Revision Lecture CS-226

Exams from Previous Years

- The lecture 2003/04 was given by A. Setzer, and therefore the exam was prepared by A. Setzer.
  - So that exam is a good model of the style of A. Setzer's exams.
- The lecture 2004/05 was given by M. Seisenberger, who was using A. Setzer's notes.
  - So the questions would be suitable. Some questions differ in style from A. Setzer's but the material covered is essentially the same.

Exams from Previous Years

- The lectures before 2003/04 were given by U. Berger and others.
  - U. Berger taught some material (and asked questions in his exams about it), which was not covered in A. Setzer's lectures.
    - Reducibility, Rice's theorem, λ-terms, complexity theory were not covered in this lecture.
  - U. Berger's notation of Turing machines and URMs was slightly different.

Coursework and Exam

- Coursework is very relevant.
  - All coursework was designed as a preparation for the exam.
  - Coursework questions are often more complex versions of typical exam questions.
- Therefore revision of the coursework and the solutions of the coursework is highly recommended.
  - Solutions for coursework2 are available in the trays in the student office.

Structure of the Exam

- As usual, there will be 3 questions out of which you should choose 2.
  - Some students answer all 3 questions.
  - That's only recommended if one notices too late that there is a problem with one question, and chooses to answer another question instead.
  - In most cases, if all 3 questions were answered by a student, the first 2 answered were the best ones – the students didn't have time to answer the 3rd one properly.
    - Then it would have been better, to answer the first 2 questions more thoroughly, or spend time on revising the answers, rather than answering the 3rd question, which then doesn't count.

Sealing of names

- When sealing your names, please make it in such a way that it can be broken easily by
  - folding the part containing your name,
  - and putting the sticker so that it coveres one corner of it and hides your name.
- The lecturer has to break the seals, once the exam has been marked, so that he can enter the marks into the spreadsheets.
- This is a lot of work if sealing is done improperly.
  - Note that Marking itself is done completely blindly.

General Comments

- Some book work questions, but most questions ask to
  - calculate something,
  - determine properties of something,
  - prove something.
    - Proofs are usually much simpler than the main proofs carried out in the lecture.
- **(Added)** I will come to the exam in the first half hour (approx.; in case you have questions).

General Comments

- Typical question are:
  - Show that certain functions/sets are or are not countable, TM-computable, URM-computable, recursive etc.
  - Determine, which function is computed by a URM program, Turing machine.
  - Does a certain statement hold (e.g. if two sets are countable, so is their union). Justify your answer.

General Comments

- All answers need to be justified.
  - Without an argument, why your answer is correct, you get always less marks.
- **Main topics:**
  - Encoding of Data Types into $\mathbb{N}$.
  - URMs
  - Turing Machines.
  - Primitive recursive and partial recursive functions, including
    - Church's thesis,
    - recursion theorem.
    - recursively enumerable predicates.

Structure of the Exam

(Inserted slide)

- **Question 1** will be **mainly** on cardinality, countable sets, diagonalisation arguments.
- **Question 2** will be **mainly** on Turing machines and URMs.
- **Question 3** will be **mainly** on primitive-recursion, recursive functions, Church Thesis, Recursion theorem, rec. enum.

Sect. 1: Introduction

- Mainly of introductory character.
- Not directly needed.
- Especially, history of computability theory not relevant for the exam.

Sect. 2: Encoding into $\mathbb{N}$

- Mathematical background will not be asked directly,
  - but plays a role everywhere in the module.
  - Especially the notion of a partial function is used heavily.
- Definition of notion “two sets being equinumerous” (having the same cardinality).
  - Note that this is a relationship between two sets.
  - Please note that in previous years (and therefore previous exams) we used “have the same cardinality” instead of “equinumerous” and “$\simeq$” instead of “$\approx$”
    - There is a notion of cardinality, but that would have been too difficult for this module.
    - What you can understand is the notion of “equinumerous”.

Sect. 2: Encoding into $\mathbb{N}$

- Relationship to number of elements of a finite set.
- Proof of $A \not\approx \mathcal{P}(A)$.
  - Example of a diagonalisation argument.
  - In general proofs using diagonalisation arguments might be important (see questions on the exam 2003/04).

Sect. 2: Encoding into $\mathbb{N}$

- Various characterisations what it means to be countable or uncountable.
  - Lemma 2.11, Corollary 2.13, Lemma 2.14, Corollary 2.15.
    - E.g. A is countable if there exists an injective $f : A \to \mathbb{N}$ or a surjective $f : \mathbb{N} \to A$.
  - Only the statement, proofs are not needed directly, but might help the intuition.
    - Used for showing that certain sets are countable, uncountable.

Sect. 2: Encoding into $\mathbb{N}$

- Lemma 2.16 (Examples of uncountable sets), Theorem 2.26 (Examples of countable sets and closure of countable sets) provide us with some examples and reference sets, and show closure properties of the countable sets.
  - Proofs especially of Theorem 2.26 are important (both directly and as an indication of how to prove that a set is countable).

Sect. 2: Encoding into $\mathbb{N}$

- Encoding of sequences into $\mathbb{N}^*$.
  - No proofs needed.
  - What kind of functions do we have
    - π, π_i, π^k, π_i^k, $(n)_i$, $\langle n_0, \ldots, n_{k-1} \rangle$, lh).
  - Formulae for calculating π.
  - Properties of these functions
    - (π is bijective, π_i inverse to π, π_i^k inverse to π^k, etc.)
- Reduction of computability to $\mathbb{N}$.
  - Only some general understanding.

Sect. 2: Encoding into $\mathbb{N}$

- In general you should know, whether certain sets are countable, e.g.
  - $\mathbb{N}$ (is countable),
  - $\mathcal{P}(\mathbb{N})$ (is uncountable),
  - $\mathbb{R}$ (is uncountable),
  - $\mathbb{Q}$ (is countable),
  - the set of finite subsets of $\mathbb{N}$ (is countable).

Sect. 2: Encoding into $\mathbb{N}$

- Complements of countable sets in uncountable sets are uncountable,
  - e.g. the set of infinite subsets of $\mathbb{N}$ is the complement of the set of finite subsets of $\mathbb{N}$; $\mathcal{P}(\mathbb{N})$ is uncountable, the set of finite subsets is countable, therefore the set of infinite subsets of $\mathbb{N}$ is uncountable.
- Subsets of countable sets are countable.
  - e.g. the set of even numbers is countable, since it is a subset of $\mathbb{N}$.
- Supersets of uncountable sets are uncountable.

Sect. 3: The URM

- Definition of a URM and URM programs, and of how a URM operates.
- Partial function $\mathrm{U}^{(n)}$ computed by a URM.
- Translation of high level constructs into low level lecture.
  - Technique of interest, not the details of these particular constructions.
  - You might use the high level constructs (unless you are explicitly asked for a URM program as defined in the lecture).

Sect. 3: The URM

- Undecidability of the Halting problem.
  - Proof is a diagonalisation argument and might be relevant.
  - See as well related proofs, e.g. Coursework 1, Qu. 6.
- In principal you should be able to
  - carry out a run of a URM (write down states and register contents during an example run with some given inputs);
  - write URM-programs which compute a function;
  - write URM-programs which achieve something (e.g. exchange content of two registers);
  - determine, what a given URM program does.

Sect. 4: Turing Machines

- Motivation, why we use TMs (for book work/essay-type like questions)
- Definition of a TM, and how it operates.
- Definition of the function $\mathrm{T}^{(n)}$ computed by a TM.
- Simulation of URMs by TMs too difficult.
  - But the proof contains ideas of how to write TMs for certain tasks.

Sect. 4: Turing Machines

- In principal you should be able to
  - Carry out a run of a TM (write down the tape, head position and state during an example run with some given inputs).
  - Write a TM, which computes a given function.
  - Write a TM, which achieves something (e.g. exchanges certain symbols on the tape).
  - Determine, what a given TM does.

Sect. 5: Algebraic View

- Definition of primitive-recursive functions and predicates.
  - The basic functions are primitive recursive.
  - Primitive-recursive functions are closed under composition.
  - Primitive-recursive functions closed under the principle of primitive recursion.
- Notations for primitive-recursive functions are useful, but you can answer questions without introducing notations.

Sect. 5: Algebraic View

- For showing directly (by using the definition of primitive-recursive functions only) that a function $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ is primitive recursive it suffices to
  - express $f(\vec{x}, 0)$ as an expression which which makes use of the constant 0, the successor function (or equivalently of addition of 1) and variables;
  - express $f(\vec{x}, n+1)$ as an expression which which makes use of the constant 0, the successor function (or equivalently of addition of 1) and variables, and $f(\vec{x}, n)$.

Sect. 5: Algebraic View

- For instance, the following shows directly that $f : \mathbb{N} \to \mathbb{N}$, $f(x) = 2x + 1$ is primitive recursive:
  - $f(0) = 1$.
  -

$$\begin{aligned} f(x+1) &= 2(x+1)+1 \\ &= 2x+1+2 \\ &= f(x)+1+1 = \mathsf{succ}(\mathsf{succ}(f(x))) \quad . \end{aligned}$$

Sect. 5: Algebraic View

- For showing that a function f is primitive recursive in general one can
  - express $f(\vec{x})$ as a term using constants, x_i, other primitive recursive functions, and operations under which primitive recursive functions are closed (especially case distinction);
    - e.g. $f(x) = \sum_{i<x+7} \chi_A(i) \cdot i$ is primitive-recursive, if A is a primitive recursive predicate;
  - or express
    - $f(\vec{x}, 0)$ by an expression as before,
    - $f(\vec{x}, y+1)$ by an expression, which can use the same as before, but additionally $f(\vec{x}, y)$.

Sect. 5: Algebraic View

- E.g., let
$$f(x) := |\{y < x \mid P(x)\}| \ .$$
- $f(x)$ is the number of $y < x$ s.t. $P(y)$ holds. f is primitive recursive, since
  - $f(0) = 0$;
  - $f(x+1) = \begin{cases} f(x)+1, & \text{if } P(x), \\ f(x), & \text{otherwise.} \end{cases}$

Sect. 5: Algebraic View

- Technique for determining primitive-recursively a $y < x$ s.t. a property holds.
  - E.g. find the second least $y < x$ s.t. $Q(y)$ holds.
- Steps:
  - Express the property as a predicate $P(y, x)$ s.t. $P(y, x)$ holds for at most one y.
  - Define f as

$$f(x) = \begin{cases} y, & \text{if } y < x \text{ and } P(y, x), \\ x, & \text{if there is no } y < x \text{ s.t. } P(y, x). \end{cases}$$

Sect. 5: Algebraic View

- Show that P is primitive-recursive
  - In the example:

$$P(y,x) :\Leftrightarrow$$
$$Q(y) \wedge (\exists z < y.Q(z)) \wedge$$
$$\neg(\exists z < y.\exists z' < y.(Q(z) \wedge Q(z') \wedge z \neq z'))$$

- So $P(x,y)$ holds if $Q(x)$ holds and there is exactly one z below x s.t. $Q(z)$ holds as well.
- So $P(y,x)$ holds, if y is the second least y' s.t. $Q(y')$ holds.
- The formula is primitive recursive, since all quantifiers are bounded (i.e. are of the form "$\exists z < y$").

Sect. 5: Algebraic View

- Then we get $f(x) = \mu y < x.P(y, x)$.
- f is primitive recursive.

Sect. 5: Algebraic View

- Note that often one can show that a function or predicate is primitive recursive by using directly the closure properties.
  - E.g. the predicate

$$P(x) :\Leftrightarrow x \text{ is a power of } 2$$

is primitive recursive, since

$$P(x) \Leftrightarrow \exists y \leq x.x = 2^y$$

and primitive recursive predicates are closed under bounded quantification, equalities between primitive recursive terms, and the exponential function is prim. rec.

Sect. 5: Algebraic View

- Examples of primitive-recursive functions, closure of primitive recursive functions under certain operations (including course-of-values recursion).
  - Proofs only relevant as example techniques for showing that a function is primitive-recursive.
  - But under which operations they are closed is important for showing that certain functions are primitive-recursive.

Sect. 5: Algebraic View

- Ackermann function as an example of a non primitive-recursive but recursive (computable) function.
  - Definition.
  - Roughly, why it is not primitive recursive.
    - Use of Lemma 5.9. (no proof of this lemma needed, only the statement).
    - Then a diagonalisation argument.
- The direct argument (without using the Ackermann function) that there are non primitive-recursive computable functions is an example of a diagonalisation argument, which might be relevant.

Sect. 5: Algebraic View

- Definition of partial recursive functions, recursive functions, recursive predicates.
- All primitive recursive functions are recursive (i.e. computable).
- Closure properties of recursive functions/relations.
- Equivalence of URM-, TM-computable functions and partial recursive functions (no proof).

Sect. 6 (a): Equivalence Theorem

- Simulation of TM programs by partial recursive functions.
  - Proof too complicated for the exam.
  - Only Kleene's Normal form theorem and the following theorems and definitions are important.
    - Especially, each partial recursive function is of the form $\{e\}^n$ for some e and n.
    - $\{e\}^n(\vec{x}) \simeq \mathsf{U}(\mu u.\mathsf{T}^n(e, \vec{x}, u))$.
  - Otherwise only a rough idea how to do it.

Sect. 6 (a): Equivalence Theorem

- In principal you should be able to
  - show that a function/predicate is recursive, partial-recursive, primitive-recursive.
- Equivalence theorem.
  - The basic ideas of the proofs and the structure of the whole proof (round robin).

Sect. 6 (b): Church-Turing Thesis

- In general the statement and arguments in favour of it are important.
- Would be typically an essay-like question.

Sect. 7: Kleene's Recursion Theore

- You should know Kleene's recursion theorem, and how to use it in order to show that certain functions, defined recursively are partial recursive.
  - See the examples (e.g. fib, Ackermannn).
- Of the S-m-n theorem, the statement might be important.

Sect. 8: R.e. Predicates

- Definition of recursively enumerable.
- An idea about why recursively enumerable predicates are relevant for computer science.
- The theorems and lemmata, without any proofs (except for closure properties of the rec. enum. sets).

Sect. 8: R.e. Predicates

- Esp.
  - Characterisation of recursively enumerable predicates (Theorem 8.5), especially

    A is r.e. $\Leftrightarrow A = \{\vec{x} \mid \exists y.R(\vec{x}, y)\}$ for some prim.-rec. R .

  - Not all recursively enumerable predicates are recursive.
  - A recursive iff A and $\mathbb{N}^n \setminus A$ are r.e.
  - Closure properties of partial recursive functions (with proofs).
  - R.e. sets are not closed under complement.