CS301: Computability and Complexity Theory (CC)

Lecture 6: Decidability

Dumitru Bogdan

Faculty of Computer Science
University of Bucharest

November 10, 2023

Table of contents

Section 1

Previously on CS301

The relationship among classes of languages

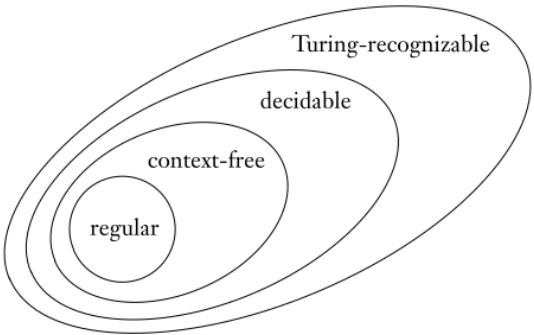

Figure: The relationship among classes of languages

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

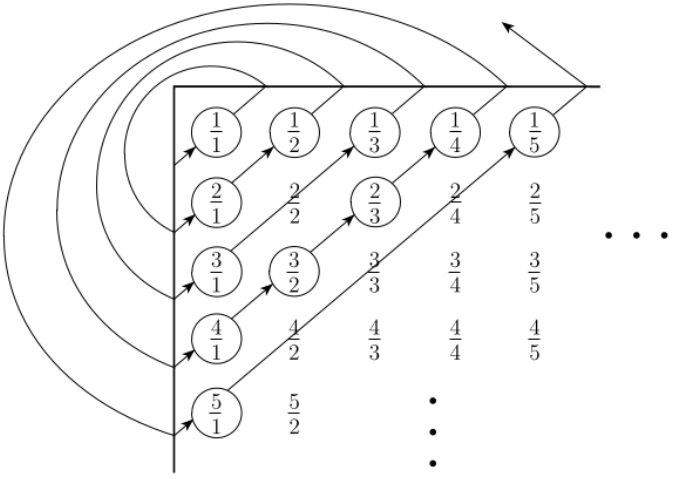

Figure: A correspondence of $\mathcal{N}$ and $\mathcal{Q}$

$\mathcal{R}$ is uncountable

Theorem

$\mathcal{R}$ is uncountable

Proof idea: Suppose that the correspondence f exists. Let $f(1) = 3.14159...$, $f(2) = 55.55555...$, $f(3) = \ldots$, and so on, just to make up some values for f. Then f pairs the number 1 with $3.14159\ldots$, the number 2 with $55.55555\ldots$, 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...
⋮	⋮

Section 2

Context setup

Context setup

Corresponding to Sipser 4.2 & 5.1

Context setup

- We continue to investigate the power of algorithms to solve problems
- We demonstrate certain problems that can be solved algorithmically and others that cannot
- Next, we examine several additional unsolvable problems
- We introduce the primary method for proving that problems are computationally unsolvable: **reducibility**

Section 3

Undecidability (cont)

Some languages are not Turing-recognizable

The preceding theorem has an important application to the theory of computation. It shows that some languages are not decidable or even Turing-recognizable, for the reason that there are uncountably many languages yet only countably many Turing machines. Because each Turing machine can recognize a single language and there are more languages than Turing machines, some languages are not recognized by any Turing machine. Such languages are not Turing-recognizable, as we state in the following theorem:

Theorem

Some languages are not Turing-recognizable

Some languages are not Turing-recognizable

Proof.

To show that the set of all TMs is countable, we first observe that the set of all strings Σ^* is countable for any alphabet Σ. With only finitely many strings of each length, we may form a list of Σ^* by writing down all strings of length 0, length 1, length 2, and so on.

The set of all TMs is countable because each TM M has an encoding into a string $< M >$. If we simply omit those strings that are not legal encodings of TMs, we can obtain a list of all TMs.

To show that the set of all languages is uncountable, we first observe that the set of all infinite binary sequences is uncountable. An **infinite binary sequence** is an unending sequence of 0s and 1s. Let $\mathcal{B}$ be the set of all infinite binary sequences. We can show that $\mathcal{B}$ is uncountable by using a proof by diagonalization similar to the one we used to show that $\mathcal{R}$ is uncountable.

Some languages are not Turing-recognizable

Proof.

Let $\mathcal{L}$ be the set of all languages over alphabet Σ. We show that $\mathcal{L}$ is uncountable by giving a correspondence with $\mathcal{B}$, thus showing that the two sets are the same size.

Let $\Sigma^* = \{s_1, s_2, s_2, \dots\}$. Each language $A \in \mathcal{L}$ has a unique sequence in $\mathcal{B}$. The ith bit of that sequence is a 1 if $s_i \in A$ and is a 0 if $s_i \notin A$, which is called the *characteristic sequence* of A. For example, if A were the language of all strings starting with a 0 over the alphabet $\{0, 1\}$, its characteristic sequence χ_A would be

$$\begin{array}{rccccccccccc} \Sigma^* = \{ & \varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000, & 001, & \cdots & \}\ ; \\ A = \{ & & 0, & & 00, & 01, & & & 000, & 001, & \cdots & \}\ ; \\ \chi_A = & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \cdots & . \end{array}$$

Some languages are not Turing-recognizable

Proof.

The function $f : \mathcal{L} \to \mathcal{B}$, where $f(A)$ equals the characteristic sequence of A, is bijective. Therefore, as $\mathcal{B}$ is uncountable, $\mathcal{L}$ is uncountable as well. Thus we have shown that the set of all languages cannot be put into a correspondence with the set of all TMs. We conclude that some languages are not recognized by any TM. □

A_{TM} is undecidable

Now we have all we need to prove

Theorem

A_{TM} is undecidable

Proof.

We assume that A_{TM} is decidable and obtain a contradiction. Suppose that H is a decider for A_{TM}. On input $< M, w >$, where M is a TM and w is a string, H halts and accepts if M accepts w. Furthermore, H halts and rejects if M fails to accept w. In other words, we assume that H is a TM, where:

$$H(< M, w >) = \begin{cases} \textit{accept if } M \textit{ accepts } w \\ \textit{reject if } M \textit{ does not accept } w \end{cases}$$

A_{TM} is undecidable

Proof.

Now we construct a new TM D with H as a subroutine. This new TM calls H to determine what M does when the input to M is its own description $< M >$. Once D has determined this information, it does the opposite. That is, it rejects if M accepts and accepts if M does not accept. The following is a description of D

$D =$ On input $< M >$, where M is a TM:

1. Run H on input $< M, < M >>$
2. Output the opposite of what H outputs: if H accepts, reject; and if H rejects, accept

A_{TM} is undecidable

Proof.

Don't be confused by the notion of running a machine on its own description! That is similar to running a program with itself as input, something that does occasionally occur in practice. For example, a compiler is a program that translates other programs. A compiler for the language Python may itself be written in Python, so running that program on itself would make sense. To summarise we have

$$D(< M >) = \begin{cases} \textit{accept if } M \textit{ does not accepts } < M > \\ \textit{reject if } M \textit{ accepts } < M > \end{cases}$$

A_{TM} is undecidable

Proof.

What happens when we run D with its own description $< D >$ as input? In that case, we get

$$D(< D >) = \begin{cases} \textit{accept if } D \textit{ does not accepts } < D > \\ \textit{reject if } D \textit{ accepts } < D > \end{cases}$$

No matter what D does, it is forced to do the opposite, which is obviously a contradiction. Thus, neither TM D nor TM H can exist □

A_{TM} is undecidable

Where is the diagonalization in the proof? It becomes apparent when you examine tables of behavior for TMs H and D. In these tables we list all TMs down the rows: $M_1, M_2, \ldots$ and all their descriptions across the columns: $< M_1 >, < M_2 >, \ldots$ The entries tell whether the machine in a given row accepts the input in a given column. The entry is accept if the machine accepts the input but is blank if it rejects or loops on that input

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	
M_3					$\cdots$
M_4	*accept*	*accept*			
$\vdots$			$\vdots$		

A_{TM} is undecidable

In the following figure, the entries are the results of running H on inputs corresponding to previous. So if M_3 does not accept input $< M_2 >$, the entry for row M_3 and column $< M_2 >$ is reject because H rejects input $< M_3, < M_2 >>$

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$			$\vdots$		

A_{TM} is undecidable

In the following figure, we add D. By our assumption, H is a TM and so is D. Therefore, it must occur on the list of $M_1, M_2, \ldots$. Note that D computes the opposite of the diagonal entries. The contradiction occurs at the point of the question mark where the entry must be the opposite of itself

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$			$\vdots$		$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		?	
$\vdots$			$\vdots$				$\ddots$

Not Turing-recognizable

We showed that A_{TM} is is Turing-recognizable. The following theorem shows that if both a language and its complement are Turing-recognizable, the language is decidable. Hence for any undecidable language, either it or its complement is not Turing-recognizable. Recall that the complement of a language is the language consisting of all strings that are not in the language. We say that a language is **co-Turing-recognizable** if it is the complement of a Turing-recognizable language.

Theorem

A language is decidable iff it is Turing-recognizable and co-Turing-recognizable

Rephrasing: a language is decidable exactly when both it and its complement are Turing-recognizable

Not Turing-recognizable

Proof.

We have two directions to prove. First, if A is decidable, we can easily see that both A and its complement A are Turing-recognizable. Any decidable language is Turing-recognizable, and the complement of a decidable language also is decidable. For the other direction, if both A and $\overline{A}$ are Turing-recognizable, we let M_1 be the recognizer for A and M_2 be the recognizer for A. The following TM M is a decider for A

M = On input w:

1. Run both M_1 and M_2 in parallel
2. If M_1 accepts, *accept*; if M_2 accepts, *reject*

Not Turing-recognizable

Proof.

Now we show that M decides A. Every string w is either in A or $\overline{A}$. Therefore, either M_1 or M_2 must accept w. Because M halts whenever M_1 or M_2 accepts, M always halts and so it is a decider. Furthermore, it accepts all strings in A and rejects all strings not in A. So M is a decider for A, and thus A is decidable. □

Not Turing-recognizable

Theorem

$\overline{A_{TM}}$ is not Turing-recognizable.

Proof.

We know that A_{TM} is Turing-recognizable. If $\overline{A_{TM}}$ also were Turing-recognizable, A_{TM} would be decidable. We know that A_{TM} is not decidable, so $\overline{A_{TM}}$ must not be Turing-recognizable. □

Section 4

Reduction

Reduction

- A **reduction** is a way of converting one problem to another problem in such a way that a solution to the second problem can be used to solve the first problem. Such reducibilities come up often in everyday life, even if we don't usually refer to them in this way.
- For example, suppose that you want to find your way around a new city. You know that doing so would be easy if you had a map. Thus, you can reduce the problem of finding your way around the city to the problem of obtaining a map of the city
- Reducibility always involves two problems, which we call A and B. If A reduces to B, we can use a solution to B to solve A. So in our example, A is the problem of finding your way around the city and B is the problem of obtaining a map
- Note that reducibility says nothing about solving A or B alone, but only about the solvability of A in the presence of a solution to B

Reduction

- Reducibility also occurs in mathematical problems
- Measuring the area of a rectangle reduces to the problem of measuring its length and width
- Solving a system of linear equations reduces to the problem of inverting a matrix
- Reducibility plays an important role in classifying problems by decidability, and later in complexity theory as well
- When A is reducible to B, solving A cannot be harder than solving B because a solution to B gives a solution to A
- In terms of computability theory, if A is reducible to B and B is decidable, A also is decidable
- Equivalently, if A is undecidable and reducible to B, B is undecidable. This is key to proving that various problems are undecidable
- In short, our method for proving that a problem is undecidable will be to show that some other problem already known to be undecidable reduces to it

Halting problem

Let's consider $HALT_{TM}$, the problem of determining whether a TM halts (by accepting or rejecting) on a given input. This problem is widely known as the **halting problem**. We use the undecidability of A_{TM} to prove the undecidability of the halting problem by reducing A_{TM} to $HALT_{TM}$. Let

$$HALT_{TM} = \{< M, w > | M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem

$HALT_{TM}$ is undecidable

$HALT_{TM}$ is undecidable

Proof idea: This proof is by contradiction. We assume that $HALT_{TM}$ is decidable and use that assumption to show that A_{TM} is decidable which is not true. The key idea is to show that A_{TM} is reducible to $HALT_{TM}$. Let's assume that we have a TM R that decides $HALT_{TM}$. Then we use R to construct S, a TM that decides A_{TM} . To get a feel for the way to construct S, pretend that you are S. Your task is to decide A_{TM} . You are given an input of the form $< M, w >$. You must output **accept** if M accepts w, and you must output **reject** if M loops or rejects on w. Try simulating M on w. If it accepts or rejects, do the same. But you may not be able to determine whether M is looping, and in that case your simulation will not terminate. That's bad because you are a decider and thus never permitted to loop. So this idea by itself does not work.

$HALT_{TM}$ is undecidable

Instead, use the assumption that you have TM R that decides $HALT_{TM}$. With R, you can test whether M halts on w. If R indicates that M doesn't halt on w, reject because $< M.w >$ isn't in A_{TM}. However, if R indicates that M does halt on w, you can do the simulation without any danger of looping.

Thus, if TM R exists, we can decide A_{TM}, but we know that A_{TM} is undecidable. By virtue of this contradiction, we can conclude that R does not exist. Therefore, $HALT_{TM}$ is undecidable.

$HALT_{TM}$ is undecidable

Proof.

Let's assume for the purpose of obtaining a contradiction that TM R decides $HALT_{TM}$. We construct TM S to decide A_{TM} , with S operating as follows.

$S =$ On input $< M, w >$, an encoding of a TM M and a string w

1. Run TM R on input $< M, w >$
2. If R rejects, *reject*
3. If R accepts, simulate M on w until it halts
4. If M has accepted, *accept*; if M has rejected, *reject*

Clearly, if R decides $HALT_{TM}$, then S decides A_{TM} . Because A_{TM} is undecidable, $HALT_{TM}$ also must be undecidable. □

E_{TM} is undecidable

Previous theorem illustrates our strategy for proving that a problem is undecidable. This strategy is common to most proofs of undecidability, except for the undecidability of A_{TM} itself, which is proved directly via the diagonalization method.

We now study several other theorems and their proofs as further examples of the reducibility method for proving undecidability. Let

$$E_{TM} = \{< M > | M \text{ is a TM and } L(M) = \emptyset\}$$

Theorem

E_{TM} is undecidable

E_{TM} is undecidable

Proof idea: We assume that E_{TM} is decidable and then show that A_{TM} is decidable getting a contradiction. Let R be a TM that decides E_{TM}. We use R to construct TM S that decides A_{TM}. How will S work when it receives input $< M, w >$? One idea is for S to run R on input $< M >$ and see whether it accepts. If it does, we know that $L(M)$ is empty and therefore that M does not accept w. But if R rejects $< M >$, all we know is that $L(M)$ is not empty and therefore that M accepts some string, however we still do not know whether M accepts the particular string w. So we need to use a different idea.

Instead of running R on $< M >$, we run R on a modification of $< M >$. We modify $< M >$ to guarantee that M rejects all strings except w, but on input w it works as usual. Then we use R to determine whether the modified machine recognizes the empty language. The only string the machine can now accept is w, so its language will be nonempty iff it accepts w. If R accepts when it is fed a description of the modified machine, we know that the modified machine doesn't accept anything and that M doesn't accept w.

E_{TM} is undecidable

Proof.

Let's create the modified machine described in the proof idea using our standard notation. We call it M_1

$M_1 =$ On input x:

1. if $x \neq w$, *reject*
2. if $x = w$, run M on input w and *accept* if M does

This machine has the string w as part of its description. It conducts the test of whether $x = w$ in the obvious way, by scanning the input and comparing it character by character with w to determine whether they are the same. Putting all this together, we assume that TM R decides E_{TM} and construct TM S that decides A_{TM} as follows

E_{TM} is undecidable

Proof.

S = In input $< M, w >$, an encoding of a TM M and a string w:

1. Use the description of M and w to construct the TM M_1 just described
2. run R in input $< M_1 >$
3. if R accepts, *reject*; if R rejects, *accept*

S is able to compute a description of M_1 from a description of M and w. It is able to do so because it only needs to add extra states to M that perform the $x = w$ test. If R were a decider for E_{TM}, S would be a decider for A_{TM}. A decider for A_{TM} cannot exist, so we know that E_{TM} must be undecidable. □

$REGULAR_{TM}$ is undecidable

Another interesting computational problem regarding TM concerns determining whether a given TM recognizes a language that also can be recognized by a simpler computational model. For example, we let $REGULAR_{TM}$ be the problem of determining whether a given TM has an equivalent finite automaton. This problem is the same as determining whether the TM recognizes a regular language. Let

$$REGULAR_{TM} = \{< M > | M \text{ is a TM and } L(M) \text{ is a regular language}\}$$

Theorem

$REGULAR_{TM}$ *is undecidable*

$REGULAR_{TM}$ is undecidable

Proof idea: This proof is by reduction from A_{TM} . We assume that $REGULAR_{TM}$ is decidable by a TM R and use this assumption to construct a TM S that decides A_{TM} . Less obvious now is how to use R's ability to assist S in its task.

The idea is for S to take its input $< M, w >$ and modify M so that the resulting TM recognizes a regular language if and only if M accepts w. We call the modified machine M_2 . We design M_2 to recognize the nonregular language $\{0^n1^n | n \geq 0\}$ if M does not accept w, and to recognize the regular language Σ^* if M accepts w. We must specify how S can construct such an M_2 from M and w. Here, M_2 works by automatically accepting all strings in $\{0^n1^n | n \geq 0\}$. In addition, if M accepts w, M_2 accepts all other strings.

We construct M_2 only for the purpose of feeding its description into the decider for $REGULAR_{TM}$ that we have assumed to exist. Once this decider returns its answer, we can use it to obtain the answer to whether M accepts w. Thus, we can decide A_{TM}, a contradiction.

$REGULAR_{TM}$ is undecidable

Proof.

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

1. Construct the following TM M_2
   $M_2 =$ On input x:
   1. If x has the form 0^n1^n, *accept*
   2. if x does not have this form, run M on input w and *accept* if M accepts w
2. Run R in input $< M_2 >$
3. If R accepts, *accept*; if R rejects, *reject*

□

Rice's theorem

Similarly, the problems of testing whether the language of a TM is a context-free language, a decidable language, or even a finite language can be shown to be undecidable with similar proofs. In fact, a general result, called **Rice's theorem**, states that determining any property of the languages recognized by TM is undecidable.

EQ_{TM} is undecidable

So far, our strategy for proving languages undecidable involves a reduction from A_{TM}.
Sometimes reducing from some other undecidable language, such as E_{TM} , is more convenient when we are showing that certain languages are undecidable.
We could prove it by a reduction from A_{TM}, but we use this opportunity to give an example of an undecidability proof by reduction from E_{TM}. Let

$$EQ_{TM} = \{< M_1, M_2 > | M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

Theorem

EQ_{TM} is undecidable

EQ_{TM} is undecidable

Proof idea: Show that if EQ_{TM} were decidable, E_{TM} also would be decidable by giving a reduction from E_{TM} to EQ_{TM}. The idea is simple. E_{TM} is the problem of determining whether the language of a TM is empty. EQ_{TM} is the problem of determining whether the languages of two TMs are the same. If one of these languages happens to be $\emptyset$, we end up with the problem of determining whether the language of the other machine is empty—that is, the E_{TM} problem. So in a sense, the E_{TM} problem is a special case of the EQ_{TM} problem wherein one of the machines is fixed to recognize the empty language. This idea makes giving the reduction easy.

EQ_{TM} is undecidable

Proof.

We let TM R decide EQ_{TM} and construct TM S to decide E_{TM} as follows

S = On input $< M >$, where M is a TM:

1. Run R on input $< M, M_1 >$, where M_1 is a TM that rejects all inputs.
2. it R accepts, *accept*; it R rejects, *reject*

□

If R decides EQ_{TM} , S decides E_{TM}. But E_{TM} is undecidable, so EQ_{TM} also must be undecidable.