CHAPTER 4
Decidability

Contents

- Decidable Languages
  - decidable problems concerning regular languages
  - decidable problems concerning context-free languages
- **The Halting Problem**
  - **The diagonalization method**
  - **The halting problem is undecidable**
  - **A Turing unrecognizable languages**

The Acceptance Problem for TMs

- In this section we will encounter several computationally unsolvable problems.
- Here we will learn techniques for proving unsolvability.
- Consider the problem of testing whether a Turing Machine accepts a given input.

$$A_{TM} = \{< M, w >: M \text{ is a TM that accepts input string } w\}.$$

Theorem 9: A_{TM} is undecidable.

- First we observe that A_{TM} is *Turing-recognizable*. (hence *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 TM *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.* "

- Note, this machine loops on input <*M,w*> if *M* loops on *w*.
- If the algorithm had some way to determine that *M* was not halting on *w*, it could *reject.*
- Hence, the ***halting problem***. We will show, an algorithm has no way to make this determination.
- *U* is an example of *universal Turing machine* first proposed by Turing. It is capable of simulating any other Turing machine from the description of that machine.
- The universal Turing machine played an important early role in stimulating the development of stored-program computers.
- **A conclusion**: the general problem of software verification is not solvable algorithmically (by computer).

The Diagonalization Method

• The proof of the undecidability of the halting problem uses a technique called ***diagonalization,*** discovered first by mathematician Georg Cantor in 1873.

• 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 other or whether they are of the same size?

• For finite sets, of course, answering these questions is easy. We 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.

• 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 idea can be extended to infinite sets.

• Assume we have two sets A and B and a function f from A to B.

• Say that f is ***one-to-one*** if it never maps two different elements of A to the same element in B, i.e., if $a \neq b$, *then* $f(a) \neq f(b)$.

• Say that f is ***onto*** if it hits every element of B, i.e., for any b in B there is an a in A such that $f(a)=b$.

• Say that A and B are the same size if there is a one-to-one, onto function $f: A \rightarrow B$.

• A function that is both one-to-one and onto is called a ***correspondence.***

• In a correspondence every element of A maps to a unique element of B and each element of B has a unique element of A mapping to it. A correspondence is simply a way of pairing the elements of A with the elements of B.

Countable sets

• ***Example 1:*** Let *N* be the set of natural numbers {1,2,…,} and *E* is the set of even natural numbers {2,4,…}. Using Cantor's definition of size we can see that *N* and *E* have the same size. The correspondence *f* mapping *N* to *E* is simply *f(n) = 2n*.

n	1	2	3	…
f(n)	2	4	6	…

• A set *A* is ***countable*** if either it is finite or it has the same size as *N*.

• ***Example 2:*** Let *Q* be the set of positive rational numbers, that is $Q = \{\frac{m}{n} : m, n \in N\}$.

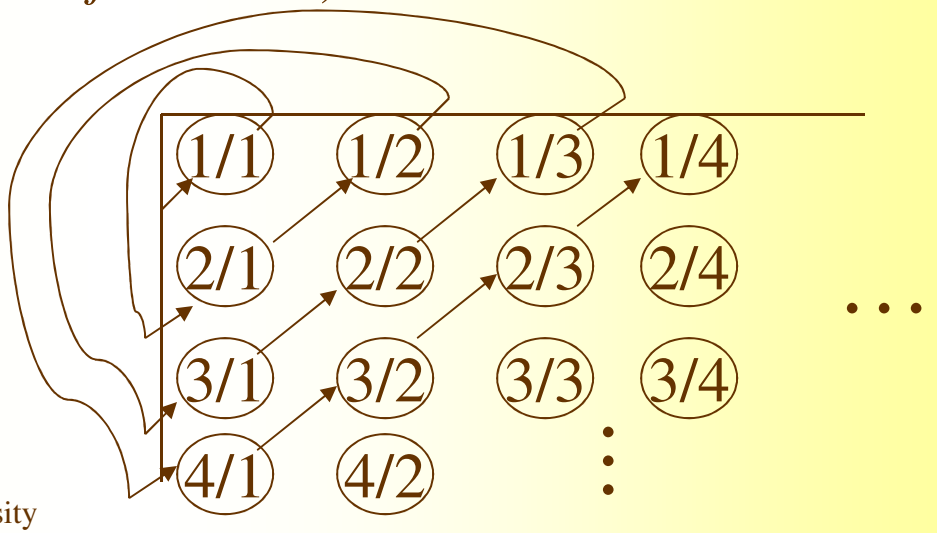
• *Q* seems to be much larger than *N*, yet these two sets have the same size: *Q is countable*.

• We make an infinite matrix containing all the positive rational numbers, as shown (the number *i/j* occurs in the *i*th row and *j*th column).

• Then we turn this matrix into a list.

• We skip an element if

it would cause a repetition.



Uncountable sets

- For some infinite sets no correspondence with N exists. Such sets are called ***uncountable.*** (*These sets simply too big.*)
- A ***real number*** is one that has a decimal representation ($\pi = 3.1415926..., \sqrt{2} = 1.4142135...$)
- ***Example:*** The set of all real numbers R is uncountable.
- Cantor proved this by using the diagonalization method.

- We show by contradiction that no correspondence exists between N and R.
- Suppose the correspondence f existed.
- f must pair all the members of N with all the members of R.
- But we will find an x in R that is not paired with anything in N, which will be our contradiction.
- We will construct this x. We choose each digit of x to make x different from one of the real numbers that is paired with an element of N.
- In the end we are sure that x is different from any real number that is paired.
- We illustrate this idea by giving an example.

We give decimal representation of x. It is a number between 0 and 1. Our objective is to ensure that x is not $f(n)$ for any n.

$x = 0.4641\ldots$ ←

n	$f(n)$
1	3.**1**4159…
2	5.5**5**555…
3	0.12**3**45…
4	0.500**0**0…
…	…

- We know that x is not $f(n)$ for any n since it differs from $f(n)$ in the nth fractional digit.

There are languages that are not T-recognizable.

- The previous result 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.
- Since 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.
- We need only to show that the set of all Turing machines is countable and the set of all languages is uncountable.

The set of all Turing machines is countable.

- First we 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 Turing machines is countable because each Turing machine *M* has an encoding into a string <*M*>.
- If we simply omit those strings that are not legal encoding of Turing machines, we can obtain a list of all Turing machines.

The set of all languages is uncountable.

- First we observe that the set B of all infinite binary sequences is uncountable.
- An ***infinite binary sequence*** is an unending sequence of 0s and 1s.
- We can show that B is uncountable by using a proof by diagonalization similar to the one we used to show that R is uncountable.

- Let L be the set of all languages over alphabet Σ.
- We show that L is uncountable by giving an correspondence with B, thus showing the two sets are the same size.
- Let $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$.
- Each language A from L has a unique sequence in 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,

$$\Sigma^* = \{\varepsilon,\ 0,\ 1,\ 00,\ 01,\ 10,\ 11,\ 000,\ 001,\ \ldots\};$$
$$A = \{\quad 0,\quad 00,\ 01,\quad\quad 000,\ 001,\ \ldots\};$$
$$\chi_A = \ 0\ \ 1\ 0\ \ 1\quad 1\quad 0\quad 0\quad 1\quad 1\quad \ldots\ .$$

- The function $f: L \rightarrow B$, where $f(A)$ equals the characteristic sequence of A, is one-to-one and onto and hence a correspondence.
- Therefore, as B is uncountable, L is uncountable as well.
- ***Thus, indeed some languages are not recognizable by any Turing Machine.***

The Acceptance Problem for TMs is undecidable

- We are ready to prove theorem 9

Theorem 9: A_{TM} is undecidable.

$$A_{TM} = \{< M, w >: M \text{ is a TM that accepts input string } w\}.$$

- We assume that A_{TM} is decidable and obtain a contradiction.
- Let H be decider for A_{TM}, that is, on input <*M*,*w*>, where *M* is a TM and *w* is a string,

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

M rejects *w* or works forever

- Consider now a TM *D* with *H* as a subroutine.

D = "on input <*M*>, where *M* is a TM:

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

- That is, $$D(< M >) = \begin{cases} accept, & \text{if } M \text{ does not accept } < M > \\ reject, & \text{if } M \text{ accepts } < M > \end{cases}$$
- What happens when we run *D* on own description <*D*> as input?

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

- This is obviously a contradiction. Hence neither TM *D* nor TM *H* can exist.

The Acceptance Problem for TMs is undecidable(cont.)

- Where is the diagonalization in the proof of theorem 9? We had
  - *H* accepts <*M,w*> exactly when *M* accepts *w*,
  - *D* rejects <*M*> exactly when *M* accepts <*M*>,
  - *D* rejects <*D*> exactly when *D* accepts <*D*>. ← (a contradiction)

	<M1>	<M2>	<M3>	<M4>	. . .
M1	*accept*		*accept*		
M2	*accept*	*accept*	*accept*	*accept*	
M3				*accept*	⋯
M4	*accept*	*accept*			
⋮			⋮		

Entry *i,j* is *accept* if *Mi* accepts <*Mj*>.

	<M1>	<M2>	<M3>	<M4>	. . .
M1	*accept*	*reject*	*accept*	*reject*	
M2	*accept*	*accept*	*accept*	*accept*	
M3	*reject*	*reject*	*reject*	*accept*	⋯
M4	*accept*	*accept*	*reject*	*reject*	
⋮			⋮		

Entry *i,j* is the value of *H* on input <*Mi,*<*Mj*>>.

	<M1>	<M2>	<M3>	<M4>	. . .	<D>	. . .
M1	*accept*	*reject*	*accept*	*reject*		*accept*	
M2	*accept*	*accept*	*accept*	*accept*		*reject*	
M3	*reject*	*reject*	*reject*	*accept*	⋯	*accept*	⋯
M4	*accept*	*accept*	*reject*	*reject*		*reject*	
⋮			⋮		⋱		
D	*reject*	*reject*	*accept*	*accept*		??	
⋮			⋮				⋱

If *D* is in the figure, a contradiction occurs at "?".

A Turing-unrecognizable language.

- We have seen a language that is undecidable. Now we demonstrate a language which is not even Turing-recognizable.
- Recall that the ***complement*** of a language L is the language $\overline{L}$ consisting of all strings that are not in the language L.
- We say that a language is ***co-Turing-recognizable*** if it is the complement of a Turing-recognizable language.

Theorem: A language is decidable if and only if it is both Turing-recognizable and co-Turing-recognizable.

Proof: ($\rightarrow$) Clearly, if L is decidable then both L and $\overline{L}$ are Turing-recognizable.

- any decidable language is Turing-recognizable and
- the complement of decidable language is decidable.

($\leftarrow$) Let *M1* be the recognizer for L and *M2* be the recognizer for $\overline{L}$.

The following TM M is a decider for L.

M = "on input w:

1. Run both *M1* and *M2* on input w in parallel.
2. If *M1* accepts, *accept*; if *M2* accepts, *reject*."

- Run in parallel means that M has two tapes, one for simulating *M1*, and the other for simulating *M2*
- M takes turns simulating one step of each machine; it continues until one of them halts with accept.
- Since every string is in L or $\overline{L}$, either *M1* or *M2* must accept w. M always halts; it is a decider.
- Moreover it accepts all strings in L and rejects all strings not in L. Hence, L is decidable.

A Turing unrecognizable language (cont.)

Corollary: $\overline{A_{TM}}$ 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.
- But it is not decidable by theorem 9.
- Hence, $\overline{A_{TM}}$ is not Turing-recognizable.