CSE 322 - Introduction to Formal Methods in Computer Science
Decidability

Dave Bacon
Department of Computer Science & Engineering, University of Washington

In the last lecture we introduced the Turing machine. We talked about how it worked, and showed an example of a Turing machine which accepted a particular language. Further we talked a bit about the difference between Turing-recognized (it can loop) and Turing-decidable.

Okay so one thing I forgot to talk about is why the heck we care about Turing machines. Well one very important reason is that they seem to capture what it means to compute in our world. That is to say that no one has thought up a physical way to build a machine which does something different that a Turing machine can do. This hypothesis, that Turing machines capture what we mean by a computer, i.e. by something which can compute an algorithm, is known as the Church-Turing thesis. It just a thesis, because we don't know whether it is true or not: however the overwhelming evidence is that it is correct. (As a side note, the computers I study, quantum computers, while they probably compute more efficiently, probably don't violate the Church-Turing thesis. You can use a classical computer to simulate a quantum computer: albeit at probably a high cost in running time.) The Church-Turing thesis is nice, not just because it is probably true, but because it means that instead of talking about Turing machines in detail we can talk about algorithms in a less formal way and still be guaranteed to not be moving outside of the realm of what a Turing machine can do. (Note that if we include the efficiency of our algorithm in our analysis we do have to be a bit careful: quantum computers are evidence that the strong Church-Turing thesis is not true.) The strong Church-Turing thesis says that not only does a Turing machine capture what it means to be an algorithm, but that all models of computation are roughly of equal computational power,

Okay, with a brief interlude about the Church-Turing thesis, lets turn to today's topic: decidability.

First lets talk about some problems which are decidable. This will lead us naturally to the next question: are there languages which are not decidable? We can also ask whether there are languages which are not Turing-recognizable. We will answer both of these in time.

I. SOME DECIDABLE LANGUAGES

Consider the language,

$$A_{DFA} = \{\langle B, w\rangle | B \text{ is a DFA which accepts the input string } w\}$$

This is the acceptance problem for DFAs. Given a description of a DFA and an input string, does this DFA accept the string w. So is this question decidable? We claim that it is decidable.

The basic idea is simple. We can construct a Turing machine M, which simulates the DFA. Actually first of all our machine M, should check to make sure the DFA written on the tape is a valid DFA. If the DFA is specified as five-tuple, the we need to check whether this DFA is a valid DFA. It is easy to think of an algorithm which does this. If it is not a valid input, then the Turing machine rejects. Now given that the B is a valid Turing machine, we can simulate the DFA using a Turing machine. Basically the machine needs to keep track of the state and keep track of a where in the input string the machine is working. Then when the simulation is done, if the machine is in an accept state, then the Turing machine accepts. If it is in a reject state at the end of the simulation, then the Turing machine rejects.

Note that the Turing machine we have described is a decider: it always either rejects or accepts, and there is no possibility of it going on forever without rejecting or accepting. Thus we have just shown that

A_{DFA} is a decidable language.

Okay, so that was a rather easy one to show decidable (thought we didn't really do this formally, which is much more tricky!) Now lets turn to another example. Consider the analogous language for context-free languages,

$$A_{CFG} = \{\langle g, w\rangle | B \text{ is a CFG that generates string } w\}$$

Now one way you might consider constructing a Turing machine which decides this language is to try all the derivations using different rules, starting with the the start variable. But this would create a Turing machine which recognizes A_{CFG} but not one which decides A_{CFG}. But we already know a way around this: the CKY algorithm! This algorithm

had a running time which was $O(|w|^3)$. Further it gave us an answer both when w was generated by the grammar and when w could not be generated by the grammar. Thus we can proceed as we did for A_{DFA} we can first verify that the grammar G is a valid grammar. Then we need to convert this grammar into Chomsky normal form. Finally we then use the CKY algorithm to check whether w can be generated by the grammar. If the CKY algorithm finds that w is generated by the grammar we accept. If the grammar is not properly formed, or the CKY algorithm find that w is not generated by the grammar, we reject. This shows that

A_{CFG} is a decidable language

II. THE HALTING PROBLEM

Okay so we've just shown that two problems about FAs and CFGs are decidable. It is then natural to define the language

$$A_{TM} = \{\langle M, w\rangle | M \text{ is a Turing Machine and accepts } w\}$$

Okay, so is this machine Turing-decidable? Uh-oh. This language is not Turing decidable. Doh. We will prove this in a bit, but first lets show that A_{TM} is Turing-recognizable.

Why is A_{TM} Turing-recognizable? Well consider a Turing machine U which recognizes A_{TM}. On input $\langle M, w\rangle$, where M is a Turing Machine and w is a string, simulate M on input w. If M ever enters into its accept state, then accept. If it ever enteres into its reject state, reject. Note that we have to show that it is possible to devise a machine which can simulate a Turing machine from its description. Such a Turing machine is called a *Universal Turing machine*. We won't explicitly construct such a machine, but only note that it is possible to do so. Indeed this very idea: that there is a machine which can take as input a description of a Turing machine and an input, and then simulate the Turing machine on that input, is at the very heart of what we mean by programming.

Okay, so we claim that A_{TM} is Turing-recognizable. This is certainly true as the machine will accept and reject appropriately. However if A_{TM} loops forever, then our machine U will loop forever. Thus the machine will not accept or reject all inputs and so it is not a decider.

So what makes A_{TM} not decidable? We'll see in a second. First note, however, that what we really want is a machine which rejects when $\langle M, w\rangle$ halts. The crux of the issues is this: is there an algorithm for deciding when a Turing machine halts on a specific input. This is why this problem is called the halting problem. The famous halting problem!

III. DIAGONALIZATION

Okay, our eventual goal is to show that A_{TM} is not decidable. But before we can prove this we need some mathematical techniques.

Consider functions f from set A to set B. Recall that a function is *one-to-one* if it never maps two different elements to the same place, i.e. if $f(a) \neq f(b)$ whenever $a \neq b$. A function is *onto* if it hits every element of B: if for every $b \in B$, there is an a such that $f(a) = b$. A function which is one-to-one and onto is a correspondence.

Okay so why all the math-speak? Well consider the following problem. The set of natural numbers $\mathcal{N} = \{1, 2, 3, \ldots\}$ is infinite. The set of even natural numbers $\mathcal{E} = \{2, 4, 6, \ldots\}$ is infinite. But which is bigger? For finite sets, we can always count up the elements to tell which is bigger, but for infinite sets, is there any way to determine whether one infinite set is bigger than the other? This problem tormented a lot of people, and it took the genius of Georg Cantor in 1873 to produce a reasonable idea about this problem. Cantor said: two finite sets have the same size if the there is a function from one set to the other which is a correspondence. In other words two finite sets have the same size if the elements of one set can be paired with the elements of the other set. Notice that this definition of size does not actually count! So Cantor suggested that we extend this idea to infinite sets.

Consider the question we posed that the beginning are there more natural numbers or more even natural numbers? Well consider the following function f from $\mathcal{N}$ to $\mathcal{E}$: $f(n) = 2n$. It is easy to see that this function is a correspondence. Thus we conclude that $\mathcal{N}$ and $\mathcal{E}$ have the same size. This definition may seem a bit bizarre considering that $\mathcal{E}$ is a proper subset of $\mathcal{N}$: but pairing each number also seems like a reasonable definition of comparing sets! Now a little terminology. A set A is countable if either it is finite or it has the same size as $\mathcal{N}$. Thus if we can find a correspondence between $\mathcal{N}$ and an infinite set $\mathcal{A}$, then we will have shown that $\mathcal{A}$ is countable.

Let's do a slightly more complicated example. Let $\mathcal{Q} = \{\frac{m}{n} | m, n \in \mathcal{N}\}$. Q is the set of positive rational numbers. At first thought it seems that $\mathcal{Q}$ is larger than $\mathcal{N}$. But is it? Suppose you made an infinite table of all of the possible fraction. Make this table have columns and rows labeled by the $\mathcal{N}$ such that an entry at the ith row and jth

column is $\frac{i}{j}$. Now is it possible to design a function f from $\mathcal{N}$ to $\mathcal{Q}$ which is a correspondence. Clearly, taking one row and maping $\mathcal{N}$ to this row isn't any good. While this will produce a correspondence for this row it won't be a correspondence for all of $\mathcal{Q}$. Consider a path which, instead of going across rows, goes along diagonals going up and right in the table. We can these paths together by taking the end of one path and trying it to a path which is the beginning of the path right below it. Then we can imagine following along this path. For example considering the upper right of the table Then the path we have described follows $\frac{1}{1}$, $\frac{2}{1}$, $\frac{1}{2}$, $\frac{3}{1}$, $\frac{2}{2}$, $\frac{1}{3}$, etc. So now we can describe how

	1	2	3	4
1	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$
2	$\frac{2}{1}$	$\frac{2}{2}$	$\frac{2}{3}$	$\frac{2}{4}$
3	$\frac{3}{1}$	$\frac{3}{2}$	$\frac{3}{3}$	$\frac{3}{4}$
4	$\frac{4}{1}$	$\frac{4}{2}$	$\frac{4}{3}$	$\frac{4}{4}$

to construct a function f from $\mathcal{N}$ to $\mathcal{Q}$ which is a correspondence. We do this by mapping each successive element of f to an element in the above path: except when that element has appeared before. In this case we skip to the next element. Okay so this function is a mapping from $\mathcal{N}$ to $\mathcal{Q}$ and it is a correspondence. So that means that $\mathcal{Q}$ is countably infinite.

At this point you might think that all infinite sets are countably infinite. But this isn't true and this is one of the coolest results of Cantor's idea. As an example of this consider the set of real number $\mathbb{R}$. Recall that real numbers are those which have a decimal representation. Things like $\pi = 3.1415926\ldots$ and $\sqrt{2} = 1.4142135\ldots$ are real numbers. Let us now show that $\mathbb{R}$ is not countable. We will do this via a proof by contradiction. Assume that $\mathbb{R}$ is countable. Then there exists a function f from $\mathcal{N}$ to $\mathbb{R}$ which is a correspondence. Let's show that if this where true, then there would exist and element of $\mathcal{R}$ which does not appear as in the range of f. Note that we will do this independent of how that hypothetical f was constructed. So how do we do this. Well consider the real number $0.x_1x_2x_3\ldots$. We choose x_i such that it is different from the ith digit after the decimal point of $f(i)$. For example if $f(1) = 3.333$ and $f(2) = 23.201$, etc, then x_1 would have to be different from 3, x_2 would have to be different from 0, etc. It is always possible to construct such a real number $0.x_1x_2x_3\ldots$. But this number cannot be in the range of f: it is guaranteed by construction to be different from all of the elements in the range. So we have shown that if there is a function which is a correspondence from $\mathcal{N}$ to $\mathbb{R}$, then there must exist an element of the reals which is not in the range of this function, which is a contradiction. Thus there must not exist such a correspondence. In other words we have shown that real numbers are not countably infinite! (There is a slight subtly in our proof: certain real numbers like $0.19999\ldots$ and $0.20000\ldots$ are actually equal. We can avoid this by never selecting 0 and 9 when we select a new digit.)

The proof we have just described is called diagonalization because we took elements from the diagonal of the function range. This is a very cool proof technique.

IV. SOME LANGUAGES ARE NOT TURING-RECOGNIZABLE

Let's use the diagonalization argument to show that

Some languages are not Turing-recognizable

We will show this by noting that every Turing machine recognizes only one language, but that there are more languages than there are Turing machines. Thus since there are more languages than machines, there must exist languages which are not recognized. Note that both the sets we are talking about here are infinite. Thus our argument is going to be about the sizes of infinite sets.

Consider first the set of all Turing machines. First of all note that Σ^* is countable. We can write down a list of all Σ^* by writing down all stringsf size 0, all strings of size 1, all strings of size 2 etc. This list is a correspondence between $\mathcal{N}$ and Σ^*. Thus Σ^* is a countable set. Next note that a description of a Turing machine is just a string, i.e. is a subset of Σ^* for some appropriate Σ. Thus we can make a list of all Turing machines by using the listing for all strings, but excluding all strings which are not valid Turing machines. Thus the set of descriptions of Turing machines is countably infinite.

Next we will show that the set of all languages is not countably infinite. To show this we first note that the set of all infinite binary sequences is uncountable. An infinite binary sequence is an unending sequence of 0s and 1s. To show that this set is uncountable, we can use a diagonalization argument similar to the one we used to show that $\mathbb{R}$ is not countably infinite. Thus the set of all infinite binary sequences is not countably infinite.

Continuing we will now show that the set of all languages is not countably infinite. Let $\mathcal{L}$ be the set of all languages over alphabet Σ. We show that $\mathcal{L}$ is uncountable by given a correspondence between $\mathcal{L}$ and the set of all infinite binary sequences. To see how to do this note that every element of $\mathcal{L}$ is a language. Let $\Sigma^* = \{s_1, s_2, \ldots\}$. Then every language has a uniques sequence of 0s and 1s formed by letting the ith bit be equal to 0 if s_i is not in the language and 1 if s_i is in the language. Thus every language has a unique infinite binary string. This is a correspondence between $\mathcal{L}$ and the set of all infinite binary sequences. Thus we have shown that there is a correspondence between these two sets. Since there is a correspondence between these two sets of and the set of all infinite binary sequences is not countable, the set of all languages is not countable.

We have now shown that some languages are not Turing-recognizble. The set of all Turing machines is countably infinite, but the set of all languages is not countably infinite. Thus there must exist a language which is not recognized by a Turing machine. Wahlah!

V. THE HALTING PROBLEM IS NOT DECIDABLE

Now we will show one of the coolest results of all time:

A_{TM} is not decidable.

Our proof will invoke the diagonalization proof, but this won't be immediately obvious.

Suppose that A_{TM} is decidable (we are doing a proof by contradiction.) Then there is a decider, call it H which decides A_{TM}. In other words there exists a Turing machine H which acts as

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

Now we will construct a new machine D which uses H has a subroutine. This machine will, on input $\langle M\rangle$ where M is a TM run H on $\langle M, \langle M\rangle\rangle$. It will then output the opposite of what H outputs. Note that this machine is running H on a description of the machine M. A quick way to summarize this is to note say that

$$D(\langle M\rangle) = \begin{cases} \text{accept, if } M \text{ does not accept } \langle M\rangle \\ \text{reject, if } M \text{ accepts } \langle M\rangle \end{cases}$$

Now consider running D on a description of D, $\langle D\rangle$. Then we obtain

$$D(\langle D\rangle) = \begin{cases} \text{accept, if } D \text{ does not accept } \langle D\rangle \\ \text{reject, if } D \text{ accepts } \langle D\rangle \end{cases}$$

No matter what D does, we always run into a contradiction! Thus neither D nor H can exist. Thus there can be no decided H which decides A_{TM}. The halting problem is not decidable.

Now where was the digaonalization in this proof? The basic idea is to examine a table whose rows are Turing machines and whose columns are descriptions of the Turing machines. Then this table can be filled in with "accepts" where the Turing machine accepts on the relevant description of a turing machine and is blank if the Turing machine rejects or loops. Now H by construction is supposed to be able to fill in this table with accepts and rejects for every input. But now consider D. D calculates the diagonal elements of our table. Actually it computes the oppose of this diagonal. But then how do we compute D on $\langle D\rangle$ from this table? It must be the opposite of itself and therefore we arrive at a contradiction.

VI. LANGUAGES WHICH ARE NOT TURING-RECOGNIZABLE

We have seen that A_{TM} is not Turing-deciable. However we have also seen that A_{TM} is Turing-recognizable. Do there exist languages which are not Turing-recognizable? Here we will show that this is ture.

First a definition. We say that a language is *co-Turing-recognizable* if it is the complement of a Turing -recognizable language. Using this definition we now claim that

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

To prove this we must show both directions. First the forward direction. If a language is decidable then it is clearly Turing-recognizable. It is also co-Turing-recognizable, because we can simply change the accept and reject states. Thus if a language is decidable, then it is Turing-recognizable and co-Turing recognizable. For the backwards direction, consider the case that A and the complement of A, $\bar{A}$ are Turing-recognizable. Then we can take two recognizers for A and $\bar{A}$ and run them in parallel on a Turing machine, M. If the machine for A accepts, then we accept, and if the machine for $\bar{A}$ accepts, then we reject. This machine will decide A. Let M_1 be the recognizer which recognizes A and M_2 be the recognizer which recognizes $\bar{A}$. Every string w is in either A or $\bar{A}$. Therefore either M_1 or M_2 must accept w. Further M halts whenever M_1 or M_2 accepts. So our new machine always halts and is therefore a decider. Thus A is decidable. This proves the claim.

Now we can construct a language which is not Turing-recognizable. in particular $\bar{A_{TM}}$ is not Turing-recognizable. Why? Because we know that A_{TM} is Turing-recognizable. If $\bar{A_{TM}}$ were also Turing-recognziable, then A_{TM} would be decidable. But we know that A_{TM} is not decidable. Thus $\bar{A_{TM}}$ is not Turing-recognizable.