The Topic of Computability Theory

A **computable function** is a function

\[ f : A \rightarrow B \]

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 computability theory we explore the *limits* of the notion of computability.
Examples

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

where $\mathbb{N} = \{0, 1, 2, \ldots \}$.

$\exp$ is computable.

However, can we really compute $\exp(1000000000000000000000000000000000)$?

Let $\text{String}$ be the set of strings of ASCII symbols.

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

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

Is $\text{check}$ computable or not?
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.
Define a function $\text{terminate} : \text{String} \rightarrow \{0, 1\}$,

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

Is $\text{terminate}$ computable?
Define a function $\text{issortingfun} : \text{String} \rightarrow \{0, 1\},$

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

Is $\text{issortingfun}$ computable?
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 \( \text{issortingfun}(q(p)) = 1 = \text{terminate}(p) \).
- If \( p \) does not terminate, then \( q(p) \) does not terminate on any input, so \( \text{issortingfun}(q(p)) = 0 = \text{terminate}(p) \).
- Our program returns now \( \text{issortingfun}(q(p)) \) which is the result of \( \text{terminate}(p) \).

So we have obtained by using a program for \( \text{issortingfun} \) a program which computes \( \text{terminate} \).

But \( \text{terminate} \) is non-computable, therefore \( \text{issortingfun} \) cannot be computable.
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*. 
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.
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 $P_1$ be reduced to deciding another problem $P_2$? (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.
Hilbert (1928)

Poses the “Entscheidungsproblem” (German for decision problem).
Entscheidungsproblem

- Hilbert refers to a theory developed by him for formalising mathematical proofs.
- Assumes that it is complete and sound, i.e. that it shows exactly all “true” formulae.
- Hilbert asks, whether there is an algorithm, which decides whether a mathematical formula is a consequence of his theory.
- Assuming his theory is complete and sound, such an algorithm would decide the truth of all mathematical formulae.
- So we can say that Hilbert asked, whether there is an algorithm which decides the truth of all mathematical formulae.
The question, whether there is an algorithm for deciding the truth of mathematical formulae is later called the “Entscheidungsproblem” (or “decision problem”).

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.
Kurt Gödel (1906 – 1978)
Introduced the recursive functions in his 1933 - 34 Princeton lectures.
History of Computability Theory

Probably the most influential computability theoretist up to now. Introduced the partial recursive functions.
History of Computability Theory

Emil Post
(1897 – 1954)
Introduced the Post problems.
History of Computability Theory

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

- Gödel (1931) proves in his first incompleteness theorem:
  - Every reasonable primitive-recursive theory (the theorem generalises to recursive i.e. computable theories) is incomplete, i.e. there is a formula s.t. neither the formula nor its negation is provable.
  - Therefore no such theory proves all true formulae.

- Recursive functions will be later shown to be the computable functions.

- Once this is established, it follows that the “Entscheidungsproblem” is unsolvable – an algorithm for deciding the truth of mathematical formulae would give rise to a complete and sound theory fulfilling Gödel’s conditions.
History of Computability Theory

Church, Turing (1936) postulate that the models of computation established above define exactly the set of all computable functions (Church-Turing thesis).

Both established undecidable problems and concluded that the Entscheidungsproblem is unsolvable even for a class of very simple formulae.

Church shows the undecidability of equality in the \( \lambda \)-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)
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?
Degrees

Reducible to

Degree

computable problems
History of Computability Theory

Yuri Vladimirovich Matiyasevich (∗ 1947)

Solves 1970 Hilbert’s 10th problem negatively: The solvability of Diophantine equations is undecidable.
Stephen Cook (Toronto)

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

- 80% Exam.
- 20% Coursework.
Course Home Page

- Located at
  [http://www.cs.swan.ac.uk/~csetzer/lectures/computability/06/index.html](http://www.cs.swan.ac.uk/~csetzer/lectures/computability/06/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 plagiarism, coursework and solutions to coursework will **not** be made available in electronic form (e.g. on this web site).
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.
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

  Main text book.

- Thomas A. Sudkamp: *Languages and machines*. Addison-Wesley 2006.


Literature

  - Criticized in Amazon Reviews. But several editions.

  - Excellent book, mainly on automata theory context-free grammars.
  - But covers Turing machines, decidability questions as well.
Literature

  - Book on basic mathematics.
  - Useful if you need to fresh up your mathematical knowledge.

  - Expensive. Postgraduate level.