

M269

Exam Revision

Contents

1	M269 Exam Revision Agenda & Aims	1
1.1	Revision strategies	2
2	Units 6 & 7	2
2.1	Computability	2
2.2	Reducing one problem to another	2
2.3	Computability — Models of Computation	3
2.4	Computability — Church-Turing Thesis	3
2.5	Computability — Turing Machine	3
2.6	Turing Machine notation	4
2.7	Computability — Decidability	4
2.8	Complexity	5
2.9	Logic	6
2.10	Justified Arguments	7
2.11	Unit 6 Sets, Databases, Logic	8
3	Units 3, 4 & 5	9
3.1	Unit 5 Optimisation	9
3.2	Unit 4 Searching	9
3.3	Unit 3 Sorting	9
3.4	Sorting Algorithms	10
4	Units 1 & 2	10
4.1	Unit 2 From Problems to Programs	10
4.2	Unit 1 Introduction	11
5	M269 Exam Section B	11
6	Exam Techniques	11
7	Web References	11
	References	14

1 M269 Exam Revision Agenda & Aims

1. Welcome and introductions
2. Revision strategies
3. Specimen exam — Part A in reverse order
4. Topics and discussion for each question
5. Exam technique

1.1 Revision strategies

- Organising your knowledge
- Each give one exam tip to the group
- TODO: add some more points

2 Units 6 & 7

2.1 Computability

Q15 topics

- Unit 7
- Computability and ideas of computation
- Complexity
- P and NP
- NP-complete
- The idea of an algorithm and what is effectively computable
- **Church-Turing thesis** Every function that would naturally be regarded as computable can be computed by a deterministic Turing Machine. (Unit 7 Section 4)

2.2 Reducing one problem to another

- To reduce problem P_1 to P_2 , invent a construction that converts instances of P_1 to P_2 that have the same answer. That is:
 - any string in the language P_1 is converted to some string in the language P_2
 - any string over the alphabet of P_1 that is not in the language of P_1 is converted to a string that is not in the language P_2
- With this construction we can solve P_1
 - Given an instance of P_1 , that is, given a string w that may be in the language P_1 , apply the construction algorithm to produce a string x
 - Test whether x is in P_2 and give the same answer for w in P_1

([Hopcroft et al., 2007](#), page 322)

- The direction of reduction is important
- If we can reduce P_1 to P_2 then (in some sense) P_2 is at least as hard as P_1 (since a solution to P_2 will give us a solution to P_1)
- So, if P_2 is decidable then P_1 is decidable
- To show a problem is undecidable we have to reduce from a known undecidable problem to it

- Since, if P_1 is undecidable then P_2 is undecidable

2.3 Computability — Models of Computation

- In automata theory, a *problem* is the question of deciding whether a given string is a member of some particular language
- If Σ is an alphabet, and L is a language over Σ , that is $L \subseteq \Sigma^*$, where Σ^* is the set of all strings over the alphabet Σ then we have a more formal definition of *decision problem*
- Given a string $w \in \Sigma^*$, decide whether $w \in L$
- Example: Testing for a prime number — can be expressed as the language L_p consisting of all binary strings whose value as a binary number is a prime number (only divisible by 1 or itself)

(Hopcroft et al., 2007, section 1.5.4)

2.4 Computability — Church-Turing Thesis

- **Church-Turing thesis** Every function that would naturally be regarded as computable can be computed by a deterministic Turing Machine.
- **physical Church-Turing thesis** Any finite physical system can be simulated (to any degree of approximation) by a Universal Turing Machine.
- **strong Church-Turing thesis** Any finite physical system can be simulated (to any degree of approximation) with polynomial slowdown by a Universal Turing Machine.
- **Shor's algorithm** (1994) — quantum algorithm for factoring integers — an NP problem that is not known to be P — also not known to be NP-complete and we have no proof that it is not in P

Reference: Section 4 of Unit 6 & 7 Reader

2.5 Computability — Turing Machine

- **Finite control** which can be in any of a finite number of *states*
- **Tape** divided into cells, each of which can hold one of a finite number of symbols
- Initially, the **input**, which is a finite-length string of symbols in the *input alphabet*, is placed on the tape
- All other tape cells (extending infinitely left and right) hold a special symbol called *blank*
- A **tape head** which initially is over the leftmost input symbol
- A **move** of the Turing Machine depends on the state and the tape symbol scanned
- A move can change state, write a symbol in the current cell, move left, right or stay

References: Hopcroft et al. (2007, page 326), Unit 6 & 7 Reader (section 5.3)

2.6 Turing Machine notation

- Q finite set of states of the finite control
- Σ finite set of *input symbols* (M269 S)
- Γ complete set of *tape symbols* $\Sigma \subset \Gamma$
- δ Transition function (M269 instructions, I)
 $\delta :: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$
 $\delta(q, X) \mapsto (p, Y, D)$
- q_0 start state $q_0 \in Q$
- B blank symbol $B \in \Gamma$ and $B \notin \Sigma$
- F set of *final or accepting states* $F \subseteq Q$

2.7 Computability — Decidability

- **Decidable** — there is a TM that will halt with yes/no for a decision problem — that is, given a string w over the alphabet of P the TM will halt and return yes/no the string is in the language P (same as *recursive* in [Recursion theory](#) — old use of the word)
- **Semi-decidable** — there is a TM will halt with yes if some string is in P but may loop forever on some inputs (same as *recursively enumerable*) — *Halting Problem*
- **Highly-undecidable** — no outcome for any input — *Totality, Equivalence Problems*
- **Halting problem** — the problem of deciding, given a program and an input, whether the program will eventually halt with that input, or will run forever — term first used by Martin Davis 1952
- **Entscheidungsproblem** — the problem of deciding whether a given statement is provable from the axioms using the rules of logic — shown to be undecidable by Turing (1936) by reduction from the *Halting problem* to it
- **Type inference and type checking** in the second-order lambda calculus (important for functional programmers, Haskell, GHC implementation)
- **Undecidable problem** — see link to list

([Turing, 1936, 1937](#))

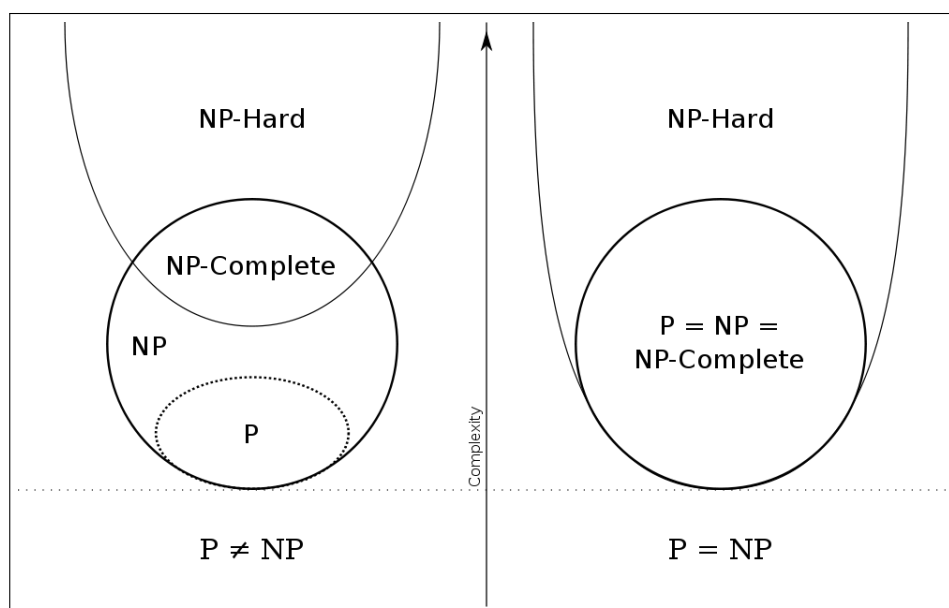
- A *problem* is really membership of a string in some language
- The number of different languages over any alphabet of more than one symbol is uncountable
- Programs are finite strings over a finite alphabet (ASCII or Unicode and hence countable).
- There must be an infinity (big) of problems more than programs.

Reference: [Hopcroft et al. \(2007, page 318\)](#)

2.8 Complexity

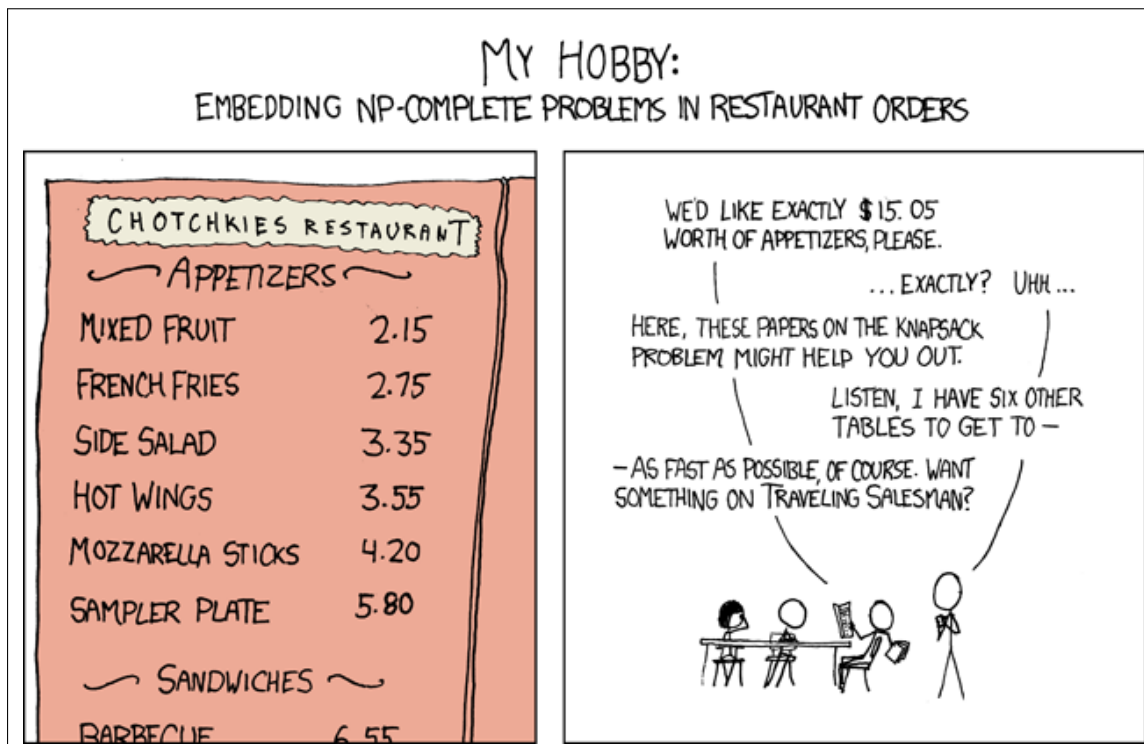
- **P**, the set of all decision problems that can be solved in polynomial time on a deterministic Turing machine
- **NP**, the set of all decision problems whose solutions can be verified (certificate) in polynomial time
- Equivalently, NP, the set of all decision problems that can be solved in polynomial time on a non-deterministic Turing machine
- A decision problem, dp is **NP-complete** if
 1. dp is in NP and
 2. Every problem in NP is reducible to dp in polynomial time
- **NP-hard** — a problem satisfying the second condition, whether or not it satisfies the first condition. Class of problems which are at least as hard as the hardest problems in NP. NP-hard problems do not have to be in NP and may not be decision problems

Euler diagram for P, NP, NP-complete and NP-hard set of problems



Source: [Wikipedia NP-complete entry](#)

- Boolean satisfiability (SAT) [Cook-Levin theorem](#)
- [Conjunctive Normal Form 3SAT](#)
- [Hamiltonian path problem](#)
- [Travelling salesman problem](#)
- **NP-complete** — see list of problems



Source & Explanation: [XKCD 287](#)

2.9 Logic

Q14 topics

- Unit 7
- Proofs
- Natural deduction
- A plethora of logics, proof systems, and different notations can be puzzling.
- [Martin Davis, Logician](#) When I was a student, even the topologists regarded mathematical logicians as living in outer space. Today the connections between logic and computers are a matter of engineering practice at every level of computer organization
- Various logics, proof systems, were developed well before programming languages and with different motivations,

References: [Davis \(1995, page 289\)](#)

- Turing machines, [Von Neumann architecture](#) and procedural languages Fortran, C, Java, Perl, Python, JavaScript
- Resolution theorem proving and logic programming — Prolog
- Logic and database query languages — SQL (Structured Query Language) and QBE (Query-By-Example) are syntactic sugar for first order logic
- [Lambda calculus](#) and functional programming with Miranda, Haskell, ML, Scala

Reference: [Halpern et al. \(2001\)](#)

2.10 Justified Arguments

Proofs, syntactic entailment, natural deduction

- Definition 7.1 An argument $\{P_1, P_2, \dots, P_n\} \vdash C$ is a justified argument if and only if either the argument is an instance of an axiom or it can be derived by means of an inference rule from one or more other justified arguments.

- Axioms

$$\Gamma \cup \{A\} \vdash A \text{ (axiom schema)}$$

- This can be read as: any formula A can be derived from the assumption (premise) of $\{A\}$ itself

See ([Thompson, 1991](#), Chp 1)

- Section 2.3 of Unit 7 (not the Unit 6, 7 Reader) gives the inference rules for $\rightarrow$, $\wedge$, and $\vee$ — only dealing with *positive propositional logic* so not making use of negation — see [List of logic systems](#)
- Usually (Classical logic) have a [functionally complete](#) set of logical connectives — that is, every binary Boolean function can be expressed in terms the functions in the set

Inference Rules — Notation

- Inference rule notation:

$$\frac{\text{Argument}_1 \quad \dots \quad \text{Argument}_n}{\text{Argument}} \text{ (label)}$$

Inference Rules — Conjunction

- $\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$ ($\wedge$ -introduction)
- $\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$ ($\wedge$ -elimination left)
- $\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$ ($\wedge$ -elimination right)

Inference Rules — Implication

- $\frac{\Gamma \cup \{A\} \vdash B}{\Gamma \vdash A \rightarrow B}$ ($\rightarrow$ -introduction)
- The above should be read as: If there is a proof (justification, inference) for B under the set of premises, Γ , augmented with A , then we have a proof (justification, inference) of $A \rightarrow B$, *under the unaugmented set of premises, Γ .*

The unaugmented set of premises, Γ may have contained A already so we cannot assume

$$(\Gamma \cup \{A\}) - \{A\} \text{ is equal to } \Gamma$$

- $\frac{\Gamma \vdash A \quad \Gamma \vdash A \rightarrow B}{\Gamma \vdash B}$ ($\rightarrow$ -elimination)

Inference Rules — Disjunction

- $\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}$ ($\vee$ -introduction left)

- $\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$ ($\vee$ -introduction right)
- Disjunction elimination

$$\frac{\Gamma \vdash A \vee B \quad \Gamma \cup \{A\} \vdash C \quad \Gamma \cup \{B\} \vdash C}{\Gamma \vdash C} \text{ (}\vee\text{-elimination)}$$

- The above should be read: if a set of premises Γ justifies the conclusion $A \vee B$ and Γ augmented with each of A or B separately justifies C , then Γ justifies C

Self-Assessment activity 7.4 — tree layout

- Let $\Gamma = \{P \rightarrow R, Q \rightarrow R, P \vee Q\}$
- $\frac{\Gamma \vdash P \vee Q \quad \Gamma \cup \{P\} \vdash R \quad \Gamma \cup \{Q\} \vdash R}{\Gamma \vdash R}$ ($\vee$ -elimination)
- $\frac{\Gamma \cup \{P\} \vdash P \quad \Gamma \cup \{P\} \vdash P \rightarrow R}{\Gamma \cup \{P\} \vdash R}$ ($\rightarrow$ -elimination)
- $\frac{\Gamma \cup \{Q\} \vdash Q \quad \Gamma \cup \{Q\} \vdash Q \rightarrow R}{\Gamma \cup \{Q\} \vdash R}$ ($\rightarrow$ -elimination)
- Complete tree layout
- $\frac{\Gamma \vdash P \vee Q \quad \frac{\frac{\Gamma \cup \{P\} \quad \Gamma \cup \{P\}}{\vdash P} \quad \frac{\vdash P \rightarrow R}{\vdash P \rightarrow R} \text{ (}\rightarrow\text{-E)}}{\Gamma \cup \{P\} \vdash R} \quad \frac{\frac{\Gamma \cup \{Q\} \quad \Gamma \cup \{Q\}}{\vdash Q} \quad \frac{\vdash Q \rightarrow R}{\vdash Q \rightarrow R} \text{ (}\rightarrow\text{-E)}}{\Gamma \cup \{Q\} \vdash R} \text{ (}\vee\text{-E)}}{\Gamma \vdash R}$

Self-assessment activity 7.4 — Linear Layout

1. $\{P \rightarrow R, Q \rightarrow R, P \vee Q\} \vdash P \vee Q$ [Axiom]
2. $\{P \rightarrow R, Q \rightarrow R, P \vee Q\} \cup \{P\} \vdash P$ [Axiom]
3. $\{P \rightarrow R, Q \rightarrow R, P \vee Q\} \cup \{P\} \vdash P \rightarrow R$ [Axiom]
4. $\{P \rightarrow R, Q \rightarrow R, P \vee Q\} \cup \{Q\} \vdash Q$ [Axiom]
5. $\{P \rightarrow R, Q \rightarrow R, P \vee Q\} \cup \{Q\} \vdash Q \rightarrow R$ [Axiom]
6. $\{P \rightarrow R, Q \rightarrow R, P \vee Q\} \cup \{P\} \vdash R$ [2, 3, $\rightarrow$ -E]
7. $\{P \rightarrow R, Q \rightarrow R, P \vee Q\} \cup \{Q\} \vdash R$ [4, 5, $\rightarrow$ -E]
8. $\{P \rightarrow R, Q \rightarrow R, P \vee Q\} \vdash R$ [1, 6, 7, $\vee$ -E]

2.11 Unit 6 Sets, Databases, Logic

Q13 Topics

- Unit 6
- SQL queries

Q12 Topics

- Unit 6
- Predicate Logic
- Translation to/from English
- Interpretations

Q11 Topics

- Unit 6

- Sets
- Propositional Logic
- Truth tables
- Valid arguments
- Infinite sets

3 Units 3, 4 & 5

3.1 Unit 5 Optimisation

- Unit 5 Optimisation
- Graphs searching: DFS, BFS
- Distance: Dijkstra's algorithm
- Greedy algorithms: Minimum spanning trees, Prim's algorithm
- Dynamic programming: Knapsack problem, Edit distance

Q10 Topics

Q9 Topics

3.2 Unit 4 Searching

- Unit 4 Searching
- String searching: Quick search Sunday algorithm, Knuth-Morris-Pratt algorithm
- Hashing and hash tables
- Search trees: Binary Search Trees
- Search trees: Height balanced trees: AVL trees

Q8 Topics

Q7 Topics

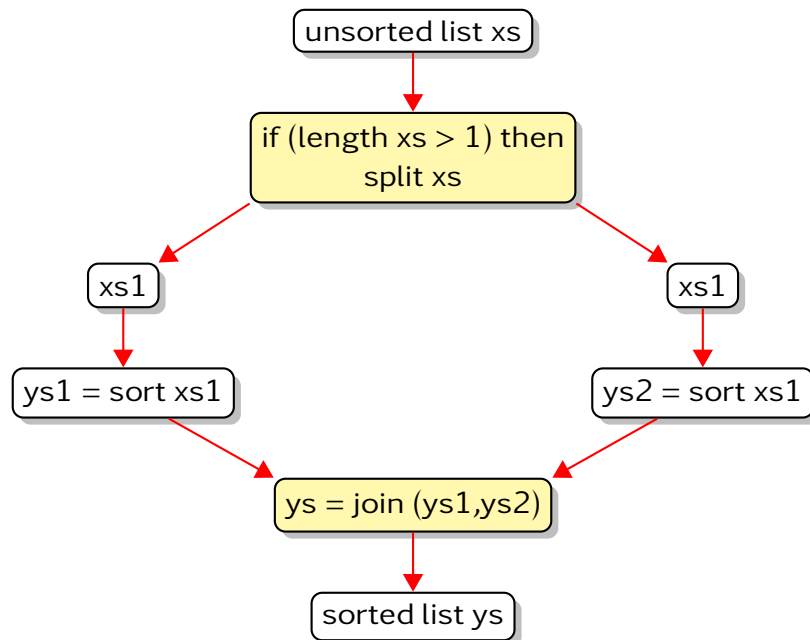
3.3 Unit 3 Sorting

- Unit 3 Sorting
- Elementary methods: Bubble sort, Selection sort, Insertion sort
- Recursion (see recursion)
- Quicksort, Merge sort
- Sorting with data structures: Tree sort, Heap sort
- See sorting notes for abstract sorting algorithm

Q6 Topics

Q5 Topics

3.4 Sorting Algorithms



Using the *Abstract sorting algorithm*, describe the *split* and *join* for:

- Insertion sort
- Selection sort
- Merge sort
- Quicksort
- Bubble sort (the odd one out)

4 Units 1 & 2

4.1 Unit 2 From Problems to Programs

- Unit 2 From Problems to Programs
- Abstract Data Types
- Pre and Post Conditions
- Logic for loops

Q4 Topics

Q3 Topics

4.2 Unit 1 Introduction

- Unit 1 Introduction
- Computation, computable, tractable
- Introducing Python
- What are the three most important concepts in programming ?
 1. **Abstraction**
 2. **Abstraction**
 3. **Abstraction**

Q2 Topics

Q1 Topics

5 M269 Exam Section B

Q16

- Multipart question
- **Binary Min Heap** is a **complete binary tree** with the min heap property
- **Min heap property** — each node is greater than or equal to its parent — a partial ordering
- **Heapsort**
 1. Build a heap
 2. Create sorted array/list by removing the root of the heap until it is empty

6 Exam Techniques

- Surviving in a time of great stress
- Each give one exam tip to the group
- TODO: add some more points

7 Web References

TODO: More Web links

Logic

- **WFF**, WFF'N Proof online <http://www.oercommons.org/authoring/1364-basic-wff-n-proof-a-teaching-guide/view>

Computability

- Computability
- Computable function
- Decidability (logic)
- Turing Machines
- Universal Turing Machine
- Turing machine simulator
- Lambda Calculus
- Von Neumann Architecture

Complexity

- Complexity class
- NP complexity
- NP complete
- Reduction (complexity)
- P versus NP problem
- Graph of NP-Complete Problems

References

- Adelson-Velskii, G M and E M Landis (1962). An algorithm for the organization of information. In *Doklady Akademii Nauk SSSR*, volume 146, pages 263–266. Translated from *Soviet Mathematics — Doklady*; 3(5), 1259–1263.
- Arora, Sanjeev and Boaz Barak (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press. ISBN 0521424267. URL <http://www.cs.princeton.edu/theory/complexity/>.
- Chiswell, Ian and Wilfrid Hodges (2007). *Mathematical Logic*. Oxford University Press. ISBN 0199215626.
- Church, Alonzo et al. (1937). Review: Am turing, on computable numbers, with an application to the entscheidungsproblem. *Journal of Symbolic Logic*, 2(1):42–43.
- Cormen, Thomas H.; Charles E. Leiserson; Ronald L. Rivest; and Clifford Stein (2009). *Introduction to Algorithms*. MIT Press, third edition. ISBN 0262533057. URL <http://mitpress.mit.edu/books/introduction-algorithms>.
- Davis, Martin (1995). Influences of mathematical logic on computer science. In *The Universal Turing Machine A Half-Century Survey*, pages 289–299. Springer.
- Davis, Martin (2012). *The Universal Computer: The Road from Leibniz to Turing*. A K Peters/CRC Press. ISBN 1466505192.
- Dowsing, R.D.; V.J Rayward-Smith; and C.D Walter (1986). *First Course in Formal Logic and Its Applications in Computer Science*. Blackwells Scientific. ISBN 0632013087.

- Franzén, Torkel (2005). *Gödel's Theorem: An Incomplete Guide to Its Use and Abuse*. A K Peters, Ltd. ISBN 1568812388.
- Fulop, Sean A. (2006). *On the Logic and Learning of Language*. Trafford Publishing. ISBN 1412023815.
- Garey, Michael R. and David S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman Co Ltd. ISBN 0716710455.
- Halbach, Volker (2010). *The Logic Manual*. OUP Oxford. ISBN 0199587841. URL <http://logicmanual.philosophy.ox.ac.uk/index.html>.
- Halpern, Joseph Y; Robert Harper; Neil Immerman; Phokion G Kolaitis; Moshe Y Vardi; and Victor Vianu (2001). On the unusual effectiveness of logic in computer science. *Bulletin of Symbolic Logic*, pages 213–236.
- Hodges, Wilfred (1977). *Logic*. Penguin. ISBN 0140219854.
- Hodges, Wilfred (2001). *Logic*. Penguin, second edition. ISBN 0141003146.
- Hopcroft, John E.; Rajeev Motwani; and Jeffrey D. Ullman (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, second edition. ISBN 0-201-44124-1.
- Hopcroft, John E.; Rajeev Motwani; and Jeffrey D. Ullman (2007). *Introduction to Automata Theory, Languages, and Computation*. Pearson, third edition. ISBN 0321514483. URL <http://infolab.stanford.edu/~ullman/ialc.html>.
- Hopcroft, John E. and Jeffrey D. Ullman (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, first edition. ISBN 020102988X.
- Lemmon, Edward John (1965). *Beginning Logic*. Van Nostrand Reinhold. ISBN 0442306768.
- Manna, Zohar (1974). *Mathematical Theory of Computation*. McGraw-Hill. ISBN 0-07-039910-7.
- Miller, Bradley W. and David L. Ranum (2011). *Problem Solving with Algorithms and Data Structures Using Python*. Franklin, Beedle Associates Inc, second edition. ISBN 1590282574. URL <http://interactivepython.org/courselib/static/pythonds/index.html>.
- Pelletier, Francis Jeffrey and Allen P Hazen (2012). A history of natural deduction. In Gabbay, Dov M; Francis Jeffrey Pelletier; and John Woods, editors, *Logic: A History of Its Central Concepts*, volume 11 of *Handbook of the History of Logic*, pages 341–414. North Holland. ISBN 0444529373. URL <http://www.ualberta.ca/~francisp/papers/PellHazenSubmittedv2.pdf>.
- Pelletier, Francis Jeffry (2000). A history of natural deduction and elementary logic textbooks. *Logical consequence: Rival approaches*, 1:105–138. URL <http://www.sfu.ca/~jeffpell/papers/pelletierNDtexts.pdf>.
- Rayward-Smith, V J (1983). *A First Course in Computability*. Blackwells Scientific. ISBN 0632011769.
- Rich, Elaine A. (2007). *Automata, Computability and Complexity: Theory and Applications*. Prentice Hall. ISBN 0132288060. URL <http://www.cs.utexas.edu/~ear/cs341/automatabook/>.

- Smith, Peter (2003). *An Introduction to Formal Logic*. Cambridge University Press. ISBN 0521008042. URL <http://www.logicmatters.net/ifl/>.
- Smith, Peter (2007). *An Introduction to Gödel's Theorems*. Cambridge University Press, first edition. ISBN 0521674530.
- Smith, Peter (2013). *An Introduction to Gödel's Theorems*. Cambridge University Press, second edition. ISBN 1107606756. URL <http://godelbook.net>.
- Smullyan, Raymond M. (1995). *First-Order Logic*. Dover Publications Inc. ISBN 0486683702.
- Teller, Paul (1989a). *A Modern Formal Logic Primer: Predicate and Metatheory: 2*. Prentice-Hall. ISBN 0139031960. URL <http://tellerprimer.ucdavis.edu>.
- Teller, Paul (1989b). *A Modern Formal Logic Primer: Sentence Logic: 1*. Prentice-Hall. ISBN 0139031707. URL <http://tellerprimer.ucdavis.edu>.
- Thompson, Simon (1991). *Type Theory and Functional Programming*. Addison Wesley. ISBN 0201416670. URL <http://www.cs.kent.ac.uk/people/staff/sjt/TTFP/>.
- Tomassi, Paul (1999). *Logic*. Routledge. ISBN 0415166969. URL <http://emilkirkegaard.dk/en/wp-content/uploads/Paul-Tomassi-Logic.pdf>.
- Turing, Alan Mathison (1936). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265.
- Turing, Alan Mathison (1937). On computable numbers, with an application to the entscheidungsproblem. a correction. *Proceedings of the London Methemathical Society*, 43:544–546.
- van Dalen, Dirk (1994). *Logic and Structure*. Springer-Verlag, third edition. ISBN 0387578390.
- van Dalen, Dirk (2012). *Logic and Structure*. Springer-Verlag, fifth edition. ISBN 1447145577.
- V.J.Rayward-Smith (1985). *A First Course in Computability*. Blackwells Scientific. ISBN 0632013079.