CS_226 Computability Theory

http://www.cs.swan.ac.uk/∼csetzer/lectures/computability/08/index.html

Course Notes, Michaelmas Term 2008

Anton Setzer
(Dept. of Computer Science, Swansea)
http://www.cs.swan.ac.uk/∼csetzer/index.html

Examples

- Define *exp* : $\mathbb{N} \to \mathbb{N}$, $\mathsf{exp}(n) := 2^n$,

  where $\mathbb{N} = \{0, 1, 2, \ldots\}$.
  exp is *computable*.
  However, can we really compute
  $\mathsf{exp}(10000000000000000000000000000000000)$?
- Let *String* be the set of strings of ASCII symbols.

  Define a function *check* : String $\to \{0, 1\}$ by

$$\mathsf{check}(p) := \begin{cases} 1 & \text{if } p \text{ is a syntactically correct Java program,} \\ 0 & \text{otherwise.} \end{cases}$$

Is check computable or not?

The Topic of Computability Theory

- A **computable function** is a function

$$f : A \to B$$

  such that there is a *mechanical procedure* for computing for every $a \in A$ the result $f(a) \in B$.
- **Computability theory** is the study of computable functions.
- In computablitity theory we explore the *limits* of the notion of computability.

Remark on Variables

- In this lecture I will often use i, j, k, l, m, n for variables denoting natural numbers.
- I will often use p, q and some others for variables denoting programs.
- I will use z for integers.
- Other letters might be used as well for variables.
- These conventions are not treated very strictly.
  - Especially when running out of letters.

Examples (Cont.)

- Define a function $\underset{\sim}{terminate}$: String $\to \{0,1\}$,

$$\text{terminate}(p) := \begin{cases} 1 & \text{if } p \text{ is a syntactically correct} \\ & \text{Java program with no input and outputs,} \\ & \text{which terminates;} \\ 0 & \text{otherwise.} \end{cases}$$

Is terminate *computable?*

Examples (Cont.)

- Define a function $\underset{\sim}{issortingfun}$: String $\to \{0,1\}$,

$$\text{issortingfun}(p) := \begin{cases} 1 & \text{if } p \text{ is a syntactically correct} \\ & \text{Java program, which has as inp} \\ & \text{a list and returns a sorted list,} \\ 0 & \text{otherwise.} \end{cases}$$

Is issortingfun *computable?*

Answer

- ____________________________
  (To be filled in during the lecture)

Explanation

- Assume issortingfun were computable.
- Then we can construct (compute) a program which computes terminate as follows:
  - Assume as input a string p.
  - Check whether it is a syntactically correct Java program with no input and outputs.
  - If no, terminate$(p) = 0$, so return 0.
  - Otherwise, create a program which is a potential sorting function as follows:
    - It takes as input a list l.
    - Then this program runs p.
    - If p has terminated, then it runs a known sorting function on l, and returns the result.

Explanation

- Let the resulting program (which depends on p) be $q(p)$.
- If p terminates, then $q(p)$ will be a sorting function, so $\mathsf{issortingfun}(q(p)) = 1 = \mathsf{terminate}(p)$.
- If p does not terminate, then $q(p)$ does not terminate on any input, so $\mathsf{issortingfun}(q(p)) = 0 = \mathsf{terminate}(p)$.
- Our program returns now $\mathsf{issortingfun}(q(p))$ which is the result of $\mathsf{terminate}(p)$.

- So we have obtained by using a program for issortingfun a program which computes terminate.
- But terminate is non-computable, therefore issortingfun cannot be computable.

Three Areas

Three Areas are involved in computability theory.

- **Mathematics.**
  - Precise definition of computability.
  - Analysis of the concept.
- **Philosophy.**
  - Validation that notions found are the correct ones.
- **Computer science.**
  - Study of relationship between these concepts and computing in the real world.

Problems in Computability

In order to understand and answer the questions we have to

- Give a precise definition of what *computable* means.
  - That will be a **mathematical definition**.
  - Such a notion is particularly important for showing that certain functions are *non-computable*.
- Then provide evidence that the definition of "*computable*" is the correct one.
  - That will be a **philosophical argument**.
- Develop methods for proving that certain functions are *computable or non-computable*.

Questions Related to The Above

- Given a function $f : A \to B$, which can be computed, can it be done *effectively*? (**Complexity theory**.)
- Can the task of deciding a given problem $P1$ be reduced to deciding another problem $P2$? (**Reducibility theory**).

More Advanced Questions

The following is beyond the scope of this module.

- Can the notion of *computability* be extended to computations on *infinite objects*
  (e.g. streams of data, real numbers,higher type operations)?
  (**Higher and abstract computability theory**).
- What is the relationship between *computing* (producing actions, data etc.) and *proving*.

Idealisation

In computability theory, one usually abstracts from limitations on

- time and
- space.

A problem will be computable, if it can be solved on an *idealised computer*, even if it the computation would take longer than the life time of the universe.

History of Computability Theory

Gottfried Wilhelm von Leibnitz (1646 – 1716)

- Built a first *mechanical calculator.*
- Was thinking about a machine for manipulating symbols in order to determine truth values of mathematical statements.
- Noticed that this requires the definition of a precise *formal language.*

History of Computability Theory

David Hilbert (1862 – 1943)

- Poses 1900 in his famous list “Mathematical Problems” as 10th problem to decide *Diophantine equations.*
  Jump over Explanation Diophantine Equations

Diophantine Equations

- Here is a short description of Diophantine Equations.
- This is the question, whether an indeterminate polynomial equation has solutions where the variables are instantiated as integers.
- Examples:
  - Solve for integers a, b the equation $ax + by = 1$ using integers x, y.
  - Solve for given n the equation $x^n + y^n = z^n$.
    - For $n \geq 3$ this is unsolvable by Fermat's Last Theorem.

Decision Problem

- **Hilbert (1928)**
  - Poses the *Entscheidungsproblem* (German for decision problem).
  - The *decision problem* is the question, whether we can decide whether a formula in predicate logic is provable or not.
    - *Predicate logic* is the standard formalisation of logic with connectives $\wedge, \vee, \rightarrow, \neg$ and quantifiers $\forall$, $\exists$.
    - Predicate logic is "*sound and complete*".
      - This means that a a formula is provable if and only if it is valid (in all models).

Decision Problem

  - So the decidability of predicate logic is the question whether we can decide whether a formula is valid (in all models) or not.
  - If predicate logic were decidable, provability in mathematics would become trivial.
- "**Entscheidungsproblem**" became one of the few German words which have entered the English language.

History of Computability Theory

- **Gödel, Kleene, Post, Turing (1930s)**
  Introduce different *models of computation* and prove that they all define the same class of computable functions.

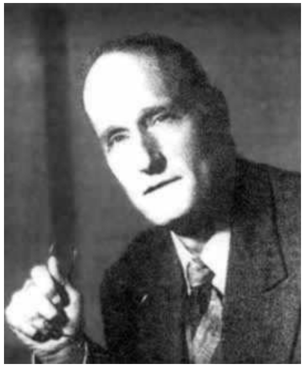
History of Computability Theory

Kurt Gödel (1906 – 1978)
Introduced the (Herbrand-Gödel-)
recursive functions
in his 1933 - 34 Princeton lectures.

History of Computability Theory

Emil Post
(1897 – 1954)
Introduced the Post proble

History of Computability Theory



Stephen Cole Kleene
(1909 – 1994)
Probably the most influential
computability theoretist up to now.
Introduced the partial recursive
functions.

History of Computability Theory

Alan Mathison Turing
(1912 – 1954)
Introduced the Turing machine.
Proved the undecidability
of the Turing-Halting problem.

Gödel's Incompleteness Theorem

- **Gödel (1931)** proves in his first incompleteness theorem:
  - Every reasonable primitive-recursive theory is incomplete, i.e. there is a formula s.t. neither the formula nor its negation is provable.
    - The theorem generalises to recursive i.e. computable theories.
    - The notions "primitive-recursive" and "recursive" will be introduced later in this module. For the moment it suffices to understand "recursive" informally as intuitively computable.

Gödel's Incompleteness Theorem

- Therefore no computable theory proves all true formulae.
- Therefore, it is undecidable whether a formula is true or not.
  - Otherwise, the theory consisting of all true formulae would be a complete computable theory.

Undecidability of the Decision Pr

- **Church, Turing (1936)** postulate that the models of computation established above define exactly the set o all computable functions (Church-Turing thesis).
- Both established independetly undecidable problems and proved that the **decision problem** is **undecidable** i.e. **unsolvable**.
  - Even for a **class of very simple formulae** we cannot decide the decision problem.

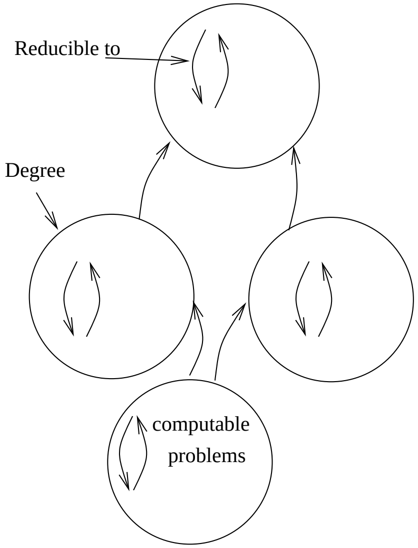
Undecidability of the Decision Pr

- Church shows the undecidability of equality in the λ-calculus.
- Turing shows the unsolvability of the **halting problem**.
  - It is undecidable whether a Turing machine (and by the Church-Turing thesis equivalently any non-interactive computer program) eventually stops.
  - That problem turns out to be the most important undecidable problem.

History of Computability Theory

Alonzo Church (1903 - 1995)

Degrees



History of Computability Theory

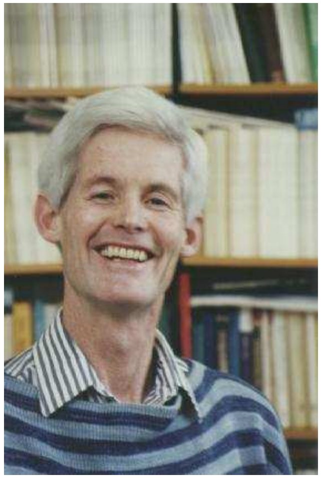
- **Post (1944)** studies degrees of unsolvability. This is the birth of degree theory.
- In degree theory one devides problems into groups ("**degrees**") of problems, which are reducible to each other.
  - **Reducible** means essentially "relative computable".
- Degrees can be ordered by using reducibility as ordering.
- The question in degree theory is: what is the structure of degrees?

History of Computability Theory

Yuri Vladimirovich Matiyasevich ($*$ 1947)

- Solves 1970 Hilbert's 10th problem negatively: The solvability of Diophantine equations is undecidable.

History of Computability Theory



Stephen Cook(Toronto)

- **Cook (1971)** introduces the complexity classes **P** and **NP** and formulates the problem, whether **P** $\neq$ **NP**.

Current State

- Concurrent and game-theoretic models of computation are developed (e.g. Prof. Moller in Swansea).
- Automata theory further developed.
- Alternative models of computation are studied (quantum computing, genetic algorithms).
- ...

Current State

- The problem **P** $\neq$ **NP** is still open. Complexity theory has become a big research area.
- Intensive study of computability on infinite objects (e.g. real numbers, higher type functionals) is carried out (e.g. U. Berger, Jens Blanck and J. Tucker in Swansea).
- Computability on inductive and co-inductive data types is studied.
- Research on program synthesis from formal proofs (e.g. U. Berger and M. Seisenberger in Swansea).

Name “Computability Theory”

- The original name was *recursion theory*, since the mathematical concept claimed to cover exactly the computable functions is called “recursive function”.
- This name was changed to *computability theory* during the last 10 years.
- Many books still have the title “recursion theory”.

Administrative Issues

Lecturer:
Dr. A. Setzer
Dept. of Computer Science
University of Wales Swansea
Singleton Park
SA2 8PP
UK

Room:	Room 211, Faraday Building
Tel.:	(01792) 513368
Fax:	(01792) 295651
Email:	a.g.setzer@swansea.ac.uk
Home page:	http://www.cs.swan.ac.uk/~csetzer/index.html

Course Home Page

- Located at
  `http://www.cs.swan.ac.uk/~csetzer/lectur`
  `computability/08/index.html`
- There is an open version,
- and a password protected version.
- The password is ____________.
- Errors in the notes will be corrected on the slides and noted on the list of errata.
- In order to reduce plagarism, coursework and solution to coursework will **not** be made available in electronic form (e.g. on this web site).

Assessment:

- 80% Exam.
- 20% Coursework.

Plan for this Module

1. Introduction.
2. Encoding of data types into $\mathbb{N}$.
3. The Unlimited Register Machine (URM) and the halting problem.
4. Turing machines.
5. The primitive recursive functions.
6. The recursive functions and the equivalence theorem.
7. The recursion theorem.
8. Semi-computable predicates.

Aims of this Module

- To become familiar with fundamental **models of computation** and the relationship between them.
- To develop an appreciation for the **limits of computation** and to learn **techniques** for **recognising unsolvable** or unfeasible **computational problems**.
- To understand the **historic** and **philosophical background** of computability theory.
- To be aware of the **impact** of the fundamental results of computability theory **to** areas of **computer science** such as **software engineering** and **artificial intelligence**.

Literature

- Cutland: *Computability*. Cambridge University Press, 1980.
  - Main text book.
- Thomas A. Sudkamp: *Languages and machines.* 3rd Edition, Addison-Wesley 2006.
- George S. Boolos, Richard C. Jeffrey, John Burgess: *Computability and logic.* 5th Ed. Cambridge Univ. Press, 2007
- Lewis/Papadimitriou: *Elements of the Theory of Computation*. Prentice Hall, 2nd Edition, 1997.
- Sipser: *Introduction to the Theory of Computation*. PWS Publishing. 2nd Edition, 2005.

Aims of this Module

- To understand the close **connection** between **computability theory** and **logic**.
- To be aware of **recent concepts** and **advances** in computability theory.
- To learn fundamental proving techniques like **induction** and **diagonalisation**.

Literature

- Martin: *Introduction to Languages and the Theory of Computation*. 3rd Edition, McGraw Hill, 2003.
  - Criticized in Amazon Reviews. But several editions.
- John E. Hopcroft, R. Motwani and J. Ullman: *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 3rd Ed, 2007.
  - Excellent book, mainly on automata theory context free grammars.
  - But covers Turing machines, decidability questions as well.

Literature

- Velleman: *How To Prove It*. Cambridge University Press, 2nd Edition, 2006.
  - Book on basic mathematics.
  - Useful if you need to fresh up your mathematical knowledge.
- Griffor (Ed.): *Handbook of Computability Theory*. North Holland, 1999.
  - Expensive. Postgraduate level.