Introduction and background review

1. What this course offers
2. How the course will deliver
3. Logic and proof techniques
4. Sets, relations, and functions

Inquiry: CS and mathematics

• What is *computer science* about?
• What is *math* about?
• In your experience, what do they have to do with each other?
• What mathematical concepts must you, as a computer-science major or a computing professional, be able to apply?
1. What this course offers

- We seek to create an *environment* to *collaboratively* investigate the topic
- *This environment will offer*
  - some questions for inquiry;
  - some information to support the inquiry;
  - ways to show learning results
  - an invitation to construct your own knowledge

Invitation: computing

*Are you curious about:*
- Problems that can’t be solved?
- What solvable problems are not worth solving?
- What is most likely to happen in the future?
- What is orderly about chaos?
- What ideas are the foundations for *all* of computer engineering, software engineering, web design, mobile apps, and robotics?
Invitation: knowing

• How do you know what you know?
• How do you know specific facts?
• How do you know general facts?
• Can you create new knowledge without observation and without hearing reports?

What we will investigate

• Discrete structures: Abstract objects that express aggregations or relationships among other mathematical objects
• Discrete or digital objects are as opposed to continuous or analog ones
• What the content of this course offers:
  • Mathematical tools applicable in CS
  • CS applications of this math
  • Ways to think abstractly about computation
Some objects of our study

- Integers
- Rational numbers
- Real numbers
- Sets
- Sequences
- Algebras
- Relations
- Functions
- Algorithms
- Graphs
- Trees
- Strings
- Languages
- Expressions
- Automata
- Assertions
- Proofs
- Logics

Discrete vs. continuous quantities

- Most empirical scientific data is *analog*: quantities along a continuum
- *Examples*: weight, pitch, distance, intensity, time
- The mathematics of the natural sciences has been mostly concerned with *real* numbers, trigonometry, and calculus
- Computers work with *digital* information
- *Example*: *symbols* are discrete
Digital representation of information

- *Information*: the meaning of data
- In computational devices and processes, data is represented by *discrete symbols*, e.g., 0 and 1
- Analog data is on a *continuum* (e.g., frequency and amplitude of sound waves)
- Analog input data may be *digitized*, converted to bits, e.g., by sound card
- Digital output may be converted to analog, e.g., by a sound card

DNA and computation

- *Life* processes discrete information, stored as proteins in DNA molecules
- Atoms and molecules have *discrete* forms
- *Example*: DNA strands are built from alphabet of only four different molecules
- In replicating, dividing, and recombining DNA, life can be said to *compute* on discrete symbolic values as a digital computer or the brain manipulates symbols logically
Why theory matters

• Every field of study, including computer science, has basic assumptions and concepts

• Example: the concepts of computing in CS, society in sociology, value in economics, etc.

• In CS, we assume that people want to reliably transform numbers and symbols

• These assumptions and concepts are the theoretical foundation of the field of study

• In CS, discrete mathematics provides part of the theoretical foundation

---

Algorithms as discrete structures

• Algorithms are abstract objects that specify step-by-step plans to transform an input to an output in a finite number of steps

• Flowcharts, pseudocode and programming languages implement algorithms

• Example: division of \( a \) by \( b \):

\[
\begin{align*}
y & \leftarrow 0 \\
\text{while } b & > 0 \\
\quad b & \leftarrow b - a \\
\quad y & \leftarrow y + 1
\end{align*}
\]
Algorithm: A precise plan to solve a functional problem in a finite number of steps

- Program designs use algorithms that convert pre-specified input to output
- Algorithms compute functions from input (parameters) to output (return values)
- In this classroom, “function” means “mapping,” not “subprogram”
- We may associate algorithms with the computations of machines and programs

Algorithms compute functions

- A function is a mapping, like a table
- Methods, procedures, and subroutines implement algorithms, which compute functions
  \[ x \rightarrow f \rightarrow y \]
- Algorithms take predefined inputs, which are the parameters to functions
- Some functions are uncomputable (Topic 5)
- All computable functions are recursively definable
Notation and definitions

• These are as in the slides for CSCI 347 and CSCI 460, based on notation used in standard computer-science texts such as
  – Hopcroft-Motwani-Ullman
  – Cormen-Leiserson-Rivest
  – Davis-Sigal-Weyuker
• Set names are all-caps
• Notation and definitions sometimes differ from Epp; in some cases, we correct Epp

Some fields of mathematics

• *Logic*: a set of languages for reasoning formally about true/false assertions
• *Set theory*: definitions and theorems about objects with given properties
• *Graph theory*: representations of relations
• *Functions*: mappings; also, the problems solved by algorithms
• *Combinatorics*: mathematics of counting and ordering objects
• *Probability*: the study of laws about quantifying the uncertainty of events
Definitions, examples, theorems, and proofs

• Mathematics and theoretical computer science use precise definitions of concepts.
• These can often be clarified by intuitive examples.
• The results are theorems: provable assertions about defined concepts.
• Proofs rely on chains of applied rules of logical inference.

A challenge

• Find the minimum number of connected line segments that together pass through all these dots.
• Start by trying out different solutions.
• What are your assumptions?
2. How the course will deliver

- What kind of classroom are you used to?
- How do you learn?
- What activities support your learning?
  - Discussion
  - Lectures
  - Your presenting
- What commitment are you ready to make?
- Can we make an agreement?

This course is an inquiry

- *FSU motto*: “Live to the truth”
- What does this mean?
- What is the truth?
- What does *truth* mean in software development and computer science?
- *Possible ways to the truth*:
  - Authority of experts
  - Our own experience, including experiments
  - Discussion
**Course plan**

1. Boolean algebras, logic, and induction
2. Sets, relations, and recurrences
3. Graphs and transition systems
4. Trees and their uses
5. Countability and decidability
6. Combinatorics and discrete probability
7. Information theory, randomness, and chaos

---

**Learning objectives, topics 1-4**

1. Explain and apply logical inference
2. Apply set-theoretic concepts, including the notions of relations, functions, and languages
3. Apply the basic notions of graph theory, including by means of proof
4. Prove properties of trees, describing their applications


**Learning objectives, T5-7**

5. Distinguish countable from uncountable sets, applying the diagonal proof method in number theory, logic, and computability

6. Apply the basic notions of combinatorics and discrete probability, including by proof

7. Explain connections between quantity of information and degree of randomness, comparing chaotic behavior to random behavior

---

**Background concepts (Data Structures and Finite/Discrete Math)**

- Propositional logic
- Direct proof
- Proof by contradiction
- Sets
- Relations
- Functions
- Graphs
- Trees
Course-wide objectives

0a. Participate in class activities throughout the semester
0b. Solve problems as part of a team
0c. Present results in the classroom
0d. Present written results
0e. Show knowledge of facts and concepts
0f. Summarize the semester’s learning
0g. Relate theoretical concepts to applications

What happens in all my courses

• For each topic, we have presentations, group work, discussion, assignments, and quizzes
• We investigate problems together, including
  – in small groups
  – in blackboard work
  – report backs from each student
• Grades are based on contribution, and on attainment of learning objectives as shown by problem-solving quiz results – best of three tries
• Semester project brings together topics
• See paper, “What we do in my classroom”
A 50-minute classroom session

1. Questions back and forth
2. I briefly present one subtopic, with slides
3. We discuss out-of-class exercises and solutions to some sample problems
4. Either
   – Problem solving in small groups, or
   – Student blackboard work, report-backs, and presentations

Course organization

• We have seven topics
• Each has an objective and 3-5 subtopics
• Each subtopic has 1-3 desired outcomes
• Some outcomes are considered core
• A set of problems for each subtopic outcome is available
• Exercises and quizzes draw from these
Role of students in course development

- I organize the materials to make sense to me
- I revise topics as I present them
- Students often help to
  - Clarify the materials
  - Correct errors

Classroom format

- Emphasis is on inquiry, activity, and collaboration
- Slides and short presentations summarize the content of the course
- We seek to create a natural critical learning environment
- Your contribution and participation matter
Group work in classroom

- Form groups of 3 students to take up a problem
- Each group should have
  - *Facilitator* – keeps discussion on track
  - *Recorder* – writes results of discussion
  - *Reporter* – presents results to class
- Participation by all in group work is one of our basic objectives in this course

Assessment and grading

*To measure:*
- Individual achievement of learning objectives
- Contribution to the learning of the class
*Breakdown: 60/40*
*Assumptions: Learning is shared and measurable*
Assessment of learning objectives

Assumptions:

- Application of concepts is measurable via core and other topic objectives
- Facts about concepts matter
- We learn by summarizing and reflecting

Assessment of contribution and participation

We assume that learning happens by:

- Inquiring and sharing inquiry
- Being present
- Solving problems together
- Activity throughout the semester
Final exam and summary quiz

- On final exam day, students will present elements of their semester projects
- During the last week of classes, we’ll have a summary quiz of multiple-choice questions and multi-topic problems

Academic integrity

- Direct use of text must be quoted and credited
- Use of ideas or other information must be credited by citations or references
- Citation standards for MLA and APA are given at www.citationmachine.net
- Plagiarism: representing someone else’s ideas or words as one’s own
- See catalog for FSU policy
What signing work means

- In this course, all code and words submitted are to be of the student who signs the work
  - Quizzes: no collaboration or device use
  - Exercises: device use and collaboration are OK, even required
- Principles:
  - Words belong to the original writer
  - Ideas belong to everyone; but we acknowledge their sources

A proposed agreement

I commit to:
- know the course material, present it clearly
- return submitted work within a week.
- answer questions helpfully

Students commit to:
- ask questions
- answer reasonable questions, risking error
- submit work on time, even if incomplete
- work sometimes in groups
- present results or lead discussions
Success

• My experience: students who want to learn are likely to do so
• It helps to be curious and to like programming
• You may benefit from taking notes in class to absorb and review knowledge
• Learning requires doing, and is interactive and social
• Overcoming obstacles can be satisfying

Pretest

• This pre-quiz will assess some capabilities presented in CS I
• Many of these are considered essential CS II outcomes
• There will be second and third chances on the objectives assessed
3. Logic and proof techniques

Inquiry:
• Once we have defined our terms, how can we precisely express what we believe?
• How can we be sure a belief is true?
• Is fantasizing helpful in math?
• What does proof have to do with computing?

Reading: Handout; Epp, Ch. 1-4

The language of logic

Values manipulated in logic are *truth values* in \{false, true\} or \{0, 1\}

Operators in propositional logic:
- Negation: *not* \( \neg \)
- Disjunction: *or* \( \lor \)
- Conjunction: *and* \( \land \)
- Implication: *if … then* \( \Rightarrow \)
- Equivalence: *iff* \( \iff \)
**Implication and equivalence**

$p \implies q$ is true whenever

- $q$ is true or
- both $p$ and $q$ are false

Assertions $p$ and $q$ are equivalent

$(p = q, p \iff q)$, whenever

- $(p \implies q) \land (q \implies p)$ or
- $p$ iff $q$ or
- $(p \land q) \lor (\neg p \land \neg q)$

**Some inference rules**

- **Modus Ponens**: $(p \land (p \implies q)) \implies q$
- **Modus Tollens**: $((p \implies q) \land \neg q) \implies \neg p$
- $p \implies q$ is defined as $\neg p \lor q$
- **Note**: if $p$ is false, then $(p \implies q)$ holds vacuously for any $q$, whether true or false
Quantifiers in predicate logic

- Variables may be *quantified*

- **Quantifiers:**
  - $\forall$ universal “for all”
  - $\exists$ existential “there exists”

- **Example:** $(\forall x \in \mathbb{N}) (\exists x') \text{ s.t. } x' = x + 1$
  (Every natural number has a successor)

- Predicate logic is characterized by the use of quantifiers, predicates, and other functions.

- **Predicate** a function that returns a truth value

---

Predicates and quantifiers

**Examples:**

- $Green(x)$ means that $x$ is green
- $Car(x)$: $x$ is a car
- $(\forall x) Red(x)$: everything is red
- $(\exists x) Car(x)$: a car exists
- $(\forall x) Car(x) \Rightarrow Blue(x)$: all cars are blue
Some proof methods

- **Contradiction** \((p \Rightarrow \neg p) \Rightarrow \neg p\)
  - Subcategory: diagonal proofs
- **Construction**
  Example: We prove that a problem is solvable by constructing an algorithm that provably solves it
  - Subcategory: proof by reduction
- **Induction:** May help prove an algorithm’s correctness

Proof by contradiction

- **Basic idea:** an assertion is true or false; not both
- **Steps:**
  1. Assume the negation of the assertion to be proven
  2. Show that this leads to a contradiction
  3. Hence the assumption must be false
- **Example theorem:** there is no largest integer.
  *Proof:* Suppose \(x\) were the largest integer. Now, \((x + 1)\) is larger. Hence \(x\) cannot be the greatest.
Proof by construction

- To show that some entity exists with a certain property \((\exists x) \, P(x)\), we construct an instance
- **Examples:**
  - to prove that \((4!)\) can be expressed using arithmetic, we may construct the expression \((4 \times 3 \times 2 \times 1)\)
  - to prove that Java can compute the factorial function, we may write a Java program that provably does so

Induction

- **Induction principle:**
  \[
  (\forall p \in (p : N \rightarrow \{T, F\})) \quad (p(0) \land (p(n) \Rightarrow p(n + 1))) \Rightarrow (\forall n) \, p(n)
  \]
  If a predicate holds for 0 and if it also holds for the successor of any value for which it holds, then the predicate holds for all natural numbers
- **Induction** is based on Peano’s inductive (recursive) definition of natural numbers:
  1. 0 is a natural number
  2. Every natural number \(x\) has a successor \(x'\) that is also a natural number
  3. There are no other natural numbers
Example: sum of linear series

Theorem: \( \sum_{k=1}^{n} k = (n^2 + n) \div 2 \)

Proof:

Base: \( \sum_{k=1}^{1} k = 1^2 + 1 = 2 \div 2 = 1 \)

Induction hypothesis: Suppose that for some \( n \),
\( \sum_{k=1}^{n} k = (n^2 + n) \div 2 \)
If so, then also \( \sum_{k=1}^{n+1} k = \frac{(n+1)^2 + (n + 1)}{2} \)
because \( \sum_{k=1}^{n} k + (n + 1) = \frac{(n^2 + n)}{2} + (n + 1) \)

\[ = \frac{(n^2 + 3n + 2)}{2} \]
\[ = \frac{((n + 1) (n + 2))}{2} \]
\[ = \frac{((n+1)^2 + (n + 1))}{2} \]

Confirming that \( P(n) \Rightarrow P(n+1) \); hence theorem holds

Example: sum of array elements

Theorem: The algorithm below sums array \( A \):

\[
\begin{align*}
\textit{sum} & \leftarrow 0 \\
\text{for } i & \leftarrow 1 \text{ to } |A| \\
\text{sum} & \leftarrow \textit{sum} + A[i]
\end{align*}
\]

Proof:

1. For any \( i \), let \( P(i) \) be the assertion that after \( i \) iterations, \( \textit{sum} \) stores sum of all elements to \( i \)
2. If \( P \) holds after \( i \) iterations, it will after \( (i + 1) \) iterations, because each iteration adds \( A[i] \)
3. By induction, after \( |A| \) iterations, \( \textit{sum} \) stores the sum of all array elements
Fantasy worlds in proofs

• When we say *suppose*, in a proof
  – We assume that all the facts of the real
    world hold
  – We enter a fantasy world in which some
    additional facts hold
  – We then may draw real-world conclusions
    from the fantasy suppositions

• *Example:* \((\forall S \subseteq \mathbb{N})\) means “Suppose \(S\) is any
  set of natural numbers…”

Skills for mathematical proofs

• These are *problem-solving* skills:
  – Flexibility
  – Willingness to use diverse approaches
  – Imagination
  – Intuition
  – Common sense

• Practice in these skills brings proficiency

• Proof rules and heuristics become familiar
  (“internalized”) with use
4. Sets, relations, and functions

Inquiry

• What are *strings* and *languages*?

• How does mathematics express *relationships? dependency*?

Sets

• A *set* is an aggregation of items, real, imaginary, or abstract, taken as a whole.

• Sets may be defined by
  – Enumeration, e.g.: $A = \{1, 2, 4\}$
  – Membership predicate, e.g.
    $$A = \{ x \mid \text{is_odd}(x) \}$$

• Sets are defined without respect to order:
  $$\{ 1, 2, 4 \} = \{ 2, 1, 4 \}$$

• A set contains no duplicates:
  $$\{ 1, 2, 4 \} = \{ 1, 2, 4, 2 \}$$
Set notation

- \( \in, \notin \): set membership, nonmembership
- \( \subseteq, \subset \): subset, proper subset
- \( \cup, \cap \): union, intersection
- \( A \times B \): Cartesian product of sets \( A \) and \( B \); a set of ordered pairs

\[
A = \{a, b, c\} \quad \text{specification by enumeration}
\]
\[
A = \{x \mid P(x)\} \quad \text{specification by predicate (}P\text{)}
\]

Union, intersection, complement

- **Membership:** “\( x \in A \)” means \( x \) is an element of set \( A \)
- **Union** of \( A \) and \( B \):
  \[
  (A \cup B) = \{ x \mid x \in A \lor x \in B \}
  \]
- **Intersection** of \( A \) and \( B \):
  \[
  (A \cap B) = \{ x \mid x \in A \land x \in B \}
  \]
- **Relative complement** of \( A \) w.r.t. \( B \):
  \[
  (A - B) = \{ x \mid x \in A \land x \notin B \}
  \]
- **Generalized union** (intersection, \( \cap \)):
  \[
  \cup \{ A, B, C \} = A \cup B \cup C
  \]
Inclusion (subsets)

- $A \subseteq B$ (set $A$ is a subset of set $B$) means that any member of $A$ is a member of $B$
- Strict inclusion: $A \subset B$ means $A \subseteq B \land A \neq B$
- If $A \subseteq B$ then $B \supseteq A$ ($B$ is a superset of $A$)
- For every set $A$, $A \subseteq A$
- The null set $\emptyset$ 
  - has no members
  - is a subset of all sets
- The power set of $A$, written $\mathcal{P}(A)$ or $2^A$, is the set of all subsets of $A$

Cartesian product of sets

- The Cartesian product of $n$ sets is the set of all $n$-tuples chosen from the sets
- *Ex:* $A \times B$ is the set of ordered pairs defined by selecting one member of each of $A$ and $B$
- $\{1,2\} \times \{a,b\} = \{(1,a), (1,b), (2,a), (2,b)\}$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1,a)</td>
<td>(1,b)</td>
</tr>
<tr>
<td>2</td>
<td>(2,a)</td>
<td>(2,b)</td>
</tr>
</tbody>
</table>
Binary relations

• *Definition:* a subset of a Cartesian product of two sets

• A relation $R$ is thus a set of ordered pairs, e.g., $\{(2,1), (3,1), (3,2), (4,1), (4,2), (4,3)\}$

• If $R$ is a relation and $(x, y) \in R$, we write $x R y$

• *Example:* $> \, \text{is a relation, because} \, 2 > 1; \, 3 > 1, \text{etc.}$


Relations and databases

• A database table is a relation

• *Examples:*
  
  – (names $\times$ IDs $\times$ addresses)
  
  – The Cartesian product of *students* and *courses* is the set of all possible pairings of *course*, *student*
  
  – A table of student registrations is a relation $R \subseteq (\text{students} \times \text{courses})$
Equivalence relations

- A relation $R$ on set $A$ is
  - Reflexive iff for all $x \in A$, $x R y$
  - Symmetric iff $(\forall \ x, y \in A) \ x R y \iff y R x$
  - Transitive iff $(\forall \ x, y, z \in A) \ x R y \land y R z \Rightarrow x R z$
- A relation that is reflexive, symmetric, and transitive is an *equivalence relation*

Graphs

- A graph $G = (V, E)$ is a set $V$ of vertices and a set $E$ of directed or undirected edges that connect pairs of vertices
- Graphs may have weighted edges
- Model communication relations, state transitions, precedence in time
- Algorithms exist to find paths, minimum-weight paths, minimal spanning trees
- *Tree*: a connected acyclic graph
Relations and directed graphs

A set of vertices $V$ and a set of edges $E$ connecting the vertices. ($E$ is a relation on $V$.)

$V = \{a, b, c, d, e\}$

$E = \{(a, b), (b, e), (b, d), (c, d), (d, c)\}$

Edges in a digraph

- An edge in a directed graph relates a vertex to another vertex

- Example: Who has sent email to whom? What intersections are connected by streets?

- A relation associates elements of sets, such as senders and recipients of email

- [pic]
 Paths

- **Path**: A sequence of pairwise adjacent vertices whose edges connect two vertices in a graph
- Path from $a$ to $d$: $(a,b,d)$

Cycles

- **Cycle**: a path from one vertex to itself that does not repeat an edge
- Cycle below: $(b, e, d)$
- A graph with no cycles is *acyclic*
A tree is hierarchic

- Examples:
  - Family tree
  - Parse tree
  - Taxonomy
  - Disk directory
  - Organizational hierarchy
  - Module hierarchy
- A tree has a **root** at top, **leaves** at bottom
- A nonroot node has one **parent** node
- A nonleaf has one or more **child** nodes
- Every tree node is the root of a subtree

Tree: a connected acyclic graph

- Exactly one path joins any pair of vertices
- Removing an edge disconnects the tree
- Adding any edge creates a cycle
- Cardinalities of edges and vertices:
  \[ |E| = |V| - 1 \]
- **Degree** of a vertex is the number of edges that include it
Parse trees

- Compilers build a program structure from tokens
- Root may be *program*
- Lexemes are leaves of tree
- Nonterminal syntax elements (e.g., *expression*, *factor*) are internal tree nodes or the root

Functions

- *Function*: a relation of which each left-hand member of a tuple is in not more than one tuple (maps to a unique value)
- *Examples*: *EVEN* is a function; *>* is a relation but not a function
- Every function has a
  - *Domain*: set of values mapped from
  - *Range*: set of values mapped to
- Computation of a function takes parameter as input, return value as output
Floor and ceiling functions

For $x, y \in \mathbb{R}$,

- Floor $\lfloor x \rfloor$ is the largest integer $y \leq x$,
- Ceiling $\lceil x \rceil$ is the smallest integer $y \geq x$,
- $x \div y = \lfloor x / y \rfloor$
- $x \mod y = x - y \lfloor x / y \rfloor$
- $Round(x) = \lfloor x - 0.5 \rfloor$

Injections and surjections

- **Injective** (one-to-one) function: one for which $(f(x) = f(y)) \Rightarrow (x = y)$
- **Example**: square root
- **Surjection**: one for which all values in range are mapped to by some value in the domain
- **Examples**: floor; absolute value
- **Bijection**: injective and surjective function
Time complexities of algorithms may be expressed as functions

- More data may require more time to process
- These functions are *not* the same as those calculated by the algorithms we analyze
- Our goal is to analyze running time independent of hardware and data size
Tools of algorithm analysis

1. Empirical performance testing
   a. timer
   b. counter

2. Theoretical tools
   The tools of *algorithm analysis*
   - Big-O notation
   - recurrences

Introducing big-O notation

- Expresses the time efficiency of an algorithm as a *function* of amount of data processed
- *Example:* To say that an algorithm is $O(n)$ means roughly that the algorithm’s running time is proportional to the size of the data set
Grouping time functions

- Functions that grow roughly at the same rate are in the same Big-O category
- We group together all constant functions, linear functions, quadratic functions, etc., as $O(1)$, $O(n)$, $O(n^2)$
- Logarithmic time: $O(\log_2 n) = O(\lg n)$
- Quadratic time: $O(n^2)$
- Exponential time: $O(2^n)$

Running time ($y$) in terms of size of input ($x$)
References