Braylee Cumrd

HW06 CS 397

1. Countability

Show that if sets A, B, C are all countably infinite then $A \cup B \cup C$ is also countably infinite. You should be defining a correspondence b/t the natural #'s and $A \cup B \cup C$ and carefully explaining why this works

The union ($\cup$) of sets S and S' is the set of all elements belonging to either S or S'.

$f(x) = \begin{cases} \text{if } x \text{ kth element of A map to } 3K-2 \\ \text{if } x \text{ kth element of B map to } 3K-1 \\ \text{if } x \text{ kth element of C map to } 3K \end{cases}$

$\mathbb{N}$		$A \cup B \cup C$
1	$\xrightarrow{3K-2}$	A
2	$\xrightarrow{3K-1}$	B
3	$\xrightarrow{3K}$	C
$\vdots$		

A correspondence we can make between the natural numbers and $A \cup B \cup C$

A correspondence we can find is to map everything in set A to a value in $\mathbb{N}$ that is in the form $3K-2$, map everything in set B to a value in $\mathbb{N}$ that is in the form $3K-1$, and final map everything in set C to a value in $\mathbb{N}$ that is in the form $3K$. (1st $\mathbb{N}$ mapped to 1st value in A, 2nd $\mathbb{N}$ mapped to 1st value in B, 3rd $\mathbb{N}$ mapped to 1st value in C . . .)

Why this works: every value in $\mathbb{N}$ must be in one of three forms $3K$, $3K-1$, or $3K-2$. We set this up similar to the correspondence w/ $\mathbb{N}$ and $\mathbb{Z}$ but instead of mapping odd #'s to the negative values of $\mathbb{Z}$ we are mapping based on $3K$, $3K-1$, $3K-2$. Since all numbers in $\mathbb{N}$ must follow this pattern, we will have no gaps on the $\mathbb{N}$ side. This makes our correspondence Injective. To prove this correspondence we also need it to be surjective so that the overall correspondence is Bijective or one-to-one. To be surjective all members of $A \cup B \cup C$ needs a matching value in $\mathbb{N}$. $A \cup B \cup C$ members have a match in $\mathbb{N}$ b/c we map $\mathbb{N}$ to exactly one member of $A \cup B \cup C$ based on the $3K$, $3K-1$, $3K-2$ rules. If we continue mapping to the 1st elements in $A \cup B \cup C$ to their $3K$, $3K-1$, $3K-2$ counterparts we will not skip a value since we can just go down the order of the elements in $A \cup B \cup C$. Thus the correspondence is Bijective

$\mathbb{N}$		$A \cup B \cup C$
1	$\longrightarrow$	A element 1
2	$\longrightarrow$	B element 1
3	$\longrightarrow$	C element 1

Idea: make one of the CFG's in EQ_{CFG} have the language $L(G) = \Sigma^*$ then to solve EQ_{CFG} the other CFG would need to be accepted by ALL_{CFG} for EQ_{CFG} to be accepted

2. Sipser Excercise 5.1

show that EQ_{CFG} is undecidable

$$EQ_{CFG} = \{\langle G, H \rangle \mid G + H \text{ are CFGs} + L(G) = L(H)\}$$

we Know that

$$ALL_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG} + L(G) = \Sigma^*\}$$

is undecidable from Theorem 5.13 (Sipser page 225)

Assume for contradiction that EQ_{CFG} is decidable

Now we need a CFG whose language resembles that of ALL_{CFG} $L(G) = \Sigma^*$. To do this a grammer we will call x For each terminal t we need a rule tS where S is our start variable like in class, we need the ϵ rule and then for each terminal t we also just need a rule t. For example if the grammer x has the alphabet $\{a, b, \phi\}$ our grammer would be defined as $x = S \to a \mid b \mid \emptyset \mid \epsilon \mid aS \mid bS \mid \emptyset S$. The language of grammer x is the same language as the CFG in ALL_{CFG} $L(x) = \Sigma^*$.

For the following proof G will represent a context free grammer

Since we assume EQ_{CFG} is decidable it can had a decider we will call M

We construct a TM R to decide ALL_{CFG} as follows

$R =$ "on input $\langle G \rangle$:

1. Create a TM M that decides EQ_{CFG}
2. Run M on $\langle G, x \rangle$
3. If M accepts $(G = x)$, accept otherwise if M rejects, reject

Contradiction: Since x is a language $L(x) = \Sigma^*$ we would need to be able to decide ALL_{CFG} to run x on the EQ_{CFG} in step 2. The turing machine M is not possible to run on $\langle G, x \rangle$ because x is undecidable by ALL_{CFG}.

Conclusion: Since we hit a contradiction, EQ_{CFG} is undecidable

3. Sipser Exercise 5.4

If $A \leq_m B$ and B is a regular language does this imply that A is a regular language? Why or why not? Explain using reduction

This does not imply that A is also a regular language.

An Example if we let

$A = \{0^n 1^n \mid n \geq 0\}$ ~ example of context free language from PDA lecture

Then A is not a regular language (It is context free)

if we let

$B = \{1^n\}$

Then B is a regular language that accepts when a string is '1' and rejects otherwise

$A \leq_m B$ and B is a regular language but A is not.

the alphabet of each can be the symbols '0' and '1'

we can show $A \leq_m B$:

We can create a function f that takes in a string as input to. A and recognizes these strings.

If function f recognizes strings from A we can have f return a value for when A accepts and a different value when A rejects.

To change this to an instance of B we can have f output the string '1' if f accepts A and '0' if f rejects B.

Thus f would map instances of A that are accepted to an accepted string in B.

f would map instances of A that are rejected by A to the string '0' which would be rejected by B.

Conclusion: We can show that $A \leq_m B$ has at least one example where B is a regular language and A is not. Thus if $A \leq_m B$ and B is a regular language this does not imply that A is a regular language.

$A_{TM} = \{ \langle M, w \rangle \mid M$ is a TM and M accepts $w \}$

E_{TM} proof look at it

↳ follow a similar idea

4. Sipser Problem 5.9

Let $T = \{ \langle M \rangle \mid M$ is a TM that accepts w^R whenever it accepts $w \}$.

Show that T is undecidable

① For contradiction assume T is decidable

② let R be the decider for T

The machine M that will be accepted into A_{TM} should be changed to the following description of M.

let M_1 = "on input x":

① if $x \neq w$ or $x \neq w^R$ reject

② if $x = w$ run M on w (check if w^R accepted)

③ if $x = w^R$ accept $(w = w^R)$

Construct S to decide A_{TM}:

S = "on input $\langle M, w \rangle$:

① use M and w to construct M_1

② run R on $\langle M_1 \rangle$

③ If R accepts accept, reject if R rejects.

Contradiction since S is an instance of A_{TM}, R could not exist or we would be able to solve A_{TM} thus T is undecidable.

The idea was to try something like what we did for E_{TM}. My thoughts were to make the M_1 machine where it (1) rejected if its input was not w or w^R (2) run M on w if $x = w$ because it still had to check if the machine accepted w^R and (3) accept if the input was the w^R because that means $w = w^R$. I could be a little off but that was the idea.

5. Sipser Problem 5.15

Consider the problem of determining wether a Turing machine M on an input w attempts to move its head left at any point during its computation on w.

(1) formulate this into a formal language

$$A = \{\langle M, w\rangle \mid w \text{ is an input}, M \text{ is a Turing machine and } M \text{ moves its head left at least once}\}$$

(2) show it is decidable

$S =$ "on input $\langle M, w\rangle$:

1. let K be the number of states in machine M
2. let l be the length of input w
3. Simulate M on w and allow M to run at least $K+1+l$ steps
   - accept if it goes left in this time
4. reject if you complete this # of steps & the machine never goes left

Idea:

There are K unique states in M

w is the length of l and M will read in at least l characters in order to decide if it will accept w

We add 1 to K (or to the whole # of steps) to make sure that the machine goes far enough to see a non-unique state during its computation (meaning it would have to go left to see a state twice)

* iff show both directions

6. Sipser Problem 5.22
Show that A is recognizable iff $A \leq_m A_{TM}$.

Way 1:

$A \leq_m A_{TM}$ → If A_{TM} is recognizable then A is turing recognizable

if A_{TM} is recognizable then A is recognizable

This is Theorem 5.28 in sipser described on slide 51 in the reducibility lecture
- I'm not sure how to prove this theorem is true but we were given it to use
- In theory to prove this we would do similar to Theorem 5.22 that we used for determining if a Turing machine is decidable
  - Since you can map A to A_{TM} there is at the very least a way to solve A w/ A being turing recognizable b/c you could always transform A into A_{TM} which is recognizable

way 2:

If A is recognizable then A_{TM} is recognizable
- To show this since we are saying A is recognizable then exists a turing machine R that recognizes A.
R can be constructed to accept on input w and accept if w is a part of L(R) and reject otherwise (if w is not in the language of R.
I'm not sure how to formally write it but you could create a TM M that takes in w. Then simulate a machine M_1 on w. M_1 would work like A_{TM} {<M,w> | M is a TM and M accepts w. If M_1 accepted then M could accept else M would reject. This works b/c for w to get to A_{TM} $w \in A$ so if A_{TM} accepts w so can M_1. This maps A_{TM} to A showing the

Statement is true in the opposite direction.

Bonus Problem:

The halting problem refers to our ability or rather lack of ability to build a turing machine that would tell us wether a program halts (accepts/rejects) or loops forever. No turing machine can solve if there is an infinite loop it can just guess if there is one. The halting problem is one of the undecidable problems and thus can be used to help us find other undecidable problems. One example I can think of comes down to compiler warnings. A compiler can't catch accurately all the places in your code where you may have an unused codepath. This is the halting problem in disguise because it is impossible to tell if you have tried all possibilities to reach this code or if a loop or something else is not allowing that codepath to be explored. Warnings from the compiler are often educated guesses on issues like an unreachable codepath or infinite loop. Another problem that the halting problem may have helped us reach as undecidable is the EQ_{TM} where we check to see if two different programs solve the same problems. The halting problem can be applied here b/c the programs are not guaranteed to solve the same problem in a set time + it would be impossible to tell whether one implementation was just taking longer, or was stuck in a loop. A more abstract example would be the Goldbach conjecture. This conjecture states that every counting # greater then 2 is equal to the sum of two prime numbers. To test that this conjecture actually worked for all cases would not be decidable because there would always be another number to test and the program may never find a counter example to halt on. We have no set answer on this conjecture b/c it is undecidable if this trend really holds for all countable numbers or just the ones we have tested.

Another concept that shows its undecidability with the halting problem is the Kolmogrov complexity. The Kolmogorov complexity of a string w denoted K(w) is the length of the shortest program which outputs w given no input. To find the smallest program you would have to try every combination + it would be impossible to tell if a program will ever output w or if it is stuck in a loop.