Research Methods in Computing Science

Unit II

Dr. Ayush Kumar Agrawal

School of Computer Science,
UPES

2025-26

Dialectic of Research in CS

Introduction

- Research in CS often evolves through debates, arguments, and counter-arguments.
- Dialectic = Structured reasoning to refine knowledge.
- Foundation for logical research thinking.

What is Dialectic?

- Derived from Greek philosophy: dialogue between opposing views.
- Goal: Arrive at a deeper understanding through reasoning.
- In CS: used in hypothesis testing, debugging, validation of models.

Dialectic Process in Research

1. Identify a claim or hypothesis.
2. Challenge with counter-arguments or experiments.
3. Refine through debate, evidence, and logic.
4. Arrive at stronger theories or solutions.

Example in Computing Science

- Claim: "Greedy algorithms always give optimal results."
- Counter: Provide counter-examples (e.g., Knapsack problem).
- Refinement: Greedy works only for specific problem classes (e.g., Huffman coding).
- Outcome: Stronger theoretical framework.

Role in Problem Formulation

- Encourages critical thinking.
- Helps in refining vague ideas into researchable problems.
- Forces researchers to consider multiple perspectives.
- Reduces bias and strengthens validity.

Applications in CS Research

- Designing algorithms (prove or disprove performance claims).
- Validating software engineering models.
- Establishing AI ethics through counter-arguments.
- Example: Debate between symbolic AI vs. neural networks.

Benefits of Dialectical Approach

- Leads to robust and reliable research.
- Prevents premature conclusions.
- Promotes interdisciplinary discussion.
- Strengthens reasoning and argumentation skills in scholars.

Summary & Discussion

- Dialectic $=$ structured debate $\rightarrow$ refined knowledge.
- Essential for hypothesis validation and model building in CS.
- **Discussion Prompt:** Can dialectical reasoning help in debugging large software systems? How?

Models of Argument

Introduction to Argument Models

- Research in CS relies heavily on logical arguments.
- Argument models help validate claims and hypotheses.
- Three key reasoning models:
  - Deductive
  - Inductive
  - Abductive

Deductive Reasoning

- General $\rightarrow$ Specific reasoning.
- If premises are true, conclusion must be true.
- Example in CS:
  - Premise: All Dijkstra's shortest path results are optimal.
  - Premise: Graph G satisfies Dijkstra's assumptions.
  - Conclusion: Result for G is optimal.

Inductive Reasoning

- Specific observations $\rightarrow$ General rules.
- Conclusion is probable, not certain.
- Example in CS:
  - Observing that a sorting algorithm outperforms Quicksort on 100 test cases.
  - Induction: Algorithm likely faster in general.

Abductive Reasoning

- Inference to the "best explanation".
- Starts with observations, seeks the simplest cause.
- Example in CS:
  - Observation: Program crashes randomly.
  - Hypothesis: Memory leak is the most likely cause.

Comparison of Argument Models

Criteria	Deductive	Inductive	Abductive
Basis	Logic	Observations	Best Explanation
Certainty	Guaranteed	Probable	Plausible
Example	Algorithm proof	ML training	Debugging

Applications in CS

- Deductive: Formal verification, algorithm proofs
- Inductive: Data mining, ML, trend analysis
- Abductive: Debugging, hypothesis generation in AI

Strengths and Limitations

- Deductive: Reliable, but requires true premises.
- Inductive: Useful, but risk of overgeneralization.
- Abductive: Creative, but may lead to false conclusions.
- Combination often yields strongest research.

Summary & Discussion

- Deduction = certainty, Induction = probability, Abduction = plausibility.
- All three are essential in CS research.
- **Discussion Prompt:** How can inductive and abductive reasoning complement deductive proofs in developing a new algorithm?

Proof Methods – Introduction

What are Proof Methods?

- Systematic approaches to establish validity of claims.
- Used in CS for algorithm correctness, software verification, system validation.
- Categories:
  - Demonstration
  - Empirical
  - Mathematical

Need for Proof in CS Research

- Research requires validation of claims.
- Proof provides:
  - Rigor
  - Reliability
  - Reproducibility
- Avoids assumptions without evidence.

Demonstration Method

- Showing correctness using test cases or prototypes.
- Example: Running a sorting algorithm on a sample dataset.
- Advantage: Easy to understand and communicate.
- Limitation: Cannot guarantee correctness in all cases.

Empirical Method

- Based on observation, experimentation, and measurement.
- Example: Benchmarking performance of an AI model on datasets.
- Advantage: Reflects real-world performance.
- Limitation: Context-dependent, may not generalize.

Mathematical Method

- Uses logic and formal proof techniques.
- Examples:
  - Proving algorithm complexity.
  - Correctness proof of finite automata.
- Advantage: Rigor and universality.
- Limitation: Requires strong mathematical foundation.

Comparison of Methods

Aspect	Demonstration	Empirical	Mathematical
Basis	Example cases	Observation	Logic
Scope	Limited	Context-based	Universal
Rigor	Low	Medium	High
Use	Early testing	Benchmarking	Theoretical proof

Applications in CS Research

- Demonstration: Teaching tools, quick prototypes.
- Empirical: Performance of ML models, network protocols.
- Mathematical: Algorithm design, cryptography proofs.

Summary & Discussion

- Proof methods = essential backbone of CS research.
- Demonstration, empirical, and mathematical proofs complement each other.
- **Discussion Prompt:** Which proof method is most reliable in software security research, and why?

Proof by Demonstration

What is Proof by Demonstration?

- Showing correctness through examples or prototypes.
- Focuses on illustrating functionality in practice.
- Often used in early stages of research and teaching.

Key Features

- Based on practical examples, not theory.
- Convincing, but not rigorous.
- Demonstrates feasibility and usability.
- Often the “first proof” before formal methods.

Example in Computing

- Claim: “This new sorting algorithm is efficient.”
- Demonstration: Run it on a few sample arrays.
- Outcome: Shows algorithm works, but does not prove efficiency for all cases.

Advantages

- Easy to communicate and visualize.
- Helps in quick validation of ideas.
- Useful for prototypes, early-stage research, and teaching.
- Encourages practical experimentation.

Limitations

- Works only for tested cases.
- Cannot guarantee universal correctness.
- May overlook corner cases and exceptions.
- Not acceptable as final rigorous proof.

Applications in CS

- Classroom teaching (e.g., demonstrating BFS/DFS on a graph).
- Early-stage algorithm design.
- Prototype validation in software engineering.
- Initial demonstration of robotics/IoT projects.

Comparison with Other Proofs

- **Demonstration:** Quick, illustrative, non-rigorous.
- **Empirical:** Based on experiments & measurements.
- **Mathematical:** Rigorously proves correctness.

Summary & Discussion

- Proof by demonstration = showing correctness with examples.
- Very useful in teaching, prototyping, early validation.
- **Discussion Prompt:** In which research stage should proof by demonstration be replaced by empirical or mathematical proof?

Empirical Proofs

What are Empirical Proofs?

- Validation based on observation, experimentation, and measurement.
- Uses data from experiments to support conclusions.
- Common in applied research and performance studies.

Key Features

- Relies on real-world evidence.
- Often uses benchmarks, simulations, or case studies.
- Results are context-dependent (system, dataset, environment).
- Provides strong practical validation but not universal proof.

Example in Computing

- Claim: "Our new machine learning model outperforms CNNs."
- Empirical Proof: Train and test on benchmark datasets (MNIST, CIFAR-10).
- Evidence: Accuracy, precision, recall, runtime.
- Limitation: Results depend on dataset and environment.

Advantages

- Reflects practical performance.
- Allows testing under real-world conditions.
- Supports iterative improvements.
- Highly persuasive in applied domains.

Limitations

- Context-specific: may not generalize.
- Resource-intensive (experiments, hardware).
- Results can be influenced by external factors.
- Cannot replace formal mathematical proof.

Applications in CS

- Benchmarking algorithms and software.
- Network performance measurement.
- AI/ML model validation.
- Usability testing in HCI and system design.

Empirical vs Demonstration vs Mathematical

- **Demonstration:** Works for a few cases, non-rigorous.
- **Empirical:** Data-driven, practical validation.
- **Mathematical:** Formal, universal proof.

Summary & Discussion

- Empirical proofs = experimental, data-based validation.
- Strong for applied domains, but not universally valid.
- **Discussion Prompt:** If empirical results contradict mathematical analysis, which should we trust more in CS research?

Mathematical Proofs in Computer Science

What are Mathematical Proofs?

- Rigorous validation using logic and mathematics.
- Provides certainty and universal applicability.
- Backbone of theoretical computer science.

Types of Mathematical Proofs

- Direct Proof
- Proof by Contradiction
- Proof by Induction
- Proof by Contrapositive

Proof by Induction (Example)

- Common in algorithm analysis.
- Example: Prove that sum of first n natural numbers is $\frac{n(n+1)}{2}$.
- Base Case: $n = 1$, true.
- Inductive Step: Assume true for $n = k$, prove for $n = k + 1$.

Proof by Contradiction (Example)

- Assume the opposite of what we want to prove.
- Example: Proving $\sqrt{2}$ is irrational.
- In CS: Prove impossibility of certain algorithms (e.g., Halting Problem).

Applications in CS

- Algorithm correctness proofs.
- Computational complexity (Big-O, NP-completeness).
- Cryptography (security proofs).
- Automata theory and formal languages.

Advantages

- Absolute rigor and certainty.
- Universally applicable (independent of datasets).
- Reveals theoretical limits of computation.

Limitations

- May not reflect real-world performance.
- Requires strong mathematical skills.
- Often abstract, not practical for applied research.

Summary & Discussion

- Mathematical proofs = most rigorous method in CS.
- Essential in theory, algorithms, and cryptography.
- **Discussion Prompt:** Should mathematical proof always be required in applied CS research, or is empirical validation sufficient?

Deduction in Computer Science

What is Deduction?

- Reasoning from general principles to specific conclusions.
- If premises are true, conclusion must be true.
- Used extensively in logic, algorithms, and formal verification.

Deductive Reasoning Process

1. Start with a general law or theorem.
2. Apply it to a specific case.
3. Derive logically valid conclusions.

Example in CS

- General Rule: "All comparison-based sorting algorithms require $\Omega(n \log n)$ comparisons in the worst case."
- Specific Case: Quicksort is comparison-based.
- Deductive Conclusion: Quicksort requires at least $\Omega(n \log n)$ comparisons.

Applications in CS

- Algorithm correctness proofs.
- Complexity analysis (runtime, space).
- Formal verification of programs.
- Security protocol validation.

Advantages of Deduction

- Provides certainty if premises are correct.
- Universally applicable.
- Builds strong theoretical foundations.

Limitations of Deduction

- Dependent on correctness of premises.
- May ignore practical performance.
- Sometimes too abstract for applied research.

Deduction vs Other Methods

- **Deduction:** Theory $\rightarrow$ Application.
- **Induction:** Observations $\rightarrow$ Generalization.
- **Abduction:** Best explanation for an observation.

Summary & Discussion

- Deduction ensures logical certainty in CS research.
- Useful in algorithms, formal proofs, and verification.
- **Discussion Prompt:** Can deductive reasoning alone validate the efficiency of a new algorithm, or must it be combined with empirical evidence?

Induction in Computer Science

What is Induction?

- Reasoning from specific observations to general principles.
- Conclusions are probable, not certain.
- Widely used in CS for learning patterns and building theories.

Inductive Reasoning Process

1. Collect observations or experimental results.
2. Identify patterns or regularities.
3. Formulate general rules or hypotheses.

Example in CS

- Observation: A sorting algorithm is faster than Quicksort on 500 test cases.
- Inductive Conclusion: Algorithm is likely faster in general.
- Use case: Machine Learning — generalizing from training data.

Applications in CS

- Machine learning and AI (generalizing from data).
- Data mining and pattern recognition.
- Software engineering (bug prediction models).
- Performance estimation of systems.

Advantages of Induction

- Powerful in discovering new knowledge.
- Useful when mathematical proofs are impractical.
- Provides predictive insights.

Limitations of Induction

- Conclusions are not guaranteed (probabilistic).
- May overgeneralize from limited data.
- Dependent on quality of dataset or observations.

Deduction vs Induction

- **Deduction:** General $\rightarrow$ Specific (certainty).
- **Induction:** Specific $\rightarrow$ General (probability).
- Both are complementary in CS research.

Summary & Discussion

- Induction = reasoning from observations to general rules.
- Strongly used in ML, data mining, software analytics.
- **Discussion Prompt:** Can inductive reasoning alone ensure reliability of an AI model, or must it always be supported by deductive reasoning?

Theoretical Models and Approaches in CS

What are Theoretical Models?

- Abstract representations of computational systems.
- Help formalize, analyze, and predict behavior of algorithms.
- Foundation of theoretical computer science.

Types of Theoretical Models

- Automata Models (Finite Automata, Pushdown Automata, Turing Machines).
- Logic-based Models (Propositional, Predicate Logic).
- Graph-based Models (Networks, trees).
- Algebraic Models (Boolean algebra, formal grammars).

Example: Turing Machine

- Abstract machine capable of simulating any algorithm.
- Defines computability and decidability.
- Basis for complexity theory (P vs NP problems).

Role of Theoretical Models in CS

- Define computational limits and possibilities.
- Provide mathematical rigor for algorithms.
- Aid in classification of problems (decidable, NP-hard, etc.).

Advantages of Theoretical Models

- Clarity and precision in research.
- Universality: Independent of hardware/software.
- Provide a foundation for formal proofs.

Limitations of Theoretical Models

- May not capture real-world constraints (time, hardware).
- Often abstract and difficult to implement.
- Sometimes oversimplify complex systems.

Applications in Research

- Automata theory $\rightarrow$ Compiler design, language recognition.
- Graph theory $\rightarrow$ Networks, AI planning.
- Logic models $\rightarrow$ Verification of programs.
- Complexity theory $\rightarrow$ Cryptography, optimization.

Summary & Discussion

- Theoretical models define the foundation of CS research.
- Provide rigorous methods but may lack practical constraints.
- **Discussion Prompt:** Are theoretical models sufficient to validate modern AI systems, or must they always be combined with empirical evaluation?

Algorithmic Approaches

What are Algorithmic Approaches?

- Systematic design and analysis of algorithms to solve problems.
- Core research method in Computer Science.
- Combines theoretical analysis with practical implementation.

Steps in Algorithmic Research

1. Define the problem clearly.
2. Design algorithmic solution(s).
3. Analyze complexity (time, space).
4. Prove correctness.
5. Test on real-world datasets.

Examples in CS

- Graph Algorithms: Dijkstra, A*.
- Sorting Algorithms: Quicksort, Merge Sort, Heap Sort.
- Optimization: Dynamic Programming, Greedy methods.
- Approximation Algorithms for NP-hard problems.

Advantages of Algorithmic Approaches

- Provide efficiency and scalability.
- Enable automation of problem-solving.
- Theoretical guarantees on performance.
- Foundational for all areas of computing.

Limitations

- May ignore real-world hardware constraints.
- Assumptions may oversimplify practical conditions.
- Difficult to apply for unstructured or fuzzy problems (e.g., human language, images).

Applications in Research

- Cryptography: Secure key generation, encryption.
- Data Science: Clustering, classification, recommendation.
- AI/ML: Optimization of neural networks.
- Robotics: Path planning and motion control.

Algorithmic vs Other Approaches

- **Algorithmic:** Clear steps, efficient, provable.
- **Empirical:** Focuses on experimental results.
- **Mathematical:** Focuses on proof and theory.
- Often combined in real research projects.

Summary & Discussion

- Algorithmic approaches = structured problem solving.
- Provide both theoretical and practical value.
- **Discussion Prompt:** Should algorithm research always prioritize efficiency, or are simplicity and readability equally important?

Software Engineering Approaches

What are SE Approaches?

- Methods used to design, develop, test, and maintain software systems.
- Combine systematic processes with empirical validation.
- Bridge between theoretical models and practical implementation.

SE in Research Methodology

- Involves structured process models (Waterfall, Agile, Spiral).
- Helps in organizing research projects as “software-like” development cycles.
- Iterative refinement of hypotheses similar to iterative coding.

Steps in SE Research Approach

1. Requirement analysis (problem definition).
2. Design (models, architecture, UML diagrams).
3. Implementation (coding, prototyping).
4. Testing verification.
5. Maintenance future enhancements.

Examples in CS Research

- Developing a machine learning pipeline as a software project.
- Applying Agile methodology for robotics system development.
- Using DevOps for reproducible computational experiments.

Advantages of SE Approaches

- Provides structured methodology for research projects.
- Improves reproducibility and maintainability.
- Encourages collaboration among researchers.
- Ensures deliverables are testable and scalable.

Limitations of SE Approaches

- Can be time-consuming and process-heavy.
- May introduce overhead in fast-paced research.
- Not always suitable for exploratory research with high uncertainty.

Applications in Research Practice

- Software prototyping for new algorithms.
- Testing frameworks for validation of research outputs.
- Agile research labs for iterative improvement.
- Model-driven engineering in academic prototypes.

Summary & Discussion

- SE approaches bring structure, testing, and reproducibility to CS research.
- Especially useful when research outputs involve software systems.
- **Discussion Prompt:** Should research prototypes be developed with the same rigor as industry-grade software, or is a lighter approach acceptable in academia?

Mathematical Modelling in Computer Science

What is Mathematical Modelling?

- Process of representing real-world systems using mathematical structures.
- In CS: models help analyze performance, predict outcomes, and validate systems.
- Bridges theory and practice.

Steps in Mathematical Modelling

1. Identify the system or problem.
2. Make assumptions and define parameters.
3. Formulate equations or models.
4. Solve/analyze model.
5. Validate with experiments or simulations.

Examples in CS

- Queueing theory $\rightarrow$ network performance analysis.
- Graph theory $\rightarrow$ social networks, routing.
- Probability models $\rightarrow$ reliability, fault tolerance.
- Differential equations $\rightarrow$ epidemic modelling with computational simulations.

Advantages of Mathematical Modelling

- Provides clarity and precision.
- Helps in prediction and optimization.
- Reduces cost by avoiding real-world trials.
- Can generalize across multiple scenarios.

Limitations

- Models may oversimplify reality.
- Accuracy depends on assumptions.
- May require high computational resources.
- Not always intuitive for non-technical stakeholders.

Applications in CS Research

- Performance analysis of distributed systems.
- Security threat modelling.
- AI/ML – mathematical formulation of optimization.
- Robotics – kinematics and dynamics modelling.

Modelling Tools and Techniques

- Mathematical software: MATLAB, Mathematica.
- Simulation tools: NS2/NS3, OMNeT++, MATLAB Simulink.
- Programming: Python (SciPy, NumPy, SymPy).

Summary & Discussion

- Mathematical models simplify complex systems into analyzable form.
- Crucial in performance prediction and optimization in CS.
- **Discussion Prompt:** Should CS researchers always prioritize mathematical models over empirical testing, or do both approaches complement each other?

Performance Estimation and Evaluation

What is Performance Evaluation?

- Process of assessing efficiency, scalability, and reliability of systems.
- In CS: applies to algorithms, software, networks, and hardware.
- Helps validate claims made in research.

Key Parameters

- **Time Efficiency:** Execution time, response time.
- **Space Efficiency:** Memory/Storage usage.
- **Scalability:** Performance with larger inputs.
- **Reliability:** Consistency under stress/failure.

Process of Performance Evaluation

1. Define performance metrics.
2. Select test environment (datasets, hardware, simulators).
3. Run experiments/measurements.
4. Analyze results and compare with benchmarks.
5. Interpret and validate findings.

Examples in CS

- Algorithm analysis → runtime complexity vs. empirical execution time.
- Cloud computing → throughput, latency, fault tolerance.
- Networks → bandwidth, jitter, packet loss.
- ML models → accuracy, precision, recall, F1 score.

Advantages

- Validates practical efficiency of research outputs.
- Provides measurable evidence for claims.
- Helps identify bottlenecks and areas of improvement.

Limitations

- Resource-intensive (time, hardware, datasets).
- Results may vary with environment or conditions.
- Difficult to generalize across different systems.

Tools for Evaluation

- Profilers: gprof, Valgrind, JProfiler.
- Simulation tools: NS3, CloudSim, OMNeT++.
- Benchmarking suites: SPEC CPU, MLPerf.
- Monitoring tools: Prometheus, Grafana.

Summary & Discussion

- Performance evaluation = critical step in CS research.
- Combines metrics, tools, and benchmarks for validation.
- **Discussion Prompt:** Should performance evaluation focus more on average-case results or worst-case guarantees in CS research?