

GEM 1501 Problem Solving With Computers

Lecture 8:

Noncomputability and Undecidability

Frank Stephan

Summary of Previous Lecture

- Growth of functions
- NP-complete problems
- Is P equal NP ?
- Even worse problems

Run-Time of Algorithms

- Let n be the length of the input (number of bits or number of symbols);
- Let $T(n)$ be the time needed by the algorithm for the most difficult input consisting of up to n symbols;
- Two major types of growth-rate for T :
Polynomial: $T(n) \in O(n^c)$ for some constant c ;
Exponential: $T(n) \in O(2^{n^c})$ for some constant c .
- A set A is in P iff there is an algorithm and a polynomial p such that for every n and every $x \in \{0, 1\}^n$, the algorithm needs at most time $p(n)$ and determines whether $x \in A$.
- A set A is in EXPTIME iff there is an algorithm and a polynomial p such that for every n and every $x \in \{0, 1\}^n$, the algorithm needs at most time $2^{p(n)}$ and determines whether $x \in A$.

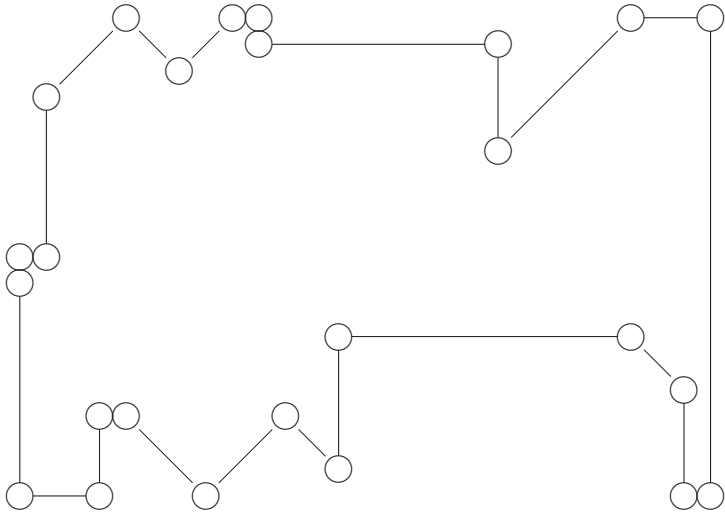
Problems and Solutions

- A decision problem is a set A of instances x such that $x \in A$ iff x has a solution y .
- Example: Monkey-Puzzles would be the set of all $(m, k, a_1, \dots, a_{m \cdot k})$ such that the tiles $a_1, \dots, a_{m \cdot k}$ can be arranged in an $m \times k$ array.
- A problem A is in P iff some algorithm can compute in polynomial time for every x whether $x \in A$ or $x \notin A$, that is, iff some algorithm can find out in polynomial time whether an instance x has a solution or not.
- A problem A is in NP iff some algorithm can verify in polynomial time whether a given solution y for a given instance x is correct; size of solutions should be bounded by a polynomial in size of x .
- A problem A is NP-complete iff A is in NP and for every further problem B in NP there is a polynomial time computable function F with $x \in B \Leftrightarrow F(x) \in A$ for all x .

Example of an NP Problem

A salesman wants to visit from his hometown 24 other places for doing business and then return home.

He does not visit any place twice.



Formulation as
NP Decision Problem

Given the cities
and a constant c ,
is there a tour
of length up to c ?

See Assignment 15.

Problems in P and NP

- In P: Quadratic Equation in one variable.
- NP-complete: Multivariate Quadratic Equations. Number of variables varies from instance to instance.
- In P: 2SAT. Resolution runs in polynomial time.
- NP-complete: 3SAT, 4SAT, SAT in general. Here, SAT is the set of all formulas ψ (in conjunctive normal form) which be made true by suitable choice of variables.
- NP-complete: Travelling Salesman Problem.
- In P: Set of primes; given n and n -digit binary number x , is x prime? Answer can be found in time polynomial in n .
- In NP: Graph-Isomorphism. Perhaps even in P.
- Beyond NP: Winning strategy for Checkers; Truth of additive formulas over integers (Presburger arithmetic).

NP-complete Problems Stand and Fall Together

- All NP-complete problems are equivalent!
- One NP-complete problem is in $P \Leftrightarrow$ all NP-problems are in P .
- One NP-problem needs exponential time \Leftrightarrow all NP-complete problems need exponential time.
- Reason for equivalence: For every NP-problem A and every NP-complete problem B there is a polynomial time computable function F with $x \in A \Leftrightarrow F(x) \in B$. In other words: every NP-problem can be translated in every NP-complete problem.
- NP-completeness indicates difficulty of problem, but some algorithms have good performance on some NP-complete problems (like finding shortest roundtour through all cities and villages of Sweden).

Hierarchy

- P (Polynomial Time)
- NP (Nondeterministic Polynomial Time)
- PSPACE (Polynomial Space)
- EXPTIME (Exponential Time)
- Double Exponential Time

Is $P \subset NP$ or $P = NP$? In everyday language: Is solving problems more difficult than checking solutions?

The Undecidable

- **Certain things are not only slow but just impossible**
- Unbounded Puzzles, Post's Correspondence Problem
- The Halting-Problem
- Recursively Enumerable Sets
- Rice's Theorem

Question

- Given a decision problem with an infinite set of inputs.
- Is there an algorithm (no matter how inefficient) that computes the correct answer for every input?
- Example Satisfiability: The set of inputs is the set of all finite sets of disjunctions. The decision problem is the question whether a given set of disjunctions of perhaps negated Boolean variables has a solution.

Set Theory

- A problem is a function from all texts (describing problem instances) to {Yes, No}.
- There are only countably many algorithms.
- There are **more than** countably many functions from texts to {Yes, No}. That is, there are **uncountably** many problems.
- Since there are **more problems than algorithms**, some problems do not have an algorithm.
- The details of the proof need knowledge on set-theory and does not give much insight how undecidable problems look like.
- Topic of this lecture is more **concrete knowledge** about the Undecidable.

The Undecidable

- Certain things are not only slow but just impossible
- **Unbounded Puzzles, Post's Correspondence Problem**
- The Halting-Problem
- Recursively Enumerable Sets
- Rice's Theorem

Monkey Puzzles Revisited

Given some types of tiles, here an example

12	12	23	23	34	34	45	45	56	56	67	67	78	78	89	89
11	12	11	23	11	34	11	45	11	56	11	67	11	78	11	89

Using arbitrary many tiles of these types and given a finite $m \times n$ area, can the tiles be arranged such that the area is covered?

12	23	34	45	56		12	23	34	45	56	67	78	89	??
12	23	34	45	56		12	23	34	45	56	67	78	89	??

12	23	34	45	56		12	23	34	45	56	67	78	89	??
12	23	34	45	56		12	23	34	45	56	67	78	89	??

12	23	34	45	56		12	23	34	45	56	67	78	89	??
11	11	11	11	11		11	11	11	11	11	11	11	11	??

A 3×5 area can be covered but a 3×9 area cannot be covered.

Is there a general algorithm solving this problem?

The general problem

- Consider the tiles from Monkey Puzzles.
- Given a set of tile types.
- Can you use tiles of this type to cover every finite $m * n$ area for all m, n ?

Example (continued)

- Is there a function in a given programming language, say JavaScript, to which you can pass a description of the tile types and which returns `true` if tiles of these types can cover every finite area, `false` if tiles of these types cannot cover some finite area.

- `function covering(tilearray)`

```
    { .... }
```

```
    var tiles = new Array();
```

```
    tiles.push(new Tile(1,2,2,3)); tiles.push(new Tile(1,3,3,5));
```

```
    tiles.push(new Tile(1,2,2,4)); tiles.push(new Tile(1,3,3,6));
```

```
    if (covering(tiles))
```

```
        { document.write("Tiles can cover every area."); }
```

```
    else
```

```
        { document.write("Tiles cannot cover every area."); }
```

Answer: NO!

- There is no Java Script program that does this job!
- There is no program in any language that does this job!
- One can prove that there is no program in any language on any computer of any speed in any amount of time that does this job!

The Undecidable

- A problem is called “undecidable” if there is no algorithm and thus no computer program which solves a given problem.
- Note: Intractable problems have bad algorithms, undecidable problems have none!
- There are many undecidable problems in mathematics and computer science.

Post's Word Correspondence Problem

- Given two rows of words with the same number of words.
- Is there a sequence (of any length) of indices such taken the words in both rows in the sequence of indices spell out the same word?
- Undecidable!

Example

- | (1) | (2) | (3) | (4) | (5) |
|------|-----|-----|------|-----|
| abb | a | bab | baba | aba |
| bbab | aa | ab | aa | a |

- Admits correspondence: 2,1,1,4,1,5:

a abb abb baba abb aba

aa bbab bbab aa bbab a

- | (1) | (2) | (3) | (4) | (5) |
|-----|-----|-----|------|-----|
| a | aa | aba | aabb | abb |
| b | ba | bba | bbba | bba |

- Does not admit any correspondence: Word made from upper line starts with a, word made from lower line starts with b.

The Undecidable

- Certain things are not only slow but just impossible
- Unbounded Puzzles, Post's Correspondence Problem
- **The Halting-Problem**
- Recursively Enumerable Sets
- Rice's Theorem

Impossibility of Program Verification

- A specification for a problem contains the set of legal inputs and outputs and the relation between them.
- Program verification: Given a specification S and a program P , is $P(I)$ terminating with output O whenever required to do so by S ?
- Is it possible to solve the verification problem by an algorithm?
- Answer: NO!
- The most famous problem from programme verification:
Given text *prgtext* of program and input x , **does the program *prgtext* with input x terminate?**
- This problem is called **The Halting Problem**.

Assume that Halting Problem is decidable ...

- Assume there is such a program:

```
function halting(prgtext,input)
  { var ishalting;
    ..... // some computations
    return(ishalting); }
```

- There is a Java Script Interpreter:

```
function simulate(prgtext,input)
  { var result;
    ..... // some computations
    return(result); }
```

- What about this function:

```
function diagonal(prgtext)
  { if ishalting(prgtext,prgtext)
    { if (simulate(prgtext,prgtext)==0)
      { return(1); } }
    return(0); }
```

... then something bad happens

- Let `diagtext` have as value the text of the function diagonal plus the subfunctions of last page.
- What output produces "`diagonal(diagtext)`"?
- The function "`diagonal`" is total because the function "`halting`" is used to test whether a simulation will terminate before the simulation is run.
- The function "`diagonal`", on input "`diagtext`", outputs 1 if the simulation says it would output 0 and outputs 0 otherwise.
- Hence either the function "`halting`" or the function "`simulate`" is false.
- Everyday experience tells that simulation is possible, thus the halting problem is undecidable. So the function "`halting`" does not exist.
- This proof-method is called "`diagonalization`": If one makes a matrix of all program-texts and all inputs, then the inputs for the function `diagonal` correspond to the diagonal of the matrix.

Halting Problem and Diagonal

```
Let texta = "function a(x) { return(0); }"
textb = "function b(x) { return(x.length); }"
textc = "function c(x) { while (x!='function a(x) { return(0); }') { }
        return(1); }"
textd = "function d(x) { return(1); }"
```

Possible Inputs	texta	textb	textc	textd	diagtext
Function-texts					
texta	0<--	0	0	0	0
textb	28	35<--	72	28	2048
textc	1	undef	undef<--	undef	undef
textd	1	1	1	1<--	1
diagtext	1	0	0	0	???

Origin of Diagonalization Proofs

- 1873: Georg Cantor invented Diagonalization for Set Theory.
- **Theorem:** There are more problems than algorithms.
- **Algorithm:** Text in any programming language.
- **Problem:** Set of texts.
- **Proof:** Consider function F assigning to every text a problem. For every P , $F(P)$ is a set of texts. For example, F might assign to program P the problem $F(P)$ solved by P .
- **New diagonal problem D** defined as follows.
 $T \in D$ iff $T \notin F(T)$.
- **D differs from $F(P)$ for all P .**
 $P \in F(P) \Rightarrow P \notin D$;
 $P \notin F(P) \Rightarrow P \in D$.
- Thus there is no function F from all texts to problems which covers all problems. There are **more problems than texts**.

Can nontermination be avoided?

- Assume some programming language XYZ does not permit to write programs which never halt.
- There is of course a function `simulatexyz` to simulate XYZ.
- Now a diagonal function can be made which cannot be written down in the programming language XYZ.

```
function diagonalxyz(prgtext)
  { return(1+simulatexyz(prgtext,prgtext)); }
```

This function cannot be computed by any function written down in the language XYZ.

- If the function would have the text `diagxyztext` in the language XYZ, then there would be a contradiction when analyzing the behaviour of the expression `diagonalxyz(diagxyztext)`.
- Removing all non-terminating programs also removes some terminating programs.

Alan Turing

- One of the Founders of the Theory of Computing.
- He said: “If a machine is expected to be infallible, it cannot also be intelligent.”
- That is, if a programming language does not permit to write infinite loops then it does not permit to program all total computable functions.
- Browsers ask users whether to continue slow programs but they do not figure out whether those programs are incorrect.

The Undecidable

- Certain things are not only slow but just impossible
- Unbounded Puzzles, Post's Correspondence Problem
- The Halting-Problem
- **Recursively Enumerable Sets**
- Rice's Theorem

Recursive Functions and R.E. Sets

- Successor: $S(x) = x + 1$ is primitive function;
- Addition: $x + 0 = x$; $x + S(y) = S(x + y)$;
- Multiplication: $x \cdot 0 = 0$; $x \cdot S(y) = (x \cdot y) + x$;
- Exponentiation: $x^0 = 1$; $x^{S(y)} = x^y \cdot x$.
- All mathematical functions which can be written down by an algorithm can also be generated - directly or indirectly - by recursion and unbounded search. Hence these functions are also called “recursive functions” or “computable functions” .
- A nonempty set A is recursively enumerable (r.e.) iff it is the range of a recursive function. The empty set is by definition recursively enumerable as well.

Another Famous Undecidable Problem

- David Hilbert, 10th problem 1900

Determination of the Solvability of a Diophantine Equation. Given a Diophantine equation with any unknown number of quantities and with rational integral numerical coefficients: To devise an algorithm according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.

- What does this mean?

A polynomial with integer coefficients and n variables is a finite sum of products of variables and integers, for example $D(x_1, x_2, x_3) = 2x_1x_2^4 - 17x_3^8$. Hilbert was interested in having an algorithm which given a polynomial D checked whether there are x_1, x_2, \dots, x_n such that $D(x_1, x_2, \dots, x_n) = 0$.

- Notation: A set A is Diophantine iff there is a polynomial D as above such that $x \in A \leftrightarrow \exists y_2, \dots, y_n (D(x, y_2, \dots, y_n) = 0)$.

History of Hilbert's 10th Problem

- 200-284: Diophantus from Alexandria (Egypt) wrote books about problems known as Diophantine equations.
- 1900: David Hilbert poses his 10th problem asking for an algorithm to check the solvability of Diophantine equations.
- 1953: Martin Davis conjectures that a set of natural numbers is given by a Diophantine equation iff it is recursively enumerable.
- 1970: Yuri Matiyasevich proved Davis' conjecture. Thus Hilbert's 10th problem has no solution.

How many Variables

- One variable is solvable. Example:

$$P(x) = 16 + 20x + 10x^2 - x^4.$$

$|x| > 47 \Rightarrow P(x) < 0$. Thus finitely many tests sufficient.

Solution at $x = 4$.

- Ten variables unsolvable.

Complicated proof.

- More: <http://logic.pdmi.ras.ru/Hilbert10/>

Decidability

Assume that A and the complement of A are both recursively enumerable. Then A is decidable.

- Let A be the range of F : $A = \{F(0), F(1), \dots\}$.
- The complement of A is the range of G :
- Every x is either in A or in the complement of A .
- On input x find first y such that either $F(y) = x$ or $G(y) = x$.
- If $x = F(y)$ then x is in A .
- If $x = G(y)$ then x is not in A .
- A is decidable.

Example

- Assume that A and its complement are recursively enumerable.
- Enumeration F : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ...;
Enumeration G : 0, 1, 4, 6, 8, 9, 10, 12, 14, 15, 16, ...;
Input x : 8.
- Search in F and G alternately until x is found.

Input x	y	$F(y)$	$G(y)$	
8	0	2	0	
	1	3	1	
	2	5	4	
	3	7	6	
	4	11	8	====> Found, 8 = G(4)
				8 is not in set A.

- So A is decidable.

The Halting Problem is R.E.

- Let A be a program terminating for all inputs I .
- $F(P, I, t) = (P, I)$ if $P(I)$ terminates in time t .
 $F(P, I, t) = (A, I)$ if $P(I)$ does not terminate in time t .
- If $P(I)$ terminates then it terminates in some time t and $F(P, I, t) = (P, I)$ for some t .
If $P(I)$ does not terminate then $F(P, I, t) = (A, I)$ for all t .
- F is an enumeration of the halting problem.

The Halting Problem is Complete

- Assume that A is recursively enumerable.
- There is a computable and total function F such that $A = \{F(0), F(1), \dots\}$.
- For x , P_x searches for y with $F(y) = x$. Here P_{4096} :

```
{ var y = 0; while (4096 != F(y)) { y = y+1; } }
```
- Translation x to P_x ; $x \in A \Leftrightarrow P_x(0)$ halts.
- Every r.e. set can be reduced to the halting problem this way.

An r.e. set which is not complete

- Are all natural r.e. sets either decidable or complete?
- Answer: Most are but not all.
- Example of an incomplete one given next two slides.
- Emil Post started 1944 investigation of incomplete sets, study continues until today.

Set of Compressible Texts

- Let C be the set of all compressible texts: $T \in C$ iff there is a Java Script program P shorter than T such that P does not need any input and outputs T .
- C is recursively enumerable.
- C is not complete, that is, the halting problem cannot be translated into C .
- Although C is not complete, it is still very hard. In particular, C is undecidable.
- Since the 1960ies, Gregory Chaitin, Andrei Kolmogorov and Ray Solomonoff investigated the topic of compressibility of texts and numbers.

The Undecidable

- Certain things are not only slow but just impossible
- Unbounded Puzzles, Post's Correspondence Problem
- The Halting-Problem
- Recursively Enumerable Sets
- **Rice's Theorem**

Program Behaviour

- Many basic properties are undecidable.
- Examples:
 - Do programs terminate?
 - Do given program lines get executed?
 - Will given programs encounter a certain error like dividing by zero?
- Rice's Theorem provides many examples.

Rice's Theorem

Rice Theorem says roughly:

It is impossible to check semantic properties of a program P .

What are syntactic and semantic properties?

- Syntactic: P contains two for-loops and one while-loop.
 P is 213480 bytes long.
 P is written in Java Script.
- Run-Time: The computation of $P(56)$ needs 242998324552 steps.
The computation of $P(n)$ needs $O(n^3)$ steps.
- Semantic: $P(0)$ halts and takes the value 5.
 P does not halt (= is undefined) on every input.

Rice's Theorem Formally

- Given V, W such that
 - V, W are two nonempty sets of programs.
 - Every program is in one but not both sets.
 - If P, Q have the same semantic behaviour then they are either both in V or both in W .
- Then one can translate the halting problem into V or into W .
- In particular, V and W are both undecidable.
- Example.
 $V = \{P : P(5) \text{ is undefined or even}\}; W = \{P : P(5) \text{ is odd}\}.$
Then V, W form a nontrivial splitting of all programs and there is a computable function F such that $F(\text{prgtext}, x) \in W$ iff prgtext is the text of a program which halts on input x .

Classes Harder Than Halting

Rice's Theorem permits to give numerous examples of undecidable problems.

- Problem whether a program halts on some input.
- Problem whether a program outputs the value 37 on some input.
- Problem whether a program halts at 0 and does not halt at 1.
- Problem whether a program always halts.
- Problem whether a program computes a function undefined at infinitely many places.

Last three classes are harder than halting. The last two problems are **highly undecidable** in the sense that even an “oracle” to decide the halting problem would not give enough information to decide these two problems.

Four levels of algorithmic behaviour

- **Highly undecidable problems**

Even if you would know the halting problem, you could not solve these problems.

Example: Does a given program halt on all inputs?

- **Undecidable problems**

No algorithm solves such problems, but some of these problems are still recursively enumerable.

Halting Problem and Post's Correspondence Problem.

- **Intractable problems**

Problems which can be solved in principle but it might take millions of millions years to wait for the program to terminate.

Examples: Winning Strategy for Checkers, Presburger Arithmetic.

- **Tractable problems**

Things a computer can solve in everyday life. Often identified with the class P of all Polynomial Time Computable Problems.

Examples: 2SAT, Primality-Test, Spanning Tree.

The Undecidable

- Certain things are not only slow but just impossible
- Unbounded Puzzles, Post's Correspondence Problem
- The Halting-Problem
- Recursively Enumerable Sets
- Rice's Theorem

Next week: Various computers

- Computers differ in their speed, architecture, design.
- Are all computers equivalent?
- In theory: yes – Given enough time and memory, any computer can “simulate” any other computer.
- In practice: no – Speed and performance of computers varies a lot.
- Restricted Variants of Computing:
 - finite state machines,
 - stack machines.