2$ is true for all $n \in N$.

End of proof.

Induction

There is another proof technique that is very similar to the principle of mathematical induction.

It is called **the second principle of mathematical induction**.

It can be used to prove that a propositional function P(n) is true for any natural number n.

Induction

The second principle of mathematical induction:

- Show that P(0) is true.
  (basis step)
- Show that if P(0) and P(1) and …and P(n), then P(n + 1) for any $n \in N$.
  (inductive step)
- Then P(n) must be true for any $n \in N$.
  (conclusion)

Induction

Example: Show that every integer greater than 1 can be written as the product of primes.

- Show that P(2) is true.
  (basis step)

2 is the product of one prime: itself.

Induction

- Show that if P(2) and P(3) and …and P(n), then P(n + 1) for any $n \in N$. (inductive step)

Two possible cases:
- If (n + 1) is **prime**, then obviously P(n + 1) is true.
- If (n + 1) is **composite**, it can be written as the product of two integers a and b such that $2 \leq a \leq b < n + 1$.
  By the **induction hypothesis**, both a and b can be written as the product of primes.
  Therefore, $n + 1 = a \cdot b$ can be written as the product of primes.

Induction

- Then P(n) must be true for any $n \in N$.
  (conclusion)

End of proof.

We have shown that **every integer greater than 1** can be written as the product of primes.

If I told you once, it must be...

Recursion

Recursive Definitions

Recursion is a principle closely related to mathematical induction.

In a **recursive definition**, an object is defined in terms of itself.

We can recursively define **sequences**, **functions** and **sets**.

Recursively Defined Sequences

Example:

The sequence $\{a_n\}$ of powers of 2 is given by $a_n = 2^n$ for $n = 0, 1, 2, \ldots$.

The same sequence can also be defined **recursively**:

$a_0 = 1$
$a_{n+1} = 2a_n$ for $n = 0, 1, 2, \ldots$

Obviously, induction and recursion are similar principles.

Recursively Defined Functions

We can use the following method to define a function with the **natural numbers** as its domain:

- Specify the value of the function at zero.
- Give a rule for finding its value at any integer from its values at smaller integers.

Such a definition is called **recursive** or **inductive definition**.

Recursively Defined Functions

Example:

$f(0) = 3$
$f(n + 1) = 2f(n) + 3$

$f(0) = 3$
$f(1) = 2f(0) + 3 = 2\cdot 3 + 3 = 9$
$f(2) = 2f(1) + 3 = 2\cdot 9 + 3 = 21$
$f(3) = 2f(2) + 3 = 2\cdot 21 + 3 = 45$
$f(4) = 2f(3) + 3 = 2\cdot 45 + 3 = 93$

Recursively Defined Functions

How can we recursively define the factorial function $f(n) = n!$?

$f(0) = 1$
$f(n + 1) = (n + 1)f(n)$

$f(0) = 1$
$f(1) = 1f(0) = 1\cdot 1 = 1$
$f(2) = 2f(1) = 2\cdot 1 = 2$
$f(3) = 3f(2) = 3\cdot 2 = 6$
$f(4) = 4f(3) = 4\cdot 6 = 24$

Recursively Defined Functions

A famous example: The Fibonacci numbers

f(0) = 0, f(1) = 1
f(n) = f(n – 1) + f(n - 2)

f(0) = 0
f(1) = 1
f(2) = f(1) + f(0) = 1 + 0 = 1
f(3) = f(2) + f(1) = 1 + 1 = 2
f(4) = f(3) + f(2) = 2 + 1 = 3
f(5) = f(4) + f(3) = 3 + 2 = 5
f(6) = f(5) + f(4) = 5 + 3 = 8

Recursively Defined Sets

If we want to recursively define a set, we need to provide two things:

- an **initial set** of elements,
- **rules** for the construction of **additional** elements from elements in the set.

Example: Let S be recursively defined by:
$3 \in S$
$(x + y) \in S$ if $(x \in S)$ and $(y \in S)$

S is the set of positive integers divisible by 3.

Recursively Defined Sets

Proof:

Let A be the set of all positive integers divisible by 3.

To show that A = S, we must show that $A \subseteq S$ and $S \subseteq A$.

Part I: To prove that $A \subseteq S$, we must show that every positive integer divisible by 3 is in S.

We will use mathematical induction to show this.

Recursively Defined Sets

Let P(n) be the statement "3n belongs to S".

Basis step: P(1) is true, because 3 is in S.

Inductive step: To show:
If P(n) is true, then P(n + 1) is true.

Assume 3n is in S. Since 3n is in S and 3 is in S, it follows from the recursive definition of S that $3n + 3 = 3(n + 1)$ is also in S.

Conclusion of Part I: $A \subseteq S$.

Fall 2002 CMSC 203 - Discrete Structures 211

Recursively Defined Sets

Part II: To show: $S \subseteq A$.

Basis step: To show:
All initial elements of S are in A. 3 is in A. True.

Inductive step: To show:
$(x + y)$ is in A whenever x and y are in S.

If x and y are both in A, it follows that $3 \mid x$ and $3 \mid y$. From Theorem I, Section 2.3, it follows that $3 \mid (x + y)$.

Conclusion of Part II: $S \subseteq A$.
Overall conclusion: $A = S$.

Fall 2002 CMSC 203 - Discrete Structures 212

Recursively Defined Sets

Another example:

The well-formed formulae of variables, numerals and operators from {+, -, *, /, ^} are defined by:

x is a well-formed formula if x is a numeral or variable.

(f + g), (f – g), (f * g), (f / g), (f ^ g) are well-formed formulae if f and g are.

Fall 2002 CMSC 203 - Discrete Structures 213

Recursively Defined Sets

With this definition, we can construct formulae such as:

(x – y)
((z / 3) – y)
((z / 3) – (6 + 5))
((z / (2 * 4)) – (6 + 5))

Recursive Algorithms

An algorithm is called **recursive** if it solves a problem by reducing it to an instance of the same problem with smaller input.

Example I: Recursive Euclidean Algorithm

procedure gcd(a, b: nonnegative integers with a < b)
if a = 0 **then** gcd(a, b) := b
else gcd(a, b) := gcd(b **mod** a, a)

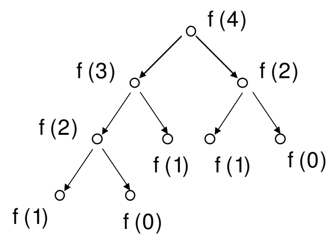
Recursive Algorithms

Example II: Recursive Fibonacci Algorithm

procedure fibo(n: nonnegative integer)
if n = 0 **then** fibo(0) := 0
else if n = 1 **then** fibo(1) := 1
else fibo(n) := fibo(n – 1) + fibo(n – 2)

Recursive Algorithms

Recursive Fibonacci Evaluation:



Recursive Algorithms

procedure iterative_fibo(n: nonnegative integer)
if n = 0 **then** y := 0
else
begin
 x := 0
 y := 1
 for i := 1 **to** n-1
 begin
  z := x + y
  x : = y
  y := z
 end
end {y is the n-th Fibonacci number}

Recursive Algorithms

For every recursive algorithm, there is an equivalent iterative algorithm.

Recursive algorithms are often shorter, more elegant, and easier to understand than their iterative counterparts.

However, iterative algorithms are usually more efficient in their use of space and time.

One, two, three, we're…

Counting

Fall 2002 CMSC 203 - Discrete Structures 220

Basic Counting Principles

Counting problems are of the following kind:

"**How many** different 8-letter passwords are there?"

"**How many** possible ways are there to pick 11 soccer players out of a 20-player team?"

Most importantly, counting is the basis for computing **probabilities of discrete events**.

("What is the probability of winning the lottery?")

Fall 2002 CMSC 203 - Discrete Structures 221

Basic Counting Principles

The sum rule:
If a task can be done in n_1 ways and a second task in n_2 ways, and if these two tasks cannot be done at the same time, then there are $n_1 + n_2$ ways to do either task.

Example:
The department will award a free computer to either a CS student or a CS professor.
How many different choices are there, if there are 530 students and 15 professors?

There are 530 + 15 = 545 choices.

Fall 2002 CMSC 203 - Discrete Structures 222

Basic Counting Principles

Generalized sum rule:

If we have tasks $T_1, T_2, .., T_m$ that can be done in $n_1, n_2, .., n_m$ ways, respectively, and no two of these tasks can be done at the same time, then there are $n_1 + n_2 + \ldots + n_m$ ways to do one of these tasks.

Basic Counting Principles

The product rule:

Suppose that a procedure can be broken down into two successive tasks. If there are n_1 ways to do the first task and n_2 ways to do the second task after the first task has been done, then there are $n_1 n_2$ ways to do the procedure.

Basic Counting Principles

Example:

How many different license plates are there that containing exactly three English letters ?

Solution:

There are 26 possibilities to pick the first letter, then 26 possibilities for the second one, and 26 for the last one.

So there are $26 \cdot 26 \cdot 26 = 17576$ different license plates.

Basic Counting Principles

Generalized product rule:

If we have a procedure consisting of sequential tasks $T_1, T_2, .., T_m$ that can be done in $n_1, n_2, .., n_m$ ways, respectively, then there are $n_1 \cdot n_2 \cdot \ldots \cdot n_m$ ways to carry out the procedure.

Basic Counting Principles

The sum and product rules can also be phrased in terms of **set theory**.

Sum rule: Let $A_1, A_2, .., A_m$ be disjoint sets. Then the number of ways to choose any element from one of these sets is $|A_1 \cup A_2 \cup \ldots \cup A_m| = |A_1| + |A_2| + \ldots + |A_m|$.

Product rule: Let $A_1, A_2, .., A_m$ be finite sets. Then the number of ways to choose one element from each set in the order $A_1, A_2, .., A_m$ is $|A_1 \times A_2 \times \ldots \times A_m| = |A_1| \cdot |A_2| \cdot \ldots \cdot |A_m|$.

Inclusion-Exclusion

How many bit strings of length 8 either start with a 1 or end with 00?

Task 1: Construct a string of length 8 that starts with a 1.

There is one way to pick the first bit (1),
two ways to pick the second bit (0 or 1),
two ways to pick the third bit (0 or 1),
.
.
.
two ways to pick the eighth bit (0 or 1).

Product rule: Task 1 can be done in $1 \cdot 2^7 = 128$ ways.

Inclusion-Exclusion

Task 2: Construct a string of length 8 that ends with 00.

There are two ways to pick the first bit (0 or 1),
two ways to pick the second bit (0 or 1),
.
.
.
two ways to pick the sixth bit (0 or 1),
one way to pick the seventh bit (0), and
one way to pick the eighth bit (0).

Product rule: Task 2 can be done in $2^6 = 64$ ways.

Inclusion-Exclusion

Since there are 128 ways to do Task 1 and 64 ways to do Task 2, does this mean that there are 192 bit strings either starting with 1 or ending with 00 ?

No, because here Task 1 and Task 2 can be done **at the same time**.

When we carry out Task 1 and create strings starting with 1, some of these strings end with 00.

Therefore, we sometimes do Tasks 1 and 2 at the same time, so **the sum rule does not apply**.

Inclusion-Exclusion

If we want to use the sum rule in such a case, we have to subtract the cases when Tasks 1 and 2 are done at the same time.

How many cases are there, that is, how many strings start with 1 **and** end with 00?

There is one way to pick the first bit (1),
two ways for the second, .., sixth bit (0 or 1),
one way for the seventh, eighth bit (0).

Product rule: In $2^5 = 32$ cases, Tasks 1 and 2 are carried out at the same time.

Inclusion-Exclusion

Since there are 128 ways to complete Task 1 and 64 ways to complete Task 2, and in 32 of these cases Tasks 1 and 2 are completed at the same time, there are

$128 + 64 - 32 = 160$ ways to do either task.

In set theory, this corresponds to sets A_1 and A_2 that are **not** disjoint. Then we have:

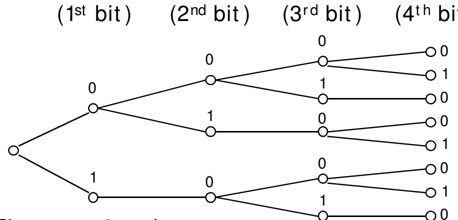
$|A_1 \cup A_2| = |A_1| + |A_2| - |A_1 \cap A_2|$

This is called the **principle of inclusion-exclusion.**

Tree Diagrams

How many bit strings of length four do not have two consecutive 1s?

Task 1 (1st bit) Task 2 (2nd bit) Task 3 (3rd bit) Task 4 (4th bit)



There are 8 strings.

The Pigeonhole Principle

The pigeonhole principle: If $(k + 1)$ or more objects are placed into k boxes, then there is at least one box containing two or more of the objects.

Example 1: If there are 11 players in a soccer team that wins 12-0, there must be at least one player in the team who scored at least twice.

Example 2: If you have 6 classes from Monday to Friday, there must be at least one day on which you have at least two classes.

The Pigeonhole Principle

The generalized pigeonhole principle: If N objects are placed into k boxes, then there is at least one box containing at least $\lceil N/k \rceil$ of the objects.

Example 1: In our 60-student class, at least 12 students will get the same letter grade (A, B, C, D, or F).

The Pigeonhole Principle

Example 2: Assume you have a drawer containing a random distribution of a dozen brown socks and a dozen black socks. It is dark, so how many socks do you have to pick to be sure that among them there is a matching pair?

There are two types of socks, so if you pick at least 3 socks, there must be either at least two brown socks or at least two black socks.

Generalized pigeonhole principle: $\lceil 3/2 \rceil = 2$.

Permutations and Combinations

How many ways are there to pick a set of 3 people from a group of 6?

There are 6 choices for the first person, 5 for the second one, and 4 for the third one, so there are $6 \cdot 5 \cdot 4 = 120$ ways to do this.

This is not the correct result!

For example, picking person C, then person A, and then person E leads to the **same group** as first picking E, then C, and then A.

However, these cases are counted **separately** in the above equation.

Permutations and Combinations

So how can we compute how many different subsets of people can be picked (that is, we want to disregard the order of picking)?

To find out about this, we need to look at **permutations.**

A **permutation** of a set of distinct objects is an ordered arrangement of these objects.

An ordered arrangement of r elements of a set is called an **r-permutation**.

Permutations and Combinations

Example: Let $S = \{1, 2, 3\}$.
The arrangement 3, 1, 2 is a permutation of S.
The arrangement 3, 2 is a 2-permutation of S.

The number of r-permutations of a set with n distinct elements is denoted by **P(n, r).**

We can calculate P(n, r) with the product rule:

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

(n choices for the first element, (n – 1) for the second one, (n – 2) for the third one..)

Permutations and Combinations

Example:

$P(8, 3) = 8\cdot7\cdot6 = 336$
$= (8\cdot7\cdot6\cdot5\cdot4\cdot3\cdot2\cdot1)/(5\cdot4\cdot3\cdot2\cdot1)$

General formula:

$P(n, r) = n!/(n-r)!$

Knowing this, we can return to our initial question:

How many ways are there to pick a set of 3 people from a group of 6 (disregarding the order of picking)?

Permutations and Combinations

An **r-combination** of elements of a set is an unordered selection of r elements from the set. Thus, an r-combination is simply a subset of the set with r elements.

Example: Let S = {1, 2, 3, 4}.
Then {1, 3, 4} is a 3-combination from S.

The number of r-combinations of a set with n distinct elements is denoted by C(n, r).

Example: C(4, 2) = 6, since, for example, the 2-combinations of a set {1, 2, 3, 4} are {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}.

Permutations and Combinations

How can we calculate C(n, r)?

Consider that we can obtain the r-permutation of a set in the following way:

First, we form all the r-combinations of the set (there are C(n, r) such r-combinations).

Then, we generate all possible orderings in each of these r-combinations (there are P(r, r) such orderings in each case).

Therefore, we have:

$P(n, r) = C(n, r) \cdot P(r, r)$

Permutations and Combinations

$$\begin{aligned} C(n, r) &= P(n, r)/P(r, r) \\ &= n!/(n-r)!/(r!/(r-r)!) \\ &= n!/(r!(n-r)!) \end{aligned}$$

Now we can answer our initial question:

How many ways are there to pick a set of 3 people from a group of 6 (disregarding the order of picking)?

$C(6, 3) = 6!/(3!\cdot 3!) = 720/(6\cdot 6) = 720/36 = 20$

There are 20 different ways, that is, 20 different groups to be picked.

Permutations and Combinations

Corollary:
Let n and r be nonnegative integers with $r \leq n$.
Then $C(n, r) = C(n, n - r)$.

Note that **"picking a group of r people from a group of n people"** is the same as **"splitting a group of n people into a group of r people and another group of (n – r) people".**

Please also look at proof on page 252.

Permutations and Combinations

Example:
A soccer club has 8 female and 7 male members. For today's match, the coach wants to have 6 female and 5 male players on the grass. How many possible configurations are there?

$$C(8, 6) \cdot C(7, 5) = 8!/(6!\cdot 2!) \cdot 7!/(5!\cdot 2!) = 28\cdot 21 = 588$$

Combinations

We also saw the following:

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

This symmetry is intuitively plausible. For example, let us consider a set containing six elements (n = 6).

Picking two elements and **leaving four** is essentially the same as **picking four** elements and **leaving two**.

In either case, our number of choices is the number of possibilities to **divide** the set into one set containing two elements and another set containing four elements.

Combinations

Pascal's Identity:

Let n and k be positive integers with $n \geq k$.
Then $C(n + 1, k) = C(n, k - 1) + C(n, k)$.

How can this be explained?

What is it good for?

Combinations

Imagine a set S containing n elements and a set T containing (n + 1) elements, namely all elements in S plus a new element **a**.

Calculating $C(n + 1, k)$ is equivalent to answering the question: How many subsets of T containing k items are there?

Case I: The subset contains (k – 1) elements of S plus the element **a**: $C(n, k - 1)$ choices.

Case II: The subset contains k elements of S and does not contain **a**: $C(n, k)$ choices.

Sum Rule: $C(n + 1, k) = C(n, k - 1) + C(n, k)$.

Pascal's Triangle

In Pascal's triangle, each number is the sum of the numbers to its upper left and upper right:

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
…   …   …   …   …   …
```

Pascal's Triangle

Since we have C(n + 1, k) = C(n, k – 1) + C(n, k) and C(0, 0) = 1, we can use Pascal's triangle to simplify the computation of C(n, k):

k

n

C(0, 0) = **1**

C(1, 0) = **1** C(1, 1) = **1**

C(2, 0) = **1** C(2, 1) = **2** C(2, 2) = **1**

C(3, 0) = **1** C(3, 1) = **3** C(3, 2) = **3** C(3, 3) = **1**

C(4, 0) = **1** C(4, 1) = **4** C(4, 2) = **6** C(4, 3) = **4** C(4, 4) = **1**

Binomial Coefficients

Expressions of the form C(n, k) are also called **binomial coefficients**.

How come?

A **binomial expression** is the sum of two terms, such as (a + b).

Now consider $(a + b)^2 = (a + b)(a + b)$.

When expanding such expressions, we have to form all possible products of a term in the first factor and a term in the second factor:

$(a + b)^2 = a \cdot a + a \cdot b + b \cdot a + b \cdot b$

Then we can sum identical terms:

$(a + b)^2 = a^2 + 2ab + b^2$

Binomial Coefficients

For $(a + b)^3 = (a + b)(a + b)(a + b)$ we have

$(a + b)^3 = aaa + aab + aba + abb + baa + bab + bba + bbb$

$(a + b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$

There is only one term a^3, because there is only one possibility to form it: Choose **a** from all three factors: C(3, 3) = 1.

There is three times the term a^2b, because there are three possibilities to choose **a** from two out of the three factors: C(3, 2) = 3.

Similarly, there is three times the term ab^2 (C(3, 1) = 3) and once the term b^3 (C(3, 0) = 1).

Binomial Coefficients

This leads us to the following formula:

$$(a+b)^n = \sum_{j=0}^{n} C(n,j) \cdot a^{n-j} b^j$$ (Binomial Theorem)

With the help of Pascal's triangle, this formula can considerably simplify the process of expanding powers of binomial expressions.

For example, the fifth row of Pascal's triangle (1 – 4 – 6 – 4 – 1) helps us to compute $(a + b)^4$:

$(a + b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$

Now it's Time for…

Recurrence Relations

Recurrence Relations

A **recurrence relation** for the sequence $\{a_n\}$ is an equation that expresses a_n is terms of one or more of the previous terms of the sequence, namely, $a_0, a_1, .., a_{n-1}$, for all integers n with $n \geq n_0$, where n_0 is a nonnegative integer.

A sequence is called a **solution** of a recurrence relation if it terms satisfy the recurrence relation.

Recurrence Relations

In other words, a recurrence relation is like a recursively defined sequence, but **without specifying any initial values (initial conditions)**.

Therefore, the same recurrence relation can have (and usually has) **multiple solutions**.

If **both** the initial conditions and the recurrence relation are specified, then the sequence is **uniquely** determined.

Recurrence Relations

Example:
Consider the recurrence relation
$a_n = 2a_{n-1} - a_{n-2}$ for $n = 2, 3, 4, \ldots$

Is the sequence $\{a_n\}$ with $a_n=3n$ a solution of this recurrence relation?
For $n \geq 2$ we see that
$2a_{n-1} - a_{n-2} = 2(3(n-1)) - 3(n-2) = 3n = a_n$.
Therefore, $\{a_n\}$ with $a_n=3n$ is a solution of the recurrence relation.

Recurrence Relations

Is the sequence $\{a_n\}$ with $a_n=5$ a solution of the same recurrence relation?
For $n \geq 2$ we see that
$2a_{n-1} - a_{n-2} = 2 \cdot 5 - 5 = 5 = a_n$.

Therefore, $\{a_n\}$ with $a_n=5$ is also a solution of the recurrence relation.

Modeling with Recurrence Relations

Example:

Someone deposits $10,000 in a savings account at a bank yielding 5% per year with interest compounded annually. How much money will be in the account after 30 years?

Solution:

Let P_n denote the amount in the account after n years.

How can we determine P_n on the basis of P_{n-1}?

Modeling with Recurrence Relations

We can derive the following **recurrence relation**:

$P_n = P_{n-1} + 0.05P_{n-1} = 1.05P_{n-1}$.

The initial condition is $P_0 = 10,000$.

Then we have:

$P_1 = 1.05P_0$

$P_2 = 1.05P_1 = (1.05)^2P_0$

$P_3 = 1.05P_2 = (1.05)^3P_0$

…

$P_n = 1.05P_{n-1} = (1.05)^nP_0$

We now have a **formula** to calculate P_n for any natural number n and can avoid the iteration.

Modeling with Recurrence Relations

Let us use this formula to find P_{30} under the initial condition $P_0 = 10,000$:

$P_{30} = (1.05)^{30} \cdot 10,000 = 43,219.42$

After 30 years, the account contains $43,219.42.

Modeling with Recurrence Relations

Another example:

Let a_n denote the number of bit strings of length n that do not have two consecutive 0s ("valid strings"). Find a recurrence relation and give initial conditions for the sequence $\{a_n\}$.

Solution:

Idea: The number of valid strings equals the number of valid strings ending with a 0 plus the number of valid strings ending with a 1.

Modeling with Recurrence Relations

Let us assume that $n \geq 3$, so that the string contains at least 3 bits.

Let us further assume that we know the number a_{n-1} of valid strings of length $(n - 1)$.

Then how many valid strings of length n are there, if the string ends with a 1?

There are a_{n-1} such strings, namely the set of valid strings of length $(n - 1)$ with a 1 appended to them.

Note: Whenever we append a 1 to a valid string, that string remains valid.

Modeling with Recurrence Relations

Now we need to know: How many valid strings of length n are there, if the string ends with a **0**?

Valid strings of length n ending with a 0 **must have a 1 as their (n – 1)st bit** (otherwise they would end with 00 and would not be valid).

And what is the number of valid strings of length $(n - 1)$ that end with a 1?

We already know that there are a_{n-1} strings of length n that end with a 1.

Therefore, there are a_{n-2} strings of length $(n - 1)$ that end with a 1.

Modeling with Recurrence Relations

So there are a_{n-2} valid strings of length n that end with a 0 (all valid strings of length (n – 2) with 10 appended to them).

As we said before, the number of valid strings is the number of valid strings ending with a 0 plus the number of valid strings ending with a 1.

That gives us the following **recurrence relation**:

$a_n = a_{n-1} + a_{n-2}$

Modeling with Recurrence Relations

What are the **initial conditions**?

$a_1 = 2$ (0 and 1)

$a_2 = 3$ (01, 10, and 11)

$a_3 = a_2 + a_1 = 3 + 2 = 5$

$a_4 = a_3 + a_2 = 5 + 3 = 8$

$a_5 = a_4 + a_3 = 8 + 5 = 13$

…

This sequence satisfies the same recurrence relation as the **Fibonacci sequence**.

Since $a_1 = f_3$ and $a_2 = f_4$, we have $a_n = f_{n+2}$.

Solving Recurrence Relations

In general, we would prefer to have an **explicit formula** to compute the value of a_n rather than conducting n iterations.

For one class of recurrence relations, we can obtain such formulas in a systematic way.

Those are the recurrence relations that express the terms of a sequence as **linear combinations** of previous terms.

Solving Recurrence Relations

Definition: A linear homogeneous recurrence relation of degree k with constant coefficients is a recurrence relation of the form:

$a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}$,

Where $c_1, c_2, .., c_k$ are real numbers, and $c_k \neq 0$.

A sequence satisfying such a recurrence relation is uniquely determined by the recurrence relation and the k initial conditions

$a_0 = C_0, a_1 = C_1, a_2 = C_2, .., a_{k-1} = C_{k-1}$.

Solving Recurrence Relations

Examples:

The recurrence relation $P_n = (1.05)P_{n-1}$ is a linear homogeneous recurrence relation of **degree one**.

The recurrence relation $f_n = f_{n-1} + f_{n-2}$ is a linear homogeneous recurrence relation of **degree two**.

The recurrence relation $a_n = a_{n-5}$ is a linear homogeneous recurrence relation of **degree five**.

Solving Recurrence Relations

Basically, when solving such recurrence relations, we try to find solutions of the form $\mathbf{a_n = r^n}$, where r is a constant.

$a_n = r^n$ is a solution of the recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}$ if and only if

$r^n = c_1r^{n-1} + c_2r^{n-2} + \ldots + c_kr^{n-k}$.

Divide this equation by r^{n-k} and subtract the right-hand side from the left:

$r^k - c_1r^{k-1} - c_2r^{k-2} - \ldots - c_{k-1}r - c_k = 0$

This is called the **characteristic equation** of the recurrence relation.

Solving Recurrence Relations

The solutions of this equation are called the **characteristic roots** of the recurrence relation.

Let us consider linear homogeneous recurrence relations of **degree two**.

Theorem: Let c_1 and c_2 be real numbers. Suppose that $r^2 - c_1 r - c_2 = 0$ has two distinct roots r_1 and r_2. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ if and only if $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for $n = 0, 1, 2, \ldots$, where α_1 and α_2 are constants.

See pp. 321 and 322 for the proof.

Solving Recurrence Relations

Example: What is the solution of the recurrence relation $a_n = a_{n-1} + 2a_{n-2}$ with $a_0 = 2$ and $a_1 = 7$?

Solution: The characteristic equation of the recurrence relation is $r^2 - r - 2 = 0$.

Its roots are $r = 2$ and $r = -1$.

Hence, the sequence $\{a_n\}$ is a solution to the recurrence relation if and only if:

$a_n = \alpha_1 2^n + \alpha_2 (-1)^n$ for some constants α_1 and α_2.

Solving Recurrence Relations

Given the equation $a_n = \alpha_1 2^n + \alpha_2 (-1)^n$ and the initial conditions $a_0 = 2$ and $a_1 = 7$, it follows that

$a_0 = 2 = \alpha_1 + \alpha_2$

$a_1 = 7 = \alpha_1 \cdot 2 + \alpha_2 \cdot (-1)$

Solving these two equations gives us $\alpha_1 = 3$ and $\alpha_2 = -1$.

Therefore, the solution to the recurrence relation and initial conditions is the sequence $\{a_n\}$ with $a_n = 3 \cdot 2^n - (-1)^n$.

Solving Recurrence Relations

$a_n = r^n$ is a solution of the linear homogeneous recurrence relation
$a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}$
if and only if
$r^n = c_1r^{n-1} + c_2r^{n-2} + \ldots + c_kr^{n-k}$.
Divide this equation by r^{n-k} and subtract the right-hand side from the left:
$r^k - c_1r^{k-1} - c_2r^{k-2} - \ldots - c_{k-1}r - c_k = 0$
This is called the **characteristic equation** of the recurrence relation.

Fall 2002 CMSC 203 - Discrete Structures 274

Solving Recurrence Relations

The solutions of this equation are called the **characteristic roots** of the recurrence relation.

Let us consider linear homogeneous recurrence relations of **degree two**.

Theorem: Let c_1 and c_2 be real numbers. Suppose that $r^2 - c_1r - c_2 = 0$ has two distinct roots r_1 and r_2. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2}$ if and only if $a_n = \alpha_1r_1^n + \alpha_2r_2^n$ for $n = 0, 1, 2, \ldots$, where α_1 and α_2 are constants.

See pp. 321 and 322 for the proof.

Fall 2002 CMSC 203 - Discrete Structures 275

Solving Recurrence Relations

Example: Give an explicit formula for the Fibonacci numbers.

Solution: The Fibonacci numbers satisfy the recurrence relation $f_n = f_{n-1} + f_{n-2}$ with initial conditions $f_0 = 0$ and $f_1 = 1$.

The characteristic equation is $r^2 - r - 1 = 0$.

Its roots are

$$r_1 = \frac{1+\sqrt{5}}{2}, \quad r_2 = \frac{1-\sqrt{5}}{2}$$

Fall 2002 CMSC 203 - Discrete Structures 276

Solving Recurrence Relations

Therefore, the Fibonacci numbers are given by

$$f_n = \alpha_1 \left(\frac{1+\sqrt{5}}{2} \right)^n + \alpha_2 \left(\frac{1-\sqrt{5}}{2} \right)^n$$

for some constants α_1 and α_2.

We can determine values for these constants so that the sequence meets the conditions $f_0 = 0$ and $f_1 = 1$:

$$f_0 = \alpha_1 + \alpha_2 = 0$$

$$f_1 = {}_1 \left(\frac{1+\sqrt{5}}{2} \right) + \alpha_2 \left(\frac{1-\sqrt{5}}{2} \right) = 1$$

Solving Recurrence Relations

The unique solution to this system of two equations and two variables is

$$\alpha = \frac{1}{\sqrt{5}}, \ \alpha_2 = \frac{1}{\sqrt{5}}$$

Fibonacci numbers:

$$f_n = \frac{1}{\sqrt{}} \left(\frac{1+\sqrt{5}}{} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{-\sqrt{5}}{2} \right)^n$$

Solving Recurrence Relations

But what happens if the characteristic equation has only one root?

How can we then match our equation with the initial conditions a_0 and a_1?

Theorem: Let c_1 and c_2 be real numbers with $c_2 \neq 0$. Suppose that $r^2 - c_1 r - c_2 = 0$ has only one root r_0. A sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ if and only if $a_n = \alpha_1 r_0^n + \alpha_2 n r_0^n$, for $n = 0, 1, 2, ..$, where α_1 and α_2 are constants.

Solving Recurrence Relations

Example: What is the solution of the recurrence relation $a_n = 6a_{n-1} - 9a_{n-2}$ with $a_0 = 1$ and $a_1 = 6$?

Solution: The only root of $r^2 - 6r + 9 = 0$ is $r_0 = 3$. Hence, the solution to the recurrence relation is

$a_n = \alpha_1 3^n + \alpha_2 n3^n$ for some constants α_1 and α_2.

To match the initial condition, we need

$a_0 = 1 = \alpha_1$

$a_1 = 6 = \alpha_1 \cdot 3 + \alpha_2 \cdot 3$

Solving these equations yields $\alpha_1 = 1$ and $\alpha_2 = 1$.

Consequently, the overall solution is given by

$a_n = 3^n + n3^n$.

You Never Escape Your …

Relations

Relations

If we want to describe a relationship between elements of two sets A and B, we can use **ordered pairs** with their first element taken from A and their second element taken from B.

Since this is a relation between **two sets**, it is called a **binary relation**.

Definition: Let A and B be sets. A binary relation from A to B is a subset of $A \times B$.

In other words, for a binary relation R we have $R \subseteq A \times B$. We use the notation aRb to denote that $(a, b) \in R$ and $a\underline{R}b$ to denote that $(a, b) \notin R$.

Relations

When (a, b) belongs to R, a is said to be **related** to b by R.

Example: Let P be a set of people, C be a set of cars, and D be the relation describing which person drives which car(s).

P = {Carl, Suzanne, Peter, Carla},

C = {Mercedes, BMW, tricycle}

D = {(Carl, Mercedes), (Suzanne, Mercedes), (Suzanne, BMW), (Peter, tricycle)}

This means that Carl drives a Mercedes, Suzanne drives a Mercedes and a BMW, Peter drives a tricycle, and Carla does not drive any of these vehicles.

Functions as Relations

You might remember that a **function** f from a set A to a set B assigns a unique element of B to each element of A.

The **graph** of f is the set of ordered pairs (a, b) such that b = f(a).

Since the graph of f is a subset of $A \times B$, it is a **relation** from A to B.

Moreover, for each element **a** of A, there is exactly one ordered pair in the graph that has **a** as its first element.

Functions as Relations

Conversely, if R is a relation from A to B such that every element in A is the first element of exactly one ordered pair of R, then a function can be defined with R as its graph.

This is done by assigning to an element $a \in A$ the unique element $b \in B$ such that $(a, b) \in R$.

Relations on a Set

Definition: A relation on the set A is a relation from A to A.

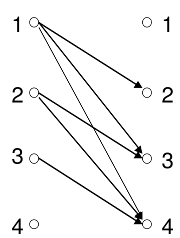
In other words, a relation on the set A is a subset of $A \times A$.

Example: Let $A = \{1, 2, 3, 4\}$. Which ordered pairs are in the relation $R = \{(a, b) \mid a < b\}$?

Fall 2002 CMSC 203 - Discrete Structures 286

Relations on a Set

Solution: $R = \{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\}$



R	1	2	3	4
1		X	X	X
2			X	X
3				X
4				

Fall 2002 CMSC 203 - Discrete Structures 287

Relations on a Set

How many different relations can we define on a set A with n elements?

A relation on a set A is a subset of $A \times A$.
How many elements are in $A \times A$?

There are n^2 elements in $A \times A$, so how many subsets (= relations on A) does $A \times A$ have?

The number of subsets that we can form out of a set with m elements is 2^m. Therefore, 2^{n^2} subsets can be formed out of $A \times A$.

Answer: We can define 2^{n^2} different relations on A.

Fall 2002 CMSC 203 - Discrete Structures 288

Properties of Relations

We will now look at some useful ways to classify relations.

Definition: A relation R on a set A is called **reflexive** if $(a, a) \in R$ for every element $a \in A$.

Are the following relations on {1, 2, 3, 4} reflexive?

R = {(1, 1), (1, 2), (2, 3), (3, 3), (4, 4)}	No.
R = {(1, 1), (2, 2), (2, 3), (3, 3), (4, 4)}	Yes.
R = {(1, 1), (2, 2), (3, 3)}	No.

Definition: A relation on a set A is called **irreflexive** if $(a, a) \notin R$ for every element $a \in A$.

Properties of Relations

Definitions:

A relation R on a set A is called **symmetric** if $(b, a) \in R$ whenever $(a, b) \in R$ for all $a, b \in A$.

A relation R on a set A is called **antisymmetric** if $a = b$ whenever $(a, b) \in R$ and $(b, a) \in R$.

A relation R on a set A is called **asymmetric** if $(a, b) \in R$ implies that $(b, a) \notin R$ for all $a, b \in A$.

Properties of Relations

Are the following relations on {1, 2, 3, 4} symmetric, antisymmetric, or asymmetric?

R = {(1, 1), (1, 2), (2, 1), (3, 3), (4, 4)}	symmetric
R = {(1, 1)}	sym. and antisym.
R = {(1, 3), (3, 2), (2, 1)}	antisym. and asym.
R = {(4, 4), (3, 3), (1, 4)}	antisym.

Properties of Relations

Definition: A relation R on a set A is called **transitive** if whenever $(a, b) \in R$ and $(b, c) \in R$, then $(a, c) \in R$ for $a, b, c \in A$.

Are the following relations on {1, 2, 3, 4} transitive?

R = {(1, 1), (1, 2), (2, 2), (2, 1), (3, 3)} Yes.

R = {(1, 3), (3, 2), (2, 1)} No.

R = {(2, 4), (4, 3), (2, 3), (4, 1)} No.

Counting Relations

Example: How many different reflexive relations can be defined on a set A containing n elements?

Solution: Relations on R are subsets of $A \times A$, which contains n^2 elements.

Therefore, different relations on A can be generated by choosing different subsets out of these n^2 elements, so there are 2^{n^2} relations.

A **reflexive** relation, however, **must** contain the n elements (a, a) for every $a \in A$.

Consequently, we can only choose among $n^2 - n = n(n - 1)$ elements to generate reflexive relations, so there are $2^{n(n-1)}$ of them.

Combining Relations

Relations are sets, and therefore, we can apply the usual **set operations** to them.

If we have two relations R_1 and R_2, and both of them are from a set A to a set B, then we can combine them to $R_1 \cup R_2$, $R_1 \cap R_2$, or $R_1 - R_2$.

In each case, the result will be **another relation from A to B**.

Combining Relations

…and there is another important way to combine relations.

Definition: Let R be a relation from a set A to a set B and S a relation from B to a set C. The **composite** of R and S is the relation consisting of ordered pairs (a, c), where $a \in A$, $c \in C$, and for which there exists an element $b \in B$ such that $(a, b) \in R$ and $(b, c) \in S$. We denote the composite of R and S by **$S \circ R$**.

In other words, if relation R contains a pair (a, b) and relation S contains a pair (b, c), then $S \circ R$ contains a pair (a, c).

Combining Relations

Example: Let D and S be relations on A = {1, 2, 3, 4}.

D = {(a, b) | b = 5 - a} "b equals (5 – a)"

S = {(a, b) | a < b} "a is smaller than b"

D = {(1, 4), (2, 3), (3, 2), (4, 1)}

S = {(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)}

$S \circ D$ = { (2, 4), (3, 3), (3, 4), (4, 2), (4, 3), (4, 4)}

D maps an element a to the element (5 – a), and afterwards S maps (5 – a) to all elements larger than (5 – a), resulting in **$S \circ D = \{(a,b) \mid b > 5 - a\}$** or **$S \circ D = \{(a,b) \mid a + b > 5\}$.**

Combining Relations

We already know that **functions** are just **special cases** of **relations** (namely those that map each element in the domain onto exactly one element in the codomain).

If we formally convert two functions into relations, that is, write them down as sets of ordered pairs, the composite of these relations will be exactly the same as the composite of the functions (as defined earlier).

Combining Relations

Definition: Let R be a relation on the set A. The powers R^n, n = 1, 2, 3, .., are defined inductively by

$R^1 = R$

$R^{n+1} = R^n \circ R$

In other words:

$R^n = R \circ R \circ \ldots \circ R$ (n times the letter R)

Combining Relations

Theorem: The relation R on a set A is transitive if and only if $R^n \subseteq R$ for all positive integers n.

Remember the definition of transitivity:

Definition: A relation R on a set A is called transitive if whenever (a, b)∈R and (b, c)∈R, then (a, c)∈R for a, b, c∈A.

The composite of R with itself contains exactly these pairs (a, c).

Therefore, for a transitive relation R, $R \circ R$ does not contain any pairs that are not in R, so $R \circ R \subseteq R$.

Since $R \circ R$ does not introduce any pairs that are not already in R, it must also be true that $(R \circ R) \circ R \subseteq R$, and so on, so that $R^n \subseteq R$.

n-ary Relations

In order to study an interesting application of relations, namely **databases**, we first need to generalize the concept of binary relations to **n-ary relations**.

Definition: Let $A_1, A_2, \ldots, A_n$ be sets. An **n-ary relation** on these sets is a subset of $A_1 \times A_2 \times \ldots \times A_n$.

The sets $A_1, A_2, \ldots, A_n$ are called the **domains** of the relation, and n is called its **degree**.

n-ary Relations

Example:

Let $R = \{(a, b, c) \mid a = 2b \wedge b = 2c \text{ with } a, b, c \in \mathbf{N}\}$

What is the degree of R?

The degree of R is 3, so its elements are triples.

What are its domains?

Its domains are all equal to the set of integers.

Is (2, 4, 8) in R?

No.

Is (4, 2, 1) in R?

Yes.

Databases and Relations

Let us take a look at a type of database representation that is based on relations, namely the **relational data model.**

A database consists of n-tuples called **records**, which are made up of **fields**.

These fields are the **entries** of the n-tuples.

The relational data model represents a database as an n-ary relation, that is, a set of records.

Databases and Relations

Example: Consider a database of students, whose records are represented as 4-tuples with the fields **Student Name**, **ID Number**, **Major**, and **GPA**:

R = {(Ackermann, 231455, CS, 3.88),
(Adams, 888323, Physics, 3.45),
(Chou, 102147, CS, 3.79),
(Goodfriend, 453876, Math, 3.45),
(Rao, 678543, Math, 3.90),
(Stevens, 786576, Psych, 2.99)}

Relations that represent databases are also called **tables**, since they are often displayed as tables.

Databases and Relations

A domain of an n-ary relation is called a **primary key** if the n-tuples are uniquely determined by their values from this domain.

This means that no two records have the same value from the same primary key.

In our example, which of the fields **Student Name**, **ID Number**, **Major**, and **GPA** are primary keys?

Student Name and **ID Number** are primary keys, because no two students have identical values in these fields.

In a real student database, only **ID Number** would be a primary key.

Databases and Relations

In a database, a primary key should remain one even if new records are added.

Therefore, we should use a primary key of the **intension** of the database, containing all the n-tuples that can ever be included in our database.

Combinations of domains can also uniquely identify n-tuples in an n-ary relation.

When the values of a **set of domains** determine an n-tuple in a relation, the **Cartesian product** of these domains is called a **composite key**.

Databases and Relations

We can apply a variety of **operations** on n-ary relations to form new relations.

Definition: The **projection** $P_{i_1, i_2, .., i_m}$ maps the n-tuple $(a_1, a_2, .., a_n)$ to the m-tuple $(a_{i_1}, a_{i_2}, .., a_{i_m})$, where $m \leq n$.

In other words, a projection $P_{i_1, i_2, .., i_m}$ keeps the m components $a_{i_1}, a_{i_2}, .., a_{i_m}$ of an n-tuple and deletes its $(n - m)$ other components.

Example: What is the result when we apply the projection $P_{2,4}$ to the student record (Stevens, 786576, Psych, 2.99) ?

Solution: It is the pair (786576, 2.99).

Databases and Relations

In some cases, applying a projection to an entire table may not only result in fewer columns, but also in **fewer rows**.

Why is that?

Some records may only have differed in those fields that were deleted, so they become **identical**, and there is no need to list identical records more than once.

Databases and Relations

We can use the **join** operation to combine two tables into one if they share some identical fields.

Definition: Let R be a relation of degree m and S a relation of degree n. The **join** $J_p(R, S)$, where $p \leq m$ and $p \leq n$, is a relation of degree $m + n - p$ that consists of all $(m + n - p)$-tuples $(a_1, a_2, .., a_{m-p}, c_1, c_2, .., c_p, b_1, b_2, .., b_{n-p})$, where the m-tuple $(a_1, a_2, .., a_{m-p}, c_1, c_2, .., c_p)$ belongs to R and the n-tuple $(c_1, c_2, .., c_p, b_1, b_2, .., b_{n-p})$ belongs to S.

Databases and Relations

In other words, to generate Jp(R, S), we have to find all the elements in R whose p last components match the p first components of an element in S.

The new relation contains exactly these matches, which are combined to tuples that contain each matching field only once.

Databases and Relations

Example: What is $J_1(Y, R)$, where Y contains the fields **Student Name** and **Year of Birth**,

Y = {(1978, Ackermann),
(1972, Adams),
(1917, Chou),
(1984, Goodfriend),
(1982, Rao),
(1970, Stevens)},

and R contains the student records as defined before ?

Databases and Relations

Solution: The resulting relation is:

{(1978, Ackermann, 231455, CS, 3.88),
(1972, Adams, 888323, Physics, 3.45),
(1917, Chou, 102147, CS, 3.79),
(1984, Goodfriend, 453876, Math, 3.45),
(1982, Rao, 678543, Math, 3.90),
(1970, Stevens, 786576, Psych, 2.99)}

Since Y has two fields and R has four, the relation $J_1(Y, R)$ has $2 + 4 - 1 = 5$ fields.

Representing Relations

We already know different ways of representing relations. We will now take a closer look at two ways of representation: **Zero-one matrices** and **directed graphs**.

If R is a relation from $A = \{a_1, a_2, \ldots, a_m\}$ to $B = \{b_1, b_2, \ldots, b_n\}$, then R can be represented by the zero-one matrix $M_R = [m_{ij}]$ with

$m_{ij} = 1$, if $(a_i, b_j) \in R$, and

$m_{ij} = 0$, if $(a_i, b_j) \notin R$.

Note that for creating this matrix we first need to list the elements in A and B in a **particular, but arbitrary order**.

Representing Relations

Example: How can we represent the relation $R = \{(2, 1), (3, 1), (3, 2)\}$ as a zero-one matrix?

Solution: The matrix M_R is given by

$$M_R = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Representing Relations

What do we know about the matrices representing a **relation on a set** (a relation from A to A) ?

They are **square** matrices.

What do we know about matrices representing **reflexive** relations?

All the elements on the **diagonal** of such matrices M_{ref} must be **1s**.

$$M_{ref} = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & 1 \end{bmatrix}$$

Representing Relations

What do we know about the matrices representing **symmetric relations**?

These matrices are symmetric, that is, $M_R = (M_R)^t$.

$$M_R = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \qquad M_R = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

symmetric matrix, symmetric relation. non-symmetric matrix, non-symmetric relation.

Representing Relations

The Boolean operations **join** and **meet** (you remember?) can be used to determine the matrices representing the **union** and the **intersection** of two relations, respectively.

To obtain the **join** of two zero-one matrices, we apply the Boolean "or" function to all corresponding elements in the matrices.

To obtain the **meet** of two zero-one matrices, we apply the Boolean "and" function to all corresponding elements in the matrices.

Representing Relations

Example: Let the relations R and S be represented by the matrices

$$M_R = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad M_S = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

What are the matrices representing R∪S and R∩S?

Solution: These matrices are given by

$$M_{R\cup S} = M_R \vee M_S = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad M_{R\cap S} = M_R \wedge M_S = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Representing Relations Using Matrices

Example: How can we represent the relation R = {(2, 1), (3, 1), (3, 2)} as a zero-one matrix?

Solution: The matrix M_R is given by

$$M_R = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Representing Relations Using Matrices

Example: Let the relations R and S be represented by the matrices

$$M_R = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad M_S = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

What are the matrices representing $R \cup S$ and $R \cap S$?

Solution: These matrices are given by

$$M_{R \cup S} = M_R \vee M_S = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad M_{R \cap S} = M_R \wedge M_S = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Representing Relations Using Matrices

Do you remember the **Boolean product** of two zero-one matrices?

Let $A = [a_{ij}]$ be an $m \times k$ zero-one matrix and $B = [b_{ij}]$ be a $k \times n$ zero-one matrix.

Then the **Boolean product** of A and B, denoted by $A \odot B$, is the $m \times n$ matrix with (i, j)th entry $[c_{ij}]$, where

$c_{ij} = (a_{i1} \wedge b_{1j}) \vee (a_{i2} \wedge b_{2i}) \vee \ldots \vee (a_{ik} \wedge b_{kj})$.

$c_{ij} = 1$ if and only if at least one of the terms $(a_{in} \wedge b_{nj}) = 1$ for some n; otherwise $c_{ij} = 0$.

Representing Relations Using Matrices

Let us now assume that the zero-one matrices $M_A = [a_{ij}]$, $M_B = [b_{ij}]$ and $M_C = [c_{ij}]$ represent relations A, B, and C, respectively.

Remember: For $M_C = M_A \odot M_B$ we have:

$c_{ij} = 1$ if and only if at least one of the terms $(a_{in} \wedge b_{nj}) = 1$ for some n; otherwise $c_{ij} = 0$.

In terms of the **relations**, this means that C contains a pair (x_i, z_j) if and only if there is an element y_n such that (x_i, y_n) is in relation A and (y_n, z_j) is in relation B.

Therefore, $C = B \circ A$ (**composite** of A and B).

Representing Relations Using Matrices

This gives us the following rule:

$M_{B\circ A} = M_A \circ M_B$

In other words, the matrix representing the **composite** of relations A and B is the **Boolean product** of the matrices representing A and B.

Analogously, we can find matrices representing the **powers of relations**:

$M_{R^n} = M_R^{[n]}$ (n-th **Boolean power**).

Representing Relations Using Matrices

Example: Find the matrix representing R^2, where the matrix representing R is given by

$$M_R = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Solution: The matrix for R^2 is given by

$$M_{R^2} = M_R^{[2]} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Representing Relations Using Digraphs

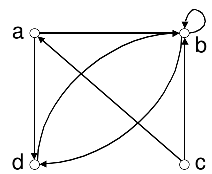
Definition: A **directed graph**, or **digraph**, consists of a set V of **vertices** (or **nodes**) together with a set E of ordered pairs of elements of V called **edges** (or **arcs**).

The vertex a is called the **initial vertex** of the edge (a, b), and the vertex b is called the **terminal vertex** of this edge.

We can use arrows to display graphs.

Representing Relations Using Digraphs

Example: Display the digraph with V = {a, b, c, d}, E = {(a, b), (a, d), (b, b), (b, d), (c, a), (c, b), (d, b)}.



An edge of the form (b, b) is called a **loop.**

Representing Relations Using Digraphs

Obviously, we can represent any relation R on a set A by the digraph with A as its vertices and all pairs (a, b)∈R as its edges.

Vice versa, any digraph with vertices V and edges E can be represented by a relation on V containing all the pairs in E.

This **one-to-one correspondence** between relations and digraphs means that any statement about relations also applies to digraphs, and vice versa.

Equivalence Relations

Equivalence relations are used to relate objects that are similar in some way.

Definition: A relation on a set A is called an equivalence relation if it is reflexive, symmetric, and transitive.

Two elements that are related by an equivalence relation R are called **equivalent**.

Equivalence Relations

Since R is **symmetric**, a is equivalent to b whenever b is equivalent to a.

Since R is **reflexive**, every element is equivalent to itself.

Since R is **transitive**, if a and b are equivalent and b and c are equivalent, then a and c are equivalent.

Obviously, these three properties are necessary for a reasonable definition of equivalence.

Equivalence Relations

Example: Suppose that R is the relation on the set of strings that consist of English letters such that aRb if and only if l(a) = l(b), where l(x) is the length of the string x. Is R an equivalence relation?

Solution:

- R is reflexive, because l(a) = l(a) and therefore aRa for any string a.
- R is symmetric, because if l(a) = l(b) then l(b) = l(a), so if aRb then bRa.
- R is transitive, because if l(a) = l(b) and l(b) = l(c), then l(a) = l(c), so aRb and bRc implies aRc.

R is an equivalence relation.

Equivalence Classes

Definition: Let R be an equivalence relation on a set A. The set of all elements that are related to an element a of A is called the **equivalence class** of a.

The equivalence class of a with respect to R is denoted by $\mathbf{[a]_R}$.

When only one relation is under consideration, we will delete the subscript R and write **[a]** for this equivalence class.

If $b \in [a]_R$, b is called a **representative** of this equivalence class.

Equivalence Classes

Example: In the previous example (strings of identical length), what is the equivalence class of the word mouse, denoted by [mouse] ?

Solution: [mouse] is the set of all English words containing five letters.

For example, 'horse' would be a representative of this equivalence class.

Equivalence Classes

Theorem: Let R be an equivalence relation on a set A. The following statements are equivalent:

- aRb
- [a] = [b]
- $[a] \cap [b] \neq \emptyset$

Definition: A **partition** of a set S is a collection of disjoint nonempty subsets of S that have S as their union. In other words, the collection of subsets A_i, $i \in I$, forms a partition of S if and only if

(i) $A_i \neq \emptyset$ for $i \in I$

- $A_i \cap A_j = \emptyset$, if $i \neq j$
- $\cup_{i \in I} A_i = S$

Equivalence Classes

Examples: Let S be the set {u, m, b, r, o, c, k, s}. Do the following collections of sets partition S ?

{{m, o, c, k}, {r, u, b, s}}	yes.
{{c, o, m, b}, {u, s}, {r}}	no (k is missing).
{{b, r, o, c, k}, {m, u, s, t}}	no (t is not in S).
{{u, m, b, r, o, c, k, s}}	yes.
{{b, o, o, k}, {r, u, m}, {c, s}}	yes ({b,o,o,k} = {b,o,k}).
{{u, m, b}, {r, o, c, k, s}, ∅}	no (∅ not allowed).

Equivalence Classes

Theorem: Let R be an equivalence relation on a set S. Then the **equivalence classes** of R form a **partition** of S. Conversely, given a partition $\{A_i \mid i \in I\}$ of the set S, there is an equivalence relation R that has the sets A_i, $i \in I$, as its equivalence classes.

Equivalence Classes

Example: Let us assume that Frank, Suzanne and George live in Boston, Stephanie and Max live in Lübeck, and Jennifer lives in Sydney.

Let R be the **equivalence relation** {(a, b) | a and b live in the same city} on the set P = {Frank, Suzanne, George, Stephanie, Max, Jennifer}.

Then R = {(Frank, Frank), (Frank, Suzanne), (Frank, George), (Suzanne, Frank), (Suzanne, Suzanne), (Suzanne, George), (George, Frank), (George, Suzanne), (George, George), (Stephanie, Stephanie), (Stephanie, Max), (Max, Stephanie), (Max, Max), (Jennifer, Jennifer)}.

Equivalence Classes

Then the **equivalence classes** of R are:

{{Frank, Suzanne, George}, {Stephanie, Max}, {Jennifer}}.

This is a **partition** of P.

The equivalence classes of any equivalence relation R defined on a set S constitute a partition of S, because every element in S is assigned to **exactly one** of the equivalence classes.

Equivalence Classes

Another example: Let R be the relation $\{(a, b) \mid a \equiv b \pmod 3\}$ on the set of integers.

Is R an equivalence relation?

Yes, R is reflexive, symmetric, and transitive.

What are the equivalence classes of R?

{{.., -6, -3, 0, 3, 6, ..},
{.., -5, -2, 1, 4, 7, ..},
{.., -4, -1, 2, 5, 8, ..}}

Yes, No, Maybe...

Boolean Algebra

Boolean Algebra

Boolean algebra provides the operations and the rules for working with the set **{0, 1}**.

These are the rules that underlie **electronic circuits**, and the methods we will discuss are fundamental to **VLSI design**.

We are going to focus on three operations:

- Boolean complementation,
- Boolean sum, and
- Boolean product

Boolean Operations

The **complement** is denoted by a bar (on the slides, we will use a minus sign). It is defined by

$-0 = 1$ and $-1 = 0$.

The **Boolean sum**, denoted by + or by OR, has the following values:

$1 + 1 = 1, \quad 1 + 0 = 1, \quad 0 + 1 = 1, \quad 0 + 0 = 0$

The **Boolean product**, denoted by · or by AND, has the following values:

$1 \cdot 1 = 1, \quad 1 \cdot 0 = 0, \quad 0 \cdot 1 = 0, \quad 0 \cdot 0 = 0$

Boolean Functions and Expressions

Definition: Let $B = \{0, 1\}$. The variable x is called a **Boolean variable** if it assumes values only from B.

A function from B^n, the set $\{(x_1, x_2, .., x_n) \mid x_i \in B, 1 \leq i \leq n\}$, to B is called a **Boolean function of degree n**.

Boolean functions can be represented using expressions made up from the variables and Boolean operations.

Boolean Functions and Expressions

The **Boolean expressions** in the variables $x_1, x_2, .., x_n$ are defined recursively as follows:

- $0, 1, x_1, x_2, .., x_n$ are Boolean expressions.
- If E_1 and E_2 are Boolean expressions, then $(-E_1)$, (E_1E_2), and $(E_1 + E_2)$ are Boolean expressions.

Each Boolean expression represents a Boolean function. The values of this function are obtained by substituting 0 and 1 for the variables in the expression.

Boolean Functions and Expressions

For example, we can create Boolean expression in the variables x, y, and z using the "building blocks" 0, 1, x, y, and z, and the construction rules:

Since x and y are Boolean expressions, so is xy.

Since z is a Boolean expression, so is (-z).

Since xy and (-z) are expressions, so is xy + (-z).

…and so on…

Boolean Functions and Expressions

Example: Give a Boolean expression for the Boolean function F(x, y) as defined by the following table:

x	y	F(x, y)
0	0	0
0	1	1
1	0	0
1	1	0

Possible solution: $F(x, y) = (-x)\cdot y$

Boolean Functions and Expressions

Another Example:

x	y	z	F(x, y, z)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Possible solution I:

$F(x, y, z) = -(xz + y)$

Possible solution II:

$F(x, y, z) = (-(xz))(-y)$

Boolean Functions and Expressions

There is a simple method for deriving a Boolean expression for a function that is defined by a table. This method is based on **minterms**.

Definition: A **literal** is a Boolean variable or its complement. A **minterm** of the Boolean variables x_1, x_2, .., x_n is a Boolean product $y_1 y_2 .. y_n$, where $y_i = x_i$ or $y_i = -x_i$.

Hence, a minterm is a product of n literals, with one literal for each variable.

Boolean Functions and Expressions

Consider F(x,y,z) again:

x	y	z	F(x, y, z)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

F(x, y, z) = 1 if and only if:

x = y = z = 0 or

x = y = 0, z = 1 or

x = 1, y = z = 0

Therefore,

F(x, y, z) =
(-x)(-y)(-z) +
(-x)(-y)z +
x(-y)(-z)

Boolean Functions and Expressions

Definition: The Boolean functions F and G of n variables are **equal** if and only if $F(b_1, b_2, .., b_n) = G(b_1, b_2, .., b_n)$ whenever $b_1, b_2, .., b_n$ belong to B.

Two different Boolean expressions that represent the same function are called **equivalent**.

For example, the Boolean expressions xy, xy + 0, and xy·1 are equivalent.

Boolean Functions and Expressions

The **complement** of the Boolean function F is the function –F, where $–F(b_1, b_2, .., b_n) = -(F(b_1, b_2, .., b_n))$.

Let F and G be Boolean functions of degree n. The **Boolean sum F+G** and **Boolean product FG** are then defined by

$(F + G)(b_1, b_2, .., b_n) = F(b_1, b_2, .., b_n) + G(b_1, b_2, .., b_n)$

$(FG)(b_1, b_2, .., b_n) = F(b_1, b_2, .., b_n)\ G(b_1, b_2, .., b_n)$

Boolean Functions and Expressions

Question: How many different Boolean functions of degree 1 are there?

Solution: There are four of them, F_1, F_2, F_3, and F_4:

x	F_1	F_2	F_3	F_4
0	0	0	1	1
1	0	1	0	1

Boolean Functions and Expressions

Question: How many different Boolean functions of degree 2 are there?

Solution: There are 16 of them, F_1, F_2, .., F_{16}:

x	y	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Boolean Functions and Expressions

Question: How many different Boolean functions of degree n are there?

Solution:

There are 2^n different n-tuples of 0s and 1s.

A Boolean function is an assignment of 0 or 1 to each of these 2^n different n-tuples.

Therefore, there are 2^{2^n} different Boolean functions.

Duality

There are useful identities of Boolean expressions that can help us to transform an expression A into an equivalent expression B (see Table 5 on page 597 in the textbook).

We can derive additional identities with the help of the **dual** of a Boolean expression.

The dual of a Boolean expression is obtained by interchanging Boolean sums and Boolean products and interchanging 0s and 1s.

Duality

Examples:

The dual of $x(y + z)$ is $x + yz$.

The dual of $-x \cdot 1 + (-y + z)$ is $(-x + 0)((-y)z)$.

The **dual of a Boolean function F** represented by a Boolean expression is the function represented by the dual of this expression.

This dual function, denoted by F^d, **does not depend** on the particular Boolean expression used to represent F.

Duality

Therefore, an identity between functions represented by Boolean expressions **remains valid** when the duals of both sides of the identity are

We can use this fact, called the to derive new identities.

For example, consider the absorption law

.

By taking the duals of both sides of this identity, x + xy , which is also an identity (and also called an absorption law).

All the properties of Boolean functions and expressions that we have discovered also apply to **other mathematical structures** propositions and sets and the operations defined on them.

Boolean algebra, then we know that all results established about Boolean algebras apply to this

For this purpose, we need an **abstract definition** of a Boolean algebra.

A Boolean algebra is a set B with two binary operations ∨ , elements 0 and 1, and a unary operation – such that the following properties hold for all x, y, and z in B:

x ∨ 1 = x (identity laws)

x ∨ (-x) = 0 (domination laws)

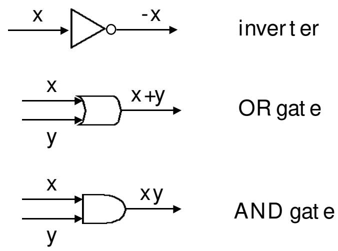
(x ∨ z = x ∨ z) and
(x ∧ z = x ∧ z) and (associative laws)

x ∨ x and x ∧ x (commutative laws)

x ∨ z) = (x ∨ (x ∨
x ∧ (y ∨ z) = (x ∧ y) ∨ (x ∧ z)

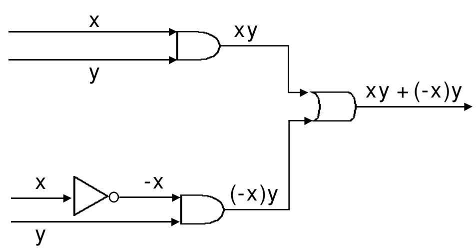
Logic Gates

Electronic circuits consist of so-called gates.
There are three basic types of gates:



Logic Gates

Example: How can we build a circuit that computes the function xy + (-x)y ?



The End