Welcome!

Discrete Mathematics for Computer Science
CompSci 102
D106 LSRC
M, W 10:05-11:20

Professor: Jeffrey Forbes

Frequently Asked Questions

- What are the prerequisites?
  - CPS 6 but CPS 100 preferred
  - Math 31 & 32
- How does this course fit into the curricula?
  - Useful foundation for courses like Compsci 130 and 140
  - Solid grounding in mathematical foundations
  - Replaces requirement of Math 135 (probability), Math 124 (Combinatorics) and Math 187 (Logic)
- What is recitation? Is it required?
  - Recitation is a more hands-on section where you will do problems and discuss solutions. Your work there will be graded.
- How do keep up to date?
  - Read web page *regularly*
    http://www.cs.duke.http://www.cs.duke.edu/courses/spring06/cps102
  - Read discussion forum *regularly*
    http://www.cs.duke.edu/phpBB2/index.php?c=75
  - Read your email

Course goals

- What we want to teach
  - Precise, reliable, powerful thinking
  - The ability to state and prove nontrivial facts, in particular about programs
  - Mathematical foundations and ideas useful throughout CS
  - Correctly read, represent and analyze various types of discrete structures using standard notations.
- What areas
  - Propositions and Proofs
  - Induction
  - Basics of Counting
  - Arithmetic Algorithms
  - Probability
  - Structures

So, what's *this* class about?

What are "discrete structures" anyway?

- "***Discrete***" (≠ "discreet"!) - Composed of distinct, separable parts. (Opposite of *continuous*.)
  discrete:*continuous* :: *digital*:*analog*
- "***Structures***" - Objects built up from simpler objects according to some definite pattern.
- "***Discrete Mathematics***" - The study of discrete, mathematical objects and structures.

What is Mathematics, really?

- It's *not* just about numbers!
- Mathematics is *much* more than that:

> Mathematics is, most generally, the study of any and all *absolutely certain* truths about any and all *perfectly well-defined* concepts.

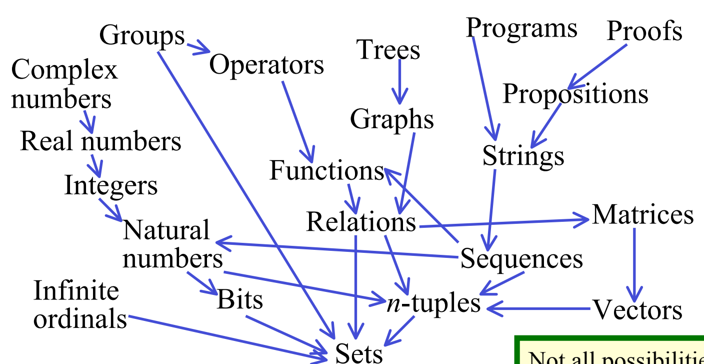
- But, these concepts can be *about* numbers, symbols, objects, images, sounds, *anything*!

Discrete Structures We'll Study

- Propositions
- Predicates
- Proofs
- Sets
- Functions
- Integers
- Summations
- Sequences
- Strings
- Permutations
- Combinations
- Relations
- Graphs

Relationships Between Structures

- "$\rightarrow$" $:\stackrel{\text{def}}{\equiv}$ "Can be defined in terms of"



Not all possibilities are shown here.

Some Notations We'll Learn

$\neg p$	$p \wedge q$	$p \oplus q$	$p \rightarrow q$	$p \Leftrightarrow q$	$\forall x\, P(x)$
$\exists x\, P(x)$	$\{a_1, \cdots, a_n\}$	$\mathbf{Z}, \mathbf{N}, \mathbf{R}$	$\therefore$	$\{x \mid P(x)\}$	$x \notin S$
$\varnothing$	$S \subseteq T$	$\lvert S \rvert$	$A \cup B$	$\overline{A}$	$\bigcap_{i=1}^{n} A_i$
$f : A \rightarrow B$	$f^{-1}(x)$	$f \circ g$	$\lfloor x \rfloor$	$\sum_{\alpha \in S} a_\alpha$	$\prod_{i=1}^{n} a_i$
O, Ω, Θ	min, max	$a \nmid b$	gcd, lcm	mod	$a \equiv b \pmod{m}$
$(a_k \cdots a_0)_b$	$[a_{ij}]$	$\mathbf{A}^{\mathrm{T}}$	$\mathbf{A}$Ï· $\mathbf{B}$	$\mathbf{A}^{[n]}$	$\binom{n}{r}$
$C(n; n_1, \cdots, n_m)$	$p(E \mid F)$	R^*	Δ	$[a]_R$	$\deg^+(v)$

Why Study Discrete Math?

- The basis of all of digital information processing is: *Discrete manipulations of discrete structures represented in memory.*
- Useful for solving the following calendar
  - Scheduling cab drivers for the Olympics
  - Akamai
  - Formal specification of XML
- Discrete math concepts are also widely used throughout math, science, engineering, economics, biology, *etc.*, …
- A generally useful tool for rational thought!

Uses for Discrete Math in Computer Science

- Advanced algorithms & data structures
- Programming language compilers & interpreters.
- Computer networks
- Operating systems
- Computer architecture
- Database management systems
- Cryptography
- Error correction codes
- Graphics & animation algorithms, game engines, *etc.*…
- *I.e.*, the whole field!

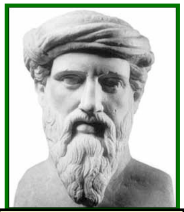
Course Outline (as per Rosen)

1. Logic (§1.1-4)
2. Proof methods (§1.5)
3. Set theory (§1.6-7)
4. Functions (§1.8)
5. Number theory (§2.4-5)
6. Number theory apps. (§2.6)
7. Proof strategy (§3.1)
8. Sequences (§3.2)
9. Summations (§3.2)
10. Countability (§3.2)
11. Inductive Proofs (§3.3)
12. Recursion (§3.4-5)
13. Program verification (§3.6)
14. Combinatorics (§4.1-4.4,4.6)
15. Probability (ch. 5)
16. Graph Theory (ch. 8)

Topics Not Covered

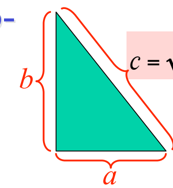
1. Algorithms!
   - See CompSci 130
2. Boolean circuits (ch. 10)
   - See CompSci 104 and EE 151
3. Models of computing (ch. 11)
   - See CompSci 140
4. Linear algebra & Matrices
   - See Math 104
5. Abstract algebra (not in Rosen)
   - Groups, rings, fields, vector spaces, algebras, *etc.*
   - See Math 121

A Proof Example



Pythagoras of Samos (ca. 569-475 B.C.)

- **Theorem:** *(Pythagorean Theorem of Euclidean geometry)* For <u>any</u> real numbers a, b, and c, if a and b are the base-length and height of a right triangle, and c is the length of its hypotenuse, then $a^2 + b^2 = c^2$.
- **Proof?**



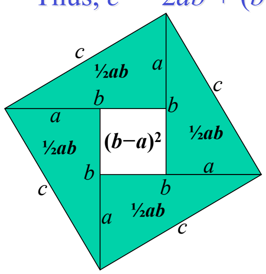
$c = \sqrt{a^2 + b^2}$

Propositions

- Statement that is either true or false
- Examples
  - ``This encryption system cannot be broken"
  - ``My program works efficiently in all cases"
  - ``There are no circumstances under which I would lie to Congress"
  - ``It is inconceivable that our legal system would execute an innocent person''
- A *theorem* is a proposition that is guaranteed by a proof

Proof of Pythagorean Theorem

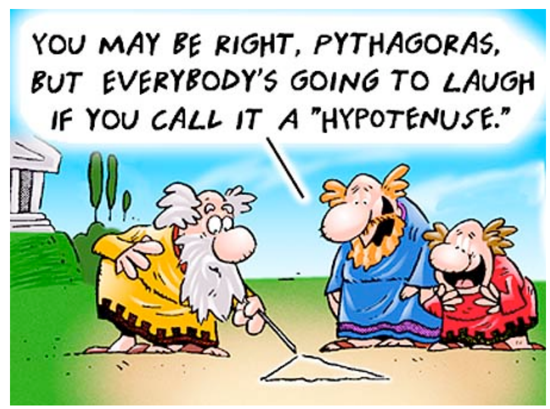
- **Proof.** Consider the below diagram:
  - Exterior square area = c^2, the sum of the following regions:
    - The area of the 4 triangles = $4(\frac{1}{2}ab) = 2ab$
    - The area of the small interior square = $(b-a)^2 = b^2-2ab+a^2$.
  - Thus, $c^2 = 2ab + (b^2-2ab+a^2) = a^2 + b^2$. ■



Note: It is easy to show that the exterior and interior quadrilaterals in this construction are indeed squares, and that the side length of the internal square is indeed $b-a$ (where b is defined as the length of the longer of the two perpendicular sides of the triangle). These steps would also need to be included in a more complete proof.

Areas in this diagram are in boldface; lengths are in a normal font weight.

Finally: Have Fun!

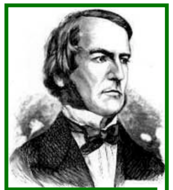


Propositional Logic (§1.1)

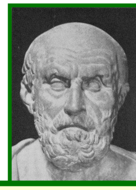
Propositional Logic is the logic of compound statements built from simpler statements using so-called *Boolean connectives*.

Some applications in computer science:

- Design of digital electronic circuits.
- Expressing conditions in programs.
- Queries to databases & search engines.



George Boole (1815-1864)



Chrysippus of Soli (ca. 281 B.C. – 205 B.C.)

Definition of a *Proposition*

Definition: A *proposition* (denoted *p*, *q*, *r*, …) is simply:

- a *statement* (*i.e.*, a declarative sentence)
  - *with some definite meaning*, (not vague or ambiguous)
- having a *truth value* that's either *true* (**T**) or *false* (**F**)
  - it is **never** both, neither, or somewhere "in between!"
    - However, you might not *know* the actual truth value,
    - and, the truth value might *depend* on the situation or context.
- Later, we will study *probability theory,* in which we assign *degrees of certainty* ("between" **T** and **F**) to propositions.
  - But for now: think True/False only!

Examples of Propositions

- "It is raining." (In a given situation.)
- "Beijing is the capital of China." • "1 + 2 = 3"

But, the following are **NOT** propositions:

- "Who's there?" (interrogative, question)
- "La la la la la." (meaningless interjection)
- "Just do it!" (imperative, command)
- "Yeah, I sorta dunno, whatever..." (vague)
- "1 + 2" (expression with a non-true/false value)

Operators / Connectives

An *operator* or *connective* combines one or more *operand* expressions into a larger expression. (*E.g.*, "+" in numeric exprs.)

- *Unary* operators take 1 operand (*e.g.,* −3); *binary* operators take 2 operands (*eg* 3 × 4).
- *Propositional* or *Boolean* operators operate on propositions (or their truth values) instead of on numbers.

Some Popular Boolean Operators

Formal Name	**Nickname**	**Arity**	**Symbol**
Negation operator	NOT	Unary	$\neg$
Conjunction operator	AND	Binary	$\wedge$
Disjunction operator	OR	Binary	$\vee$
Exclusive-OR operator	XOR	Binary	$\oplus$
Implication operator	IMPLIES	Binary	$\rightarrow$
Biconditional operator	IFF	Binary	$\leftrightarrow$

The Negation Operator

The unary *negation operator* "¬" (*NOT*) transforms a prop. into its logical *negation*.

E.g. If p = "I have brown hair."

then $\neg p$ = "I do **not** have brown hair."

The *truth table* for NOT:

p	$\neg p$
T	F
F	T
Operand column	Result column

T :≡ True; F :≡ False
":≡" means "is defined as"

The Conjunction Operator

The binary *conjunction operator* "∧" (*AND*) combines two propositions to form their logical *conjunction*.

∧ND

E.g. If p="I will have salad for lunch." and q="I will have steak for dinner.", then $p \wedge q$="I will have salad for lunch **and** I will have steak for dinner."

Remember: "∧" points up like an "A", and it means "AND"

Conjunction Truth Table

- Note that a conjunction $p_1 \wedge p_2 \wedge \ldots \wedge p_n$ of n propositions will have 2^n rows in its truth table.

Operand columns

p	q	$p \wedge q$
F	F	F
F	T	F
T	F	F
T	T	T

- Also: ¬ and ∧ operations together are sufficient to express *any* Boolean truth table!

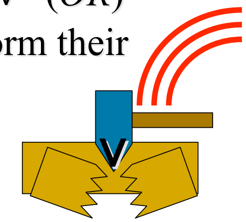
The Disjunction Operator

The binary *disjunction operator* "∨" (*OR*) combines two propositions to form their logical *disjunction*.

p="My car has a bad engine."

q="My car has a bad carburetor."

$p \lor q$="Either my car has a bad engine, **or** my car has a bad carburetor."



After the downward-pointing "axe" of "∨" splits the wood, you can take 1 piece OR the other, or both.

Meaning is like "and/or" in English.

Disjunction Truth Table

- Note that $p \lor q$ means that *p* is true, or *q* is true, **or both** are true!
- So, this operation is also called *inclusive or,* because it **includes** the possibility that both *p* and *q* are true.
- "¬" and "∨" together are also universal.

p	*q*	$p \lor q$
F	F	F
F	T	**T**
T	F	**T**
T	T	T

Note difference from AND

Nested Propositional Expressions

- Use parentheses to *group sub-expressions*: "I just saw my old *f*riend, and either he's *g*rown or I've *s*hrunk." = $f \land (g \lor s)$
  - $(f \land g) \lor s$ would mean something different
  - $f \land g \lor s$ would be ambiguous
- By convention, "¬" takes *precedence* over both "∧" and "∨".
  - $\neg s \land f$ means $(\neg s) \land f$, **not** $\neg (s \land f)$

A Simple Exercise

Let *p*="It rained last night",
q="The sprinklers came on last night,"
r="The lawn was wet this morning."

Translate each of the following into English:

$\neg p$ = "It didn't rain last night."

$r \land \neg p$ = "The lawn was wet this morning, and it didn't rain last night."

$\neg r \lor p \lor q$ = "Either the lawn wasn't wet this morning, or it rained last night, or the sprinklers came on last night."

The *Exclusive Or* Operator

The binary *exclusive-or operator* "⊕" (*XOR*) combines two propositions to form their logical "exclusive or" (exjunction?).

p = "I will earn an A in this course,"

q = "I will drop this course,"

$p \oplus q$ = "I will either earn an A in this course, or I will drop it (but not both!)"

Exclusive-Or Truth Table

- Note that $p \oplus q$ means that p is true, or q is true, but **not both**!
- This operation is called *exclusive or,* because it **excludes** the possibility that both p and q are true.
- "¬" and "⊕" together are **not** universal.

p	q	$p \oplus q$
F	F	F
F	T	T
T	F	T
T	T	**F**

Note difference from OR.

Natural Language is Ambiguous

Note that <u>English</u> "or" can be <u>ambiguous</u> regarding the "both" case!

"Pat is a singer or Pat is a writer." - $\vee$

"Pat is a man or Pat is a woman." - $\oplus$

p	q	p "or" q
F	F	F
F	T	T
T	F	T
T	T	?

Need context to disambiguate the meaning!

For this class, assume "or" means <u>inclusive</u>.

The *Implication* Operator

The *implication* $\underbrace{p}_{\text{antecedent}} \rightarrow \underbrace{q}_{\text{consequent}}$ states that p implies q.

I.e., If p is true, then q is true; but if p is not true, then q could be either true or false.

E.g., let p = "You study hard."
q = "You will get a good grade."

$p \rightarrow q$ = "If you study hard, then you will get a good grade." (else, it could go either way)

Implication Truth Table

- $p \rightarrow q$ is **false** <u>only</u> when p is true but q is **not** true.
- $p \rightarrow q$ does **not** say that p <u>causes</u> q!
- $p \rightarrow q$ does **not** require that p or q **<u>are ever true</u>**!
- *E.g.* "(1=0) → pigs can fly" is TRUE!

p	q	$p \rightarrow q$
F	F	T
F	T	T
T	F	**F**
T	T	T

The <u>only</u> False case!

Examples of Implications

- "If this lecture ever ends, then the sun will rise tomorrow." *True* or *False*?
- "If Tuesday is a day of the week, then I am a penguin." *True* or *False*?
- "If 1+1=6, then Bush is president." *True* or *False*?
- "If the moon is made of green cheese, then I am richer than Bill Gates." *True* or *False*?

Why does this seem wrong?

- Consider a sentence like,
  - "If I wear a red shirt tomorrow, then I will win the lottery!"
- In logic, we consider the sentence **True** so long as either I don't wear a red shirt, or I win the lottery.
- But, in normal English conversation, if I were to make this claim, you would think that I was lying.
  - Why this discrepancy between logic & language?

Resolving the Discrepancy

- In English, a sentence "if p then q" usually really *implicitly* means something like,
  - "<u>In all possible situations</u>, if p then q."
    - That is, "For p to be true and q false is *impossible*."
    - Or, "I *guarantee* that no matter what, if p, then q."
- This can be expressed in *predicate logic* as:
  - "For all situations s, if p is true in situation s, then q is also true in situation s"
  - Formally, we could write: $\forall s, P(s) \rightarrow Q(s)$
- *That* sentence is logically ***False*** in our example, because for me to wear a red shirt and for me to not win the lottery is a *possible* (even if not actual) situation.
  - Natural language and logic then agree with each other.

English Phrases Meaning $p \to q$

- "p implies q"
- "if p, then q"
- "if p, q"
- "when p, q"
- "whenever p, q"
- "q if p"
- "q when p"
- "q whenever p"
- "p only if q"
- "p is sufficient for q"
- "q is necessary for p"
- "q follows from p"
- "q is implied by p"

We will see some equivalent logic expressions later.

Converse, Inverse, Contrapositive

Some terminology, for an implication $p \to q$:

- Its *converse* is: $q \to p$.
- Its *inverse* is: $\neg p \to \neg q$.
- Its *contrapositive*: $\neg q \to \neg p$.
- One of these three has the *same meaning* (same truth table) as $p \to q$. Can you figure out which?

How do we know for sure?

Proving the equivalence of $p \to q$ and its contrapositive using truth tables:

p	q	$\neg q$	$\neg p$	$p \to q$	$\neg q \to \neg p$
F	F	T	T	T	T
F	T	F	T	T	T
T	F	T	F	F	F
T	T	F	F	T	T

The *biconditional* operator

The *biconditional* $p \leftrightarrow q$ states that p is true *if and only if (IFF)* q is true.

When we say **P if and only if q** , we are saying that P *says the same thing* as Q.

Examples?

Truth table?

Biconditional Truth Table

- $p \leftrightarrow q$ means that p and q have the **same** truth value.
- Note this truth table is the exact **opposite** of $\oplus$'s!

  Thus, $p \leftrightarrow q$ means $\neg(p \oplus q)$
- $p \leftrightarrow q$ does **not** imply that p and q are true, or that either of them causes the other, or that they have a common cause.

p	q	$p \leftrightarrow q$
F	F	T
F	T	F
T	F	F
T	T	T

Boolean Operations Summary

- We have seen 1 unary operator (out of the 4 possible) and 5 binary operators (out of the 16 possible). Their truth tables are below.

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \rightarrow q$	$p \leftrightarrow q$
F	F	T	F	F	F	T	T
F	T	T	F	T	T	T	F
T	F	F	F	T	T	F	F
T	T	F	T	T	F	T	T

Some Alternative Notations

Name:	not	and	or	xor	implies	iff
Propositional logic:	$\neg$	$\wedge$	$\vee$	$\oplus$	$\rightarrow$	$\leftrightarrow$
Boolean algebra:	$\overline{p}$	pq	$+$	$\oplus$		
C/C++/Java (wordwise):	!	&&	||	!=		==
C/C++/Java (bitwise):	~	&	|	^		
Logic gates: