CS301: Computability and Complexity Theory (CC)

Lecture 5: Decidability

Dumitru Bogdan

Faculty of Computer Science
University of Bucharest

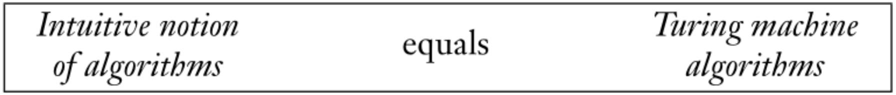
November 3, 2023

Table of contents

Section 1

Previously on CS301

The Definition of Algorithm



Intuitive notion of algorithms equals *Turing machine algorithms*

Figure: The Church–Turing thesis

Decidable languages

- We gave some examples of languages that are decidable by algorithms
- We focused on languages concerning automata and grammars
- We focused on those languages because certain problems of this kind are related to specific applications

A_{DFA} is decidable

Bellow language contains the encodings of all DFAs together with strings that the DFAs accept

$$A_{DFA} = \{< B, w > | B \text{ is a DFA that accepts input string } w\}$$

Theorem

A_{DFA} is decidable

Proof idea: All we need is to present a TM M that decides A_{DFA} $M =$ On input $< B, w >$, where B is a DFA and w is a string:

1. Simulate B on w
2. If simulation ends in accept state, *accept*. If ends in nonaccepting state, *reject*

A_{NFA} is decidable

Bellow language contains the encodings of all NFAs together with strings that the NFAs accept

$$A_{NFA} = \{< B, w > | B \textit{ is a NFA that accepts input string } w\}$$

Theorem

A_{NFA} is decidable

A_{NFA} is decidable

Proof.

We present a TM N that decides A_{NFA} . We could design N to operate like M , simulating an NFA instead of a DFA. Instead, we'll do it differently to illustrate a new idea: Have N use M as a subroutine. Because M is designed to work with DFAs, N first converts the NFA it receives as input to a DFA before passing it to M.

$N =$ On input $< B, w >$, where B is a NFA and w is a string:

1. Convert NFA B to an equivalent DFA C
2. Run TM M from previous theorem on input $< C, w >$
3. If M accepts, *accept*; otherwise, *reject*

Running TM M in stage 2 means incorporating M into the design of N as a subprocedure □

A_{REX} is decidable

Bellow language contains the encodings of all REXs together with strings that the REXs generates

$$A_{REX} = \{< R, w > | R \textit{ is a regular expression that generates string } w\}$$

Theorem

A_{REX} is decidable

A_{REX} is decidable

Proof.

The following TM P decides A_{REX}

$P =$ On input $< R, w >$, where R is a regular expression and w is a string

1. Convert regular expression R to an equivalent NFA A
2. Run TM N on input $< A, w >$
3. If N accepts, *accept*; if N rejects, *reject*

$\square$

Section 2

Context setup

Context setup

Corresponding to Sipser 4.1 & 4.2

Context setup

- Previously we introduced the TM as a model of a general purpose computer
- We have defined the notion of algorithm in terms of TM by means of the Church–Turing thesis
- We begin to investigate the power of algorithms to solve problems
- We demonstrate certain problems that can be solved algorithmically and others that cannot

Section 3

Decidability (cont)

Context-free language are decidable

Next, we consider the problem of determining whether two context-free grammars generate the same language. Let

$$EQ_{CFG} = \{< G, H > \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$$

We used the decision procedure for E_{DFA} to prove that EQ_{DFA} is decidable. Because E_{CFG} also is decidable, you might think that we can use a similar strategy to prove that EQ_{CFG} is decidable. But something is wrong with this idea. The class of context-free languages is not closed under complementation or intersection. EQ_{CFG} is not decidable.

Context-free language are decidable

Theorem

Every context-free language is decidable

Proof idea: Let A be a CFL. Our objective is to show that A is decidable. One (bad) idea is to convert a PDA for A directly into a TM. That isn't hard to do because simulating a stack with the TM's more versatile tape is easy. The PDA for A may be nondeterministic, but that seems okay because we can convert it into a nondeterministic TM and we know that any nondeterministic TM can be converted into an equivalent deterministic TM. Yet, there is a difficulty. Some branches of the PDA's computation may go on forever, reading and writing the stack without ever halting. The simulating TM then would also have some nonhalting branches in its computation, and so the TM would not be a decider. A different idea is necessary. Instead, we prove this theorem with the TM S that we designed in previously to decide A_{CFG}.

Context-free language are decidable

Proof.

Let G be a CFG for A and design a TM M_G that decides A. We build a copy of G into M_G. It works as follows

$M_G =$ On input w:

1. Run TM S on input $< G, w >$
2. If this machine accepts, *accept*; otherwise, *reject*

□

The relationship among classes of languages

This theorem provides the final link in the relationship among the four main classes of languages that we have described so far: regular, context-free, decidable, and Turing-recognizable

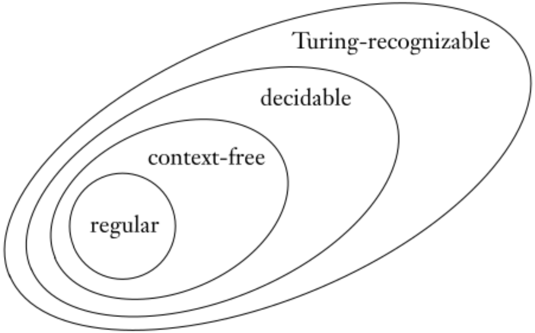


Figure: The relationship among classes of languages

Section 4

Undecidability

Undecidability

- Next, we prove one of the most philosophically important theorems of the theory of computation: There is a specific problem that is algorithmically unsolvable
- Computers appear to be so powerful that you may believe that all problems will eventually yield to them. The theorem presented next demonstrates that computers are limited in a fundamental way
- Ordinary problems that people want to solve turn out to be computationally unsolvable
- We aim to help you develop a feeling for the types of problems that are unsolvable and to learn techniques for proving unsolvability
- First problem we study is to determine whether a TM accepts a given input string

A_{TM} is undecidable

Let

$$A_{TM} = \{< M, w > \mid M \textit{ is a TM and } M \textit{ accepts } w\}$$

Theorem

A_{TM} is undecidable

We first observe that A_{TM} is Turing-recognizable. This theorem shows that recognizers are more powerful than deciders. Requiring a TM to halt on all inputs restricts the kinds of languages that it can recognize. The following Turing machine U recognizes A_{TM}

$U =$ On input $< M, w >$ where M is a TM and w is a string:

1. Simulate M on input w
2. If M ever enters its accept state, *accept*; if M ever enters its reject state, *reject*

A_{TM} is undecidable

Observe that this machine loops on input $< M, w >$ if M loops on w, which is why this machine does not decide A_{TM}. If the algorithm had some way to determine that M was not halting on w, it could reject in this case. However, an algorithm has no way to make this determination, as we shall see.

The TM U is interesting in its own right. It is an example of the **Universal Turing Machine** first proposed by Alan Turing in 1936. This machine is called universal because it is capable of simulating any other Turing machine from the description of that machine.

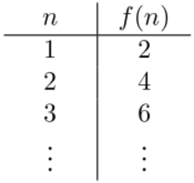
The proof of the undecidability of A_{TM} uses a technique called **diagonalization** 1, discovered by mathematician Georg Cantor in 1873

The Diagonalizaton Method

- Cantor was concerned with the problem of measuring the sizes of infinite sets
- If we have two infinite sets, how can we tell whether one is larger than the other or whether they are of the same size?
- For finite sets, of course, answering these questions is easy. We simply count the elements in a finite set, and the resulting number is its size
- But if we try to count the elements of an infinite set, we will never finish
- For example, take the set of even integers and the set of all strings over 0,1. Both sets are infinite and thus larger than any finite set, but is one of the two larger than the other?
- Cantor proposed a rather nice solution to this problem. He observed that two finite sets have the same size if the elements of one set can be paired with the elements of the other set
- This method compares the sizes without resorting to counting. We can extend this idea to infinite sets

Example

Let $\mathcal{N}$ be the set of natural numbers $\{1, 2, 3, \dots\}$ and let $\mathcal{E}$ be the set of even natural numbers $\{2, 4, 6, \dots\}$. Using Cantor's definition of size, we can see that $\mathcal{N}$ and $\mathcal{E}$ have the same size.The bijection f mapping $\mathcal{N}$ to $\mathcal{E}$ is simply $f(n) = 2n$.



n	$f(n)$
1	2
2	4
3	6
$\vdots$	$\vdots$

This example seems bizarre. Intuitively, $\mathcal{E}$ seems smaller than $\mathcal{N}$ because $\mathcal{E}$ is a proper subset of $\mathcal{N}$. But pairing each member of $\mathcal{N}$ with its own member of $\mathcal{E}$ is possible, so we declare these two sets to be the same size.

The Diagonalizaton Method

Definition

A set A is **countable** if either it is finite or it has the same size as $\mathcal{N}$

Example

Let us study an even stranger example. Let $\mathcal{Q} = \{\frac{m}{n} | m, n \in \mathcal{N}\}$ be the set of positive rational numbers. $\mathcal{Q}$ seems to be much larger than $\mathcal{N}$. Yet, these two sets have the same size according to our definition. We give a correspondence with $\mathcal{N}$ to show that $\mathcal{Q}$ is countable. One easy way to do so is to list all the elements of $\mathcal{Q}$. Then we pair the first element on the list with the number 1 from $\mathcal{N}$, the second element on the list with the number 2 from $\mathcal{N}$, and so on. We must ensure that every member of $\mathcal{Q}$ appears only once on the list.

To get this list, we make an infinite matrix containing all the positive rational numbers. The ith row contains all numbers with numerator i and the jth column has all numbers with denominator j. So the number $\frac{i}{j}$ occurs in the ith row and jth column.

Example

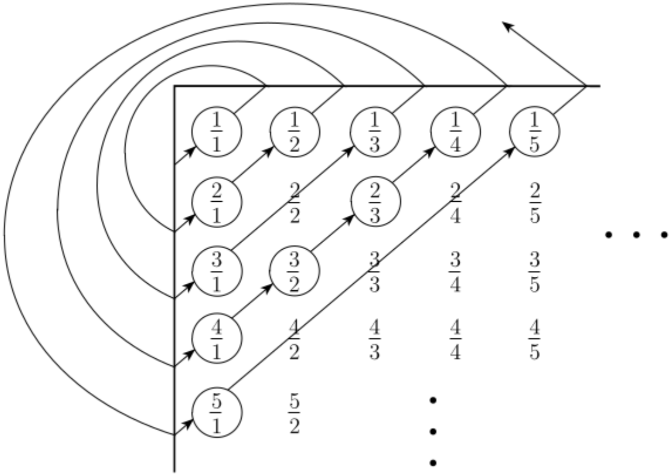


Figure: A correspondence of $\mathcal{N}$ and $\mathcal{Q}$

Example

Now we turn this matrix into a list. One (bad) way to attempt it would be to begin the list with all the elements in the first row. That isn't a good approach because the first row is infinite, so the list would never get to the second row. Instead we list the elements on the diagonals, which are superimposed on the diagram, starting from the corner.

The first diagonal contains the single element $\frac{1}{1}$ and the second diagonal contains the two elements $\frac{2}{1}$ and $\frac{2}{1}$. As result, the first three elements of the list are $\frac{1}{1}$, $\frac{2}{1}$ and $\frac{1}{2}$.

In the third diagonal, a complication arises. It contains $\frac{3}{1}$, $\frac{2}{2}$ and $\frac{1}{3}$. If we simply added these to the list, we would repeat $\frac{1}{1} = \frac{2}{2}$. We avoid doing so by skipping an element when it would cause a repetition. So we add only the two new elements $\frac{3}{1}$ and $\frac{1}{3}$. Continuing in this way, we obtain a list of all the elements of $\mathcal{Q}$

Uncountable sets

After seeing the correspondence of $\mathcal{N}$ and $\mathcal{Q}$, you might think that any two infinite sets can be shown to have the same size. After all, you need only demonstrate a correspondence, and this example shows that surprising correspondences do exist. However, for some infinite sets, no correspondence with $\mathcal{N}$ exists. These sets are simply too big. Such sets are called **uncountable**.

The set of real numbers ($\mathcal{R}$) is an example of an uncountable set. A real number is one that has a decimal representation. The numbers π and $\sqrt{2}$ are examples of real numbers. Cantor proved that $\mathcal{R}$ is uncountable.

$\mathcal{R}$ is uncountable

Theorem

$\mathcal{R}$ is uncountable

Proof.

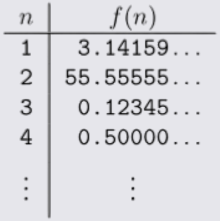
In order to show that R is uncountable, we show that no correspondence exists between $\mathcal{N}$ and $\mathcal{R}$. The proof is by contradiction. Suppose that a correspondence f existed between $\mathcal{N}$ and $\mathcal{R}$. Our job is to show that f fails to work as it should. For it to be a correspondence, f must pair all the members of $\mathcal{N}$ with all the members of $\mathcal{R}$. But we will find an x in $\mathcal{R}$ that is not paired with anything in $\mathcal{N}$, which will be our contradiction.

The way we find this x is by actually constructing it. We choose each digit of x to make x different from one of the real numbers that is paired with an element of $\mathcal{N}$. In the end, we are sure that x is different from any real number that is paired.

$\mathcal{R}$ is uncountable

Proof.

We can illustrate this idea by giving an example. Suppose that the correspondence f exists. Let $f(1) = 3.14159...$, $f(2) = 55.55555...$, $f(3) = \dots$, and so on, just to make up some values for f. Then f pairs the number 1 with $3.14159\dots$, the number 2 with $55.55555\dots$, and so on. The following table shows a few values of a hypothetical correspondence f between $\mathcal{N}$ and $\mathcal{R}$.



n	$f(n)$
1	3.14159...
2	55.55555...
3	0.12345...
4	0.50000...
$\vdots$	$\vdots$

$\mathcal{R}$ is uncountable

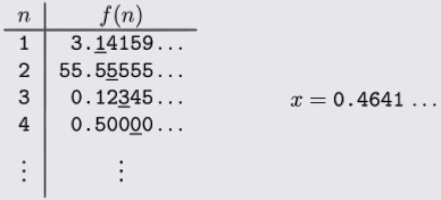
Proof.

We construct the desired x by giving its decimal representation. It is a number between 0 and 1, so all its significant digits are fractional digits following the decimal point. Our objective is to ensure that $x \neq f(n)$ for any n. To ensure that $x \neq f(n)$, we let the first digit of x be anything different from the first fractional digit 1 of $f(1) = 3.14159...$ Arbitrarily, we let it be 4. To ensure that $x \neq f(n)$, we let the second digit of x be anything different from the second fractional digit 5 of $f(2) = 55.555555...$ Arbitrarily, we let it be 6. The third fractional digit of $f(3) = 0.12345...$ is 3, so we let x be anything different say, 4.

$\mathcal{R}$ is uncountable

Proof.

Continuing in this way down the diagonal of the table for f, we obtain all the digits of x, as shown in the following table.



n	$f(n)$
1	$3.\underline{1}4159\ldots$
2	$55.5\underline{5}555\ldots$
3	$0.12\underline{3}45\ldots$
4	$0.500\underline{0}0\ldots$
$\vdots$	$\vdots$

$x = 0.4641\ldots$

We know that x is not $f(n)$ for any n because it differs from $f(n)$ in the nth fractional digit. (A slight problem arises because certain numbers, such as 0.1999... and 0.2000..., are equal even though their decimal representations are different. We avoid this problem by never selecting the digits 0 or 9 when we construct x) $\square$