CHAPTER 3
The Church-Turing Thesis

Contents

The definition of algorithm

- Informally, ***an algorithm is a collection of simple instructions for carrying out some task.***
- Algorithms play an important role in CS and Math.
- Even though algorithms have had a long history in mathematics (finding prime numbers, greatest common divisors, …), the notion of algorithms itself was not defined precisely until the twentieth century.
- Before that, mathematicians had an intuitive notion of what algorithms were and relied upon that notion when using and describing them.
- The intuitive notion was insufficient for gaining a deeper understanding of algorithms.
- The story "Hilbert's tenth problem" relates how the precise definition of algorithm was crucial to one important mathematical problem.
- In 1900, mathematician David Hilbert, in his lecture (at the International Congress of Mathematicians in Paris), identified twenty-three mathematical problems and posed them as challenge for the coming century.
- The tenth problem on his list concerned algorithms.

  <u>Devise</u> a process according to which it can be determined by finite number of operations whether a polynomial has an integral root.

Hilbert's tenth problem

- A ***polynomial*** is a sum of terns, where each ***term*** is a product of certain variables and a constant called a ***coefficient.***
- $6 \cdot x \cdot x \cdot x \cdot y \cdot z \cdot z = 6x^3yz^2$ is a term with coefficient 6.
- $6x^3yz^2 + 3xy^2 - x^3 - 10$ is a polynomial with four terms over the variables *x,y,* and *z.*
- A ***root*** of a polynomial is an assignment of values to variables so that the value of the polynomial is 0. That polynomial has a root *x=5, y=3, z=0.*
- This root is ***integral*** since all the variables are assigned integer values.
- Some polynomials have an integral root and some do not.
- So, Hilbert's tenth problem was to devise an algorithm that tests whether a polynomial has an integral root.
- We now know that no algorithm exists for this task; *it is algorithmically unsolvable.*
- For mathematicians of that period to come to this conclusion with their intuitive concept of algorithm would have been virtually impossible.
- The intuitive concept of algorithm may have been adequate for giving algorithms for certain tasks, but it was useless for showing that no algorithm exists for a particular task.
- Proving that an algorithm does not exist requires having a clear definition of algorithm. Progress on the tenth problem had to wait for that definition.

Church-Turing thesis

- The definition came in the 1936 papers of ***A. Church*** and ***A.Turing***.
- Church used a notational system called λ-calculus to define algorithms.
- Turing did it with his 'machines'.
- These two definitions were shown to be equivalent.
- This connection between the informal notion of algorithm and the precise definition has come to be called the ***Church-Turing thesis***.

Intuitive notion of algorithms	equals	Turing machine algorithms

- This thesis provides the definition of algorithm necessary to resolve Hilbert's tenth problem.
- In 1970, Yuri Matijasevich showed that no algorithm exists for testing whether a polynomial has integral roots.
- Later we will see the techniques that form the basis for proving that this and other problems are algorithmically unsolvable.

Hilbert's tenth problem(cont.)

- We formulate Hilbert's tenth problem in our terminology.
- Let *D* = {*p*: *p* is polynomial with an integral root}. Hilbert's tenth problem asks whether the language (set) *D* is decidable.
- The answer is negative. We can show that *D* is Turing-recognizable, but not decidable.
- Let first consider a simpler problem: it is an analog of Hilbert's tenth problem for polynomials that have only a single variable, e.g. $4x^3 - 2x^2 + x - 7.$
- *Let D1* = {*p*: *p* is polynomial over *x* with an integral root}. Here is a Turing machine *M1* that recognizes *D1*:

  M1= " the input is a polynomial *p* over the variable *x*.

  1. Evaluate *p* with *x* set successively to the values 0,1,-1,2,-2,… . If at any point the polynomial evaluates to 0, *accept*."

- Clearly, if *p* has an integral root, *M1* will find it and accept. If *p* does not have an integral root, *M1* will run forever.
- For multivariable case, we can present similar Turing machine *M* that recognizes *D*. *M* will go through all possible settings of its variables to integral values.
- Both *M1* and *M* are recognizers but not deciders. We can convert *M1* to be decider for *D1* since we can calculate bounds within which the roots of a single variable polynomial must lie and restrict the search to these bounds. If a root is not found within these bounds, the machine *rejects.* $x_0 \in [-kc_{\max} / |c_1|, kc_{\max} / |c_1|]$
- Matijasevich's theorem shows that calculating such bounds for multivariable polynomials is impossible.

Terminology for describing Turing Machines

Three variants of description.

1. The **formal description**: spells out in full the Turing machine's states, transition function, and so on.
2. The **implementation description**: uses English prose to describe the way that the Turing machine moves its head and the way that it stores data on its tape.
3. The **high-level description**: uses English prose to describe an algorithm, ignoring the implementation model. At this level we do not need to mention how the machine manages its tape or head.

- From now on we will use only high-level descriptions.
- The input to a TM is always a string.

  if we want to provide an object other than a string as input, we must first represent that object as a string. Strings can easily represent polynomials, graphs, grammars, automata, and any combination of those objects.

- Notation for the encoding of an object *O* into its representation as a string is <*O*>. A string <*O1,O2,…,Ok*> is the encoding of several objects *O1,O2,…,Ok*.
- We will use the following format for describing TM algorithms:
  - We describe TM algorithm with an indented segments of text within quotes.
  - We break the algorithm into stages, each usually involving many individual steps of the TM's computation.
  - The first line of the algorithm describes the input to the machine. If the input is simply *w*, the input is taken to be a string *w*. If the input is the encoding of an object as in <*A*>, the TM first implicitly tests whether the input properly encodes an object of the desired form and rejects it if it doesn't.

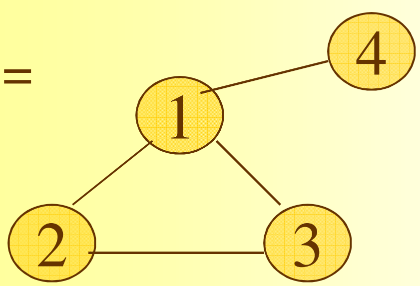
Example

- Let *A* be the language consisting of all strings representing undirected graphs that are connected.
- Graph is connected if every node can be reached from every other node by traveling along the edges of the graph.
- We write *A* = {<*G*>*: G* is a connected undirected graph}.
- The following is a high-level description of a TM *M* that decides *A*.

M = "On input <*G*>, the encoding of a graph *G*:

1. Select the first node of *G* and mark it.
2. Repeat the following stage until no new nodes are marked.
3. For each node in *G*, mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of *G* to determine whether they all are marked. If they are, *accept*; otherwise *reject*."

G =



<*G* >= (1,2,3,4)((1,2), (2,3), (3,1), (1,4))

A graph *G* and its encoding <*G*>.