Halting Problem

CSE 311 Spring 2022
Lecture 29

Announcements

Remember the final is on Monday starting at 12:30!

Do not come to the exam if:

- you're required to isolate (according to the university's flow chart)
- you think it would be wise to isolate, even if not officially required.

In both cases, send email to Robbie as soon as you know and we'll schedule a conflict exam for you.

If you got an email from me, you're in the small room (CSE2 G04).

If not, you're in Kane 120.

- In Kane: we'll ask those not wearing masks to sit toward the front, and those with lower comfort levels/desiring to wear masks to sit toward the back.
- Sitting every other seat.

Irregularity: one more example

Full outline

1. Suppose for the sake of contradiction that L is regular. Then there is some DFA M that recognizes L.

2. Let S be [fill in with an infinite set of prefixes].

3. Because the DFA is finite and S is infinite, there are two (different) strings x, y in S such that x and y go to the same state when read by M *[you don't get to control x, y other than having them not equal and in S]*

4. Consider the string z [argue exactly one of xz, yz will be in L]

5. Since x, y both end up in the same state, and we appended the same z, both xz and yz end up in the same state of M. Since $xz \in L$and $yz \notin L$, M does not recognize L. But that's a contradiction!

6. So L must be an irregular language.

Practical Tips

When you're choosing the set S, think about what the DFA would "have to count"

That is fundamentally why a language is irregular. The set S is the way we prove it! Whatever we "need to remember" it's different for every element of S.

If your strings have an "obvious middle" (like between the 0's and 1's) that's a good place to start.

Let’s Try another

The set of strings with balanced parentheses is not regular.

What do you want S to be? What would you have to count?

The number of unclosed parentheses.

Let $S = \ldots$

Let’s Try another

The set of strings with balanced parentheses is not regular.

What do you want S to be? What would you have to count?

The number of unclosed parentheses.

Want S to be a set with infinitely many strings with different numbers of unclosed parentheses.

Let $S = (^*$

Outline for (*

1. Suppose for the sake of contradiction that L is regular. Then there is some DFA M that recognizes L.

2. Let S be (*

3. Because the DFA is finite and S is infinite, there are two (different) strings x, y in S such that x and y go to the same state when read by M Observe that $x = (^a$ for some integer a, $y = (^b$ for some integer b with $a \neq b$.

4. Consider the string z [argue exactly one of xz, yz will be in L]

5. Since x, y both end up in the same state, and we appended the same z, both xz and yz end up in the same state of M. Since $xz \in L$and $yz \notin L$, M does not recognize L. But that's a contradiction!

6. So L must be an irregular language.

Full outline

1. Suppose for the sake of contradiction that L is regular. Then there is some DFA M that recognizes L.

2. Let S be $(^*$

3. Because the DFA is finite and S is infinite, there are two (different) strings x, y in S such that x and y go to the same state when read by M Observe that $x = (^a$ for some integer a, $y = (^\mathrm{b}$ for some integer b with $a \neq b$.

4. Consider the string z=$)^a$ xz is a balanced set of parentheses (since there are the same number of each and all the open-parentheses come before the close parentheses). But yz is not balanced because $a \neq b$.

5. Since x, y both end up in the same state, and we appended the same z, both xz and yz end up in the same state of M. Since $xz \in L$and $yz \notin L$, M does not recognize L. But that's a contradiction!

6. So L must be an irregular language.

One more, just the key steps

What about $\{a^k b^k c^k : k \geq 0\}$?

One more, just the key steps

What about $\{a^k b^k c^k : k \geq 0\}$?

$S = \{a^k : k \geq 0\}$ or $S = \{a^k b^k : k \geq 0\}$ are equally good choices.

Your suffix is $b^j c^j$ for the first and c^j for the second.

$S = \{a^k b : k \geq 0\}$ also works. $z = b^{j-1} c^j$ is your suffix. The proof is a little more tedious, but you can make it through.

Uncountability/Halting

Let's Do Another!

Let $B = \{0,1\}$. Call a function $g: \mathbb{N} \to B$ a "binary valued function"

Intuitively, g would be something like
public boolean g(BigInteger input){ }

If we could write that g in Java.

How many possible $g: \mathbb{N} \to B$ are there?

Our Second big takeaway

How many Java methods can we write:

public boolean g(int input) ?

Can you list them?

Yeah!! Put them in **lexicographic** order

i.e. in increasing order of length, with ties broken by alphabetical order.

Wait…that means the number of such Java programs is countable.

And…the number of functions we're supposed to write is uncountable.

Our Second big takeaway

There are more functions $g: \mathbb{N} \rightarrow B$ than there are Java programs to compute them.

Some function must be **uncomputable**.

That is there is no piece of code which tells you the output of the function when you give it the appropriate input.

Not just Java

This isn't just about java programs. (all we used about java was that its programs are strings)…that's…well every programming language.

There are functions that simply cannot be computed.

Doesn't matter how clever you are. How fancy your new programming language is. Just doesn't work.*

*there's a difference between `int` and $\mathbb{N}$ here, for the proof to work you really need all integers to be valid inputs, not just integers in a certain range.

Does this matter?

It’s even worse than that – almost all functions are not computable.

So…how come this has never happened to you?

This might not be meaningful yet. Almost all functions are also inexpressible in a finite amount of English (English is a language too!)

You’ve probably never decided to write a program that computes a function you couldn’t describe in English…

Are there any problems anyone is **interested** in solving that aren’t computable?

A Practical Uncomputable Problem

Every pressed the run button on your code and have it take a long time?

Like an infinitely long time?

What didn’t your compiler…like, tell you **not** to push the button yet.

It tells you when your code doesn’t compile before it runs it…why doesn’t it check for infinite loops?

The Halting Problem

The Halting Problem

Given: source code for a program P and x an input we could give to P
Return: True if P will halt on x, False if it runs forever (e.g. goes in an infinite loop or infinitely recurses)

This would be super useful to solve!

We can't solve it…let's find out why.

A Proof By Contradiction

Suppose, for the sake of contradiction, there is a program H, which given input `P.java`, x will accurately report

"P would halt when run with input x" or

"P will run forever on input x."

Important: H does not just compile `P.java` and run it. To count, H needs to return "halt" or "doesn't" in a finite amount of time.

And remember, it's not a good idea to say "but H has to run P.java to tell if it'll go into an infinite loop" that's what we're trying to prove!!

A Very Tricky Program.

```
Diagonal.java(String x){
    Run H.exe on input <x, x>
    if(H.exe says “x halts on x”)
        while(true){//Go into an infinite loop
            int x=2+2;
        }
    else //H.exe says “x doesn’t halt on x”
        return; //halt.
}
```

So, uhh that’s a weird program.

What do we do with it?

USE IT TO BREAK STUFF

Does `Diagonal.java` halt when its input is `Diagonal.java`?

Let’s assume it does and see what happens…

A Very Tricky Program.

```
Diagonal.java(String x){
   Run H.exe on input <x, x>
   if(H.exe says "x halts on x")
       while(true){//Go into an infinite loop
            int x=2+2;
       }
   else //H.exe says "x doesn't halt on x"
      return; //halt.
}
```

Imagine Diagonal.java halts on Diagonal.java.
Then H better say it halts.
So it goes into an infinite loop.

Wait shoot.

So, uhh that’s a weird program.

What do we do with it?

USE IT TO BREAK STUFF

Does `Diagonal.java` halt when its input is `Diagonal.java`?

Let’s assume it does and see what happens…

That didn’t work.

Let’s assume it doesn’t and see what happens…

A Very Tricky Program.

```
Diagonal.java(String x){
    Run H.exe on input <x, x>
    if(H.exe says “x halts on x”)
        while(true){//Go into an infinite loop
            int x=2+2;
        }
    else //H.exe says “x doesn’t halt on x”
        return; //halt.
}
```

Imagine Diagonal.java doesn’t halt on Diagonal.java.
Then H better say it doesn’t halt.
So we go into the else branch.
And it halts

Wait shoot.

So, uhh that's a weird program.

What do we do with it?

USE IT TO BREAK STUFF

Does `Diagonal.java` halt when its input is `Diagonal.java`?

Let's assume it does and see what happens…

That didn't work.

Let's assume it doesn't and see what happens…

That didn't work either.

There's no third option. It either halts or it doesn't. And it doesn't do either. That's a contradiction! H.exe can't exist.

So…

So there is no general-purpose algorithm that decides whether any input program (on any input string).

The Halting Problem is undecidable (i.e. uncomputable) there is no algorithm that solves every instance of the problem correctly.

What that does and doesn’t mean

That doesn’t mean that there aren’t algorithms that often get the answer right

For example, if there’s no loops, no recursion, and no method calls, it definitely halts. No problem with that kind of program existing.

This isn’t just a failure of computers – if you think **you** can do this by hand, well…

…you cant either.

Takeaways

Don’t expect that there’s a better IDE/better compiler/better programming language coming that will make it possible to tell if your code is going to hit an infinite loop.

It’s not coming.

More Uncomputable problems

Imagine we gave the following task to 142 students:

Write a program that prints “Hello World”

Can you make an autograder?

Technically…NO!

More Uncomputable problems

Imagine we gave the following task to 142 students:

Write a program that prints “Hello World”

Can you make an autograder?

Technically…NO!

In practice, we declare the program wrong if it runs for 1 minute or so. That’s not right 100% of the time, but it’s good enough for your programming classes.

How Would we prove that?

With a **reduction**

Suppose, for the sake of contradiction, I can solve the HelloWorld problem. (i.e. on input P.java I can tell whether it eventually prints HelloWorld)

Let W.exe solve that problem.

Consider this program…

A Reduction

```
Trick(P,x){
Run P on x, //(but only simulate printing if P prints things)
Print “Hello World”
}
```

This actually prints “hello world” iff P halts on x.

Plug Trick into W and….we solved the Halting Problem!

Reductions in General

The big idea for reductions is “reusing code”

Just like calling a library

But doing it in contrapositive form.

Instead of

“If I have a library, then I can solve a new problem” reductions do the contrapositive:
“If I can solve a problem I know I shouldn’t be able to, then that library function can’t exist”

Fun (Scary?) Fact

Rice’s Theorem

Says any “non-trivial” behavior of programs cannot be computed (in finite time).

What Comes next?

CSE 312 (foundations II)

Fewer proofs ☹

Basics of probability theory (super useful in algorithms, ML, and just everyday life). Fundamental statistics.

CSE 332 (data structures and parallelism)

Data structures, a few fundamental algorithms, parallelism.

Graphs. Graphs everywhere.

Also, induction. [same for 421, 422 the algorithms courses]

CSE 431 (complexity theory)

What can't you do with computers **in a reasonable amount of time.**

Beautiful theorems – more on CFGs, DFAs/NFAs as well.

We’ve Covered A LOT

Propositional Logic.

Boolean logic and circuits.

Boolean algebra. You’ll use quantifiers in 332 to define big-O

Predicates, quantifiers and predicate logic.

Inference rules and formal proofs for propositional and predicate logic.

English proofs.

Set theory. 431 is basically 10 weeks of fun set proofs.

Modular arithmetic.

Prime numbers. Interested in crypto? They’ll come back.

GCD, Euclid's algorithm and modular inverse

No really. A lot

Induction and Strong Induction.

Lots of induction proof [sketches] in 332

Recursively defined functions and sets.

Structural induction.

Regular expressions.

You'll see these in compilers

Context-free grammars and languages.

Relations and composition.

Transitive-reflexive closure.

Graph representation of relations and their closures.

You'll use graphs at least once a week for the rest of your CS career.

Like A lot a lot.

DFAs, NFAs and language recognition.

Cross Product construction for DFAs.

Finite state machines with outputs at states.

Conversion of regular expressions to NFAs.

Powerset construction to convert NFAs to DFAs.

Equivalence of DFAs, NFAs, Regular Expressions

Method to prove languages not accepted by DFAs.

Cardinality, countability and diagonalization

Undecidability: Halting problem and evaluating properties of programs.

Promise you won't ever try to solve the Halting Problem? It's tempting to try to sometimes if you don't remember it's undecidable