M269

Exam Revision

Phil Molyneux

17 May 2014

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

M269 Exam

Revision strategies

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

M269 Specimen Exam

Q15 Topics

- Unit 7
- Computability and ideas of computation
- Complexity
- P and NP
- NP-complete

Computability

Ideas of Computation

- 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)

Computability

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

Computability

Direction of Reduction

- 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 an known undecidable problem to it
- Since, if P_1 is undecidable then P_2 is undecidable

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)

Computability

Church-Turing Thesis & Quantum Computing

- **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

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

Computability

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)
  $\delta :: Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$
  $\delta(q, X) \mapsto (p, Y, D)$
- 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* $F \subseteq Q$

- **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 with 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*

Computability

Undecidable 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

Computability

Why undecidable problems must exist

- 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.

Complexity

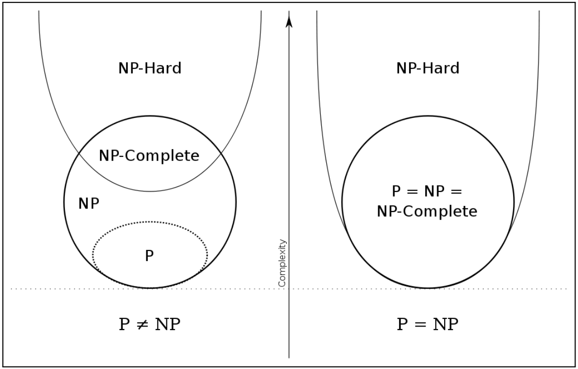
P and NP

- 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

Complexity

P and NP — Diagram

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



Source: Wikipedia NP-complete entry

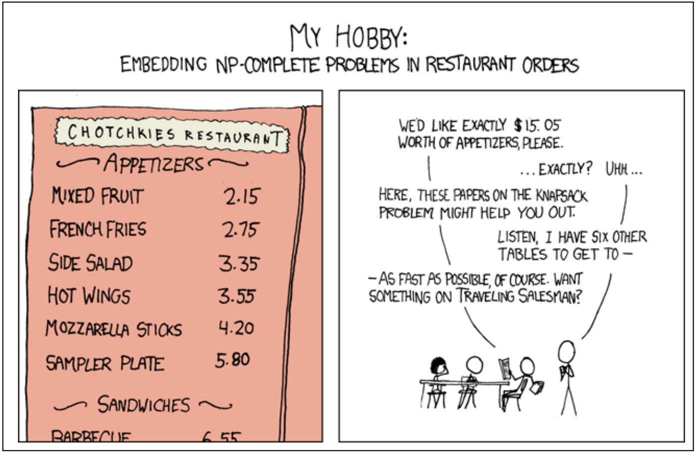
Complexity

NP-complete problems

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

Complexity

Knapsack Problem



Source & Explanation: XKCD 287

M269 Specimen Exam

Q14 topics

- Unit 7
- Proofs
- Natural deduction

Logic

Logicians, Logics, Notations

- 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,

Logic

Logic and Programming Languages

- 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

Logic

Justified Arguments

- Definition 7.1 An argument $\{P_1, P_2, \ldots, P_n\} \vdash C$ 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

Logic

Justified Arguments

- 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

Justified Arguments

Inference Rules — Notation

- Inference rule notation:

$$\frac{Argument_1 \quad \ldots \quad Argument_n}{Argument}\ (label)$$

Justified Arguments

Inference Rules — Conjunction

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

Justified Arguments

Inference Rules — Implication

- $\dfrac{\Gamma \cup \{A\} \vdash B}{\Gamma \vdash A \to B}$ ($\to$-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 $A \to B$, *under the unaugmented set of premises,* Γ.

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

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

- $\dfrac{\Gamma \vdash A \qquad \Gamma \vdash A \to B}{\Gamma \vdash B}$ ($\to$-elimination)

Justified Arguments

Inference Rules — Disjunction

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

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

- 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

Justified Arguments

Self-Assessment activity 7.4

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

Justified Arguments

Self-assessment activity 7.4 — Linear Layout

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

M269 Specimen Exam
Q13 Topics

- Unit 6
- SQL queries

M269 Specimen Exam

Q12 Topics

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

M269 Specimen Exam

Q11 Topics

- Unit 6
- Sets
- Propositional Logic
- Truth tables
- Valid arguments
- Infinite sets

M269 Specimen Exam

Unit 5 Topics, Q9, Q10

- 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

M269 Specimen Exam

Unit 4 Topics, Q7, Q8

- 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

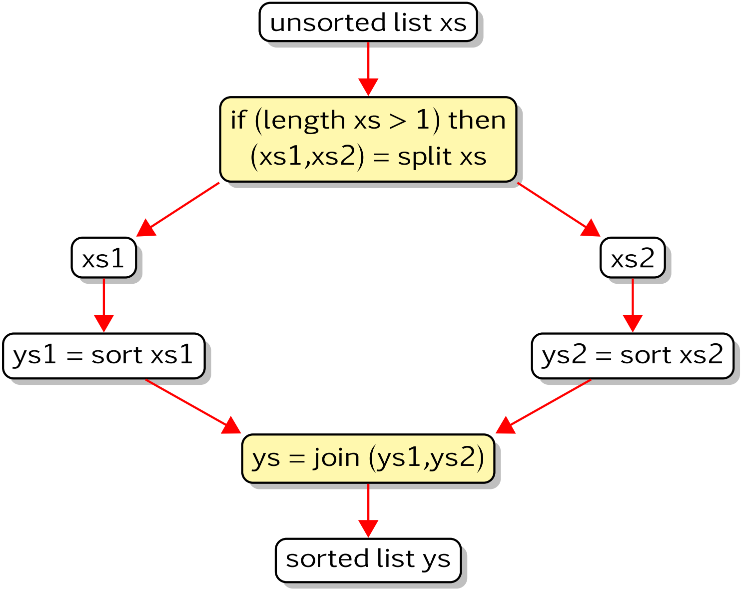
M269 Specimen Exam

Unit 3 Topics, Q5, Q6

- 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

Unit 3 Sorting

Abstract Sorting Algorithm



Unit 3 Sorting

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)

M269 Specimen Exam

Unit 2 Topics, Q3, Q4

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

M269 Specimen Exam

Unit 1 Topics, Q1, Q2

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

M269 Specimen Exam

Unit 1 Topics, Q1, Q2

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

M269 Specimen Exam

Unit 1 Topics, Q1, Q2

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

M269 Specimen Exam

Unit 1 Topics, Q1, Q2

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

M269 Exam Section B

Q16 Topics

- Multipart question

Q16 Topics

Binary Min Heap & Heapsort

- **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

M269 Exam

Exam Techniques

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