Assignment #4

Monday, April 4, 2022 2:41 PM

4.4 We build a Turing machine *T* for deciding if the language $A\varepsilon_{CFG}$. We say that for all CFG's labeled by *C:*

- If *C* derives ε, then $T(\langle G \rangle)$ accepts the input
- If *C* does not derive ε, then $T(\langle G \rangle)$ rejects the input

For the decidability of $A\varepsilon_{CFG}$, we convert *C* into a CFG in Chomsky Normal Form, which we will call *G'*. We say that if $S \to \varepsilon$ is the rule of CFG *G'*, where *S* is the start variable, then that means that *G'* derives ε, so *C* also generates ε since $L(G) = L(G')$. As *G'* is in CNF, then that means $S \to \varepsilon$ is the only ε rule in *G'*, and thus $\varepsilon \in L(G')$. If *G'* does not contain the rule $S \to \varepsilon$, then $\varepsilon \notin L(G')$.
We are then able to enumerate the steps of a new Turing Machine *U*:

1. Convert *G* into CFG *G'* in Chomsky Normal Form
2. If *G'* contains the rule $S \to \varepsilon$, then accept
    a. Reject otherwise

Given that we were able to create the above construction, it is clear that $\langle G \rangle \in A\varepsilon_{CFG}$ iff $\langle G, \varepsilon \rangle \in A\varepsilon_{CFG}$. The construction is correct, and $A\varepsilon_{CFG}$ is thus decidable.

4.13 We build a Turing Machine *T* for deciding the language *A* with input string *w*.

1. Check that *w* actually encodes a pair of regular expressions ($\langle R, S \rangle$). Reject if not.
2. Translate *R* and *S* into DFA's *DR* and *DS*, respectively.
3. Build a DFA *DSC* that accepts the language $\overline{L(DS)}$
4. Build a DFA *IN* that is the intersection of *DR* with *DSC*
5. We run the Turing Machine *T* with the input $\langle IN \rangle$ to determine if $L(IN)$ is empty
    a. If *T* accepts $\langle IN \rangle$ (intersection of *DR* and *DSC* is empty), then accept *w*
    b. If *T* rejects $\langle IN \rangle$ (intersection of *DR* and *DSC* is not empty), then reject *w*

We then will show that *T* decides *A* by proving it halts on all inputs.

- If we let *w* be some word, then it has two possibilities: It either represents a pair or regular expressions or not. In the event it does not, then *T* stops and rejects *w*. Otherwise, it assumes *w* represents a pair of regular expressions.
- The DFA's described in steps 2, 3, and 4 can be constructed in a finite time and are possible (regular languages are closed under complementation and intersection)

Thus, we have enumerated all cases and proven that *T* will halt in all of them after some finite time. It is thus proved that *A* can be recognized by some decidable Turing Machine, and is therefore decidable.

4.28 We construct a Turing Machine *T* that decides *C*:

- We construct a DFA *S* that will recognize the regular expression $\Sigma^*\{x\}\Sigma^*$
- We construct a DFA *IN* to represent the context-free language of $L(G) \cap L(S)$
- We utilize a Turing Machine *F* that decides if *L(IN)* is empty or not (decider E_{CFG}). If accept, then we have T reject. If *F* rejects, then accept.

We also know that the intersection of a regular language and context-free language is (at least) a context-free language (Lecture 8). Thus, we can say that *IN* will be a CFG. As *A* represents all strings with *x* as a substring, we can say that if *G* produces some string *y* with substring *x*, then *L(IN)* should be non-empty. We can then conclude that *C* is decidable.

5.14 We will assume for the sake of contradiction that *BB* is actually computable, which then means that there must exist a Turing Machine (we'll call it *T*) that can compute it. Without loss of generality, let *T* be a Turing Machine that, on input 1^n, halts with $1^{BB(n)}$ for each value of *n*. We now build a new Turing Machine *U* that will halt when started

with a blank tape based on *T*. The steps for *U* are as follows:

1. Turing Machine *U* writes *n* 1's on the tape
2. Turing Machine *U* doubles the number of 1's on the tape
3. Turing Machine *U* executes Turing Machine *T* on the input of 1^{2n}

Turing Machine *T* will then halt with *BB(2n)* 1's on the tape if it starts from a blank tape. For implementing *U*, we require at most *n* states, with *a* number of states for steps 2 and 3, where *a* is a constant.

By definition, $BB(n+a)$ is the maximum number of 1's where Turing Machine's with $(n+a)$ states will halt, which is at least the number of 1's that *U* halts with. This would imply then that $BB(2n) \leq BB(n+a)$, which would hold for all *n*. However, we can see that *BB(k)* is a strictly increasing function, meaning that, since $a < n$, then $BB(n+a) < BB(2n)$. This is a contradiction, which proves that our original assumption of *BB* being computable is incorrect. Thus, we have proven that *BB* is not computable.

Bonus We will now prove it grows faster than any computable function. We fix an increasing computable function *f*, which when given a $k \in \mathbb{N}$, a Turing Machine T_k, and a blank tape, while compute the first $f(k)$ and then move around until halting for *b* steps. We can observe that the number of states of T_k is bounded by *b+k*, where *b* is the constant defined earlier. From this, we can see that $f(k) \leq BB(b+k)\ for\ all\ k$. From establishing *f* as an increasing computable function, we define some new function *g(k)* such that $g(k) = f(2k)$. From this, we say that $g(k) \leq BB(c+k)\ for\ all\ k$, where *c* is some constant similar to *b*. Because of this, BB must eventually dominate *f(k)* when *k* becomes larger than *c* ($c \leq k$). We show this as follows: $f(c+k) \leq f(2k) = g(k) \leq BB(c+k)$, and since we chose some arbitrary *f(k)* function, we can say that for all functions , $f(k) \leq BB(k)\ for\ all\ c \leq k$

5.19 We will prove decidability using a generic instance of the problem, which we will show as the following example: $\left\{\left[\frac{x_0}{y_0}\right], \left[\frac{x_1}{y_1}\right], \ldots \left[\frac{x_n}{y_n}\right]\right\}, where\ |x_i| = |y_k|\ \forall\ 1 \leq i \leq k.$ After we find the match of the Silly Post Correspondence Problem (SPCP), we then look to see if the numerator is equal to the denominator for the current "tile", as the lengths of the top and bottom are identical. For each tile, there are only two options: the numerator and denominator are the same, or they are not. This binary choice means that a decision can be made for each tile, and thus the SPCP is easily decidable.

5.20 We propose that there is an i^{th} position in any string generated from $\{1\}^*$. We let *S* be the subset of $\{1\}^*$, where the i^{th} position of any generated string from S would be "1". The number of subsets of *S* is infinite, as there are as many strings as possible binary strings, and therefore uncountable. The uncountable set is not Turing recognizable as there is more subsets than Turing machines. Therefore, we can say that there exists an unrecognizable susbet of $\{1\}^*$, so it will be undecicable as well.

5.21 To prove undecidability, we must note the following:

1. If *P* has a match with $t_{i1}t_{i2} \ldots t_{ik} = b_{i1}b_{i2} \ldots b_{ik}$, then we can also see that the string $t_{i1}t_{i2} \ldots t_{ik}a_{ik}a_{ik-1} \ldots a_{i1}$ has at minimum two different way to derive it: one from *T* and the other from *B*
2. If the *G* is ambiguous, then we have some string *x* should have multiple derivations. As *G* is creating *x*, we can then write *x* as $wa_{j1}a_{j2} \ldots a_{jm}$ some *w* that does not have symbols from $a_i s$.

After checking the grammar *G*, we can observe that the derivation of *B* and *T* each generate, at most, one string of the same form of *s*. The multiple derivations are as follows:

$S \rightarrow T \rightarrow s = t_{jm}t_{jm-1} \ldots a_{j1}a_{j2} \ldots a_{jm}$

$S \rightarrow B \rightarrow s \rightarrow b_{jm}b_{jm-1} \ldots b_{jk}$

Thus, we can say that $t_{jm}t_{jm-1} \ldots t_{j1} = b_{jm}b_{jm-1}b_{j1}$. Through 1 and 2, *P* has a match if and only if *G* is ambigous. We can then say that the reduction from PCP to $AMBIG_{CFG}$ works. Thus, $AMBIG_{CFG}$ is undecidable.

5.31 We construct a Turing Machine M_{input} that takes as input *x* and does the following:

On input <*x*>,

i. If *x=1*, then accept

ii. If *x* is odd, set the new value to $x = 3 + x1$. If *x* is even, then set the new value of $x = \frac{x}{2}$
iii. Go back to the first step

This Turing Machine will loop if *x* never equals 1.

We will then construct a Turing Machine M_{loop} that iterates over all positive integers and looks for a counter-example to the *3x+1* conjecture. Our input that we pass *to H* will be the result of running M_{input} *on x*, as without embedding the output, M_{input} will run forever and never provide a result to M_{loop} .To do this, we define the steps of M_{loop} as follows:

On input <*y*>,

i. Ignore the input
ii. For each natural number, $n = 1,2,3 \ldots$:
iii. Run *H* on $\langle M_{input}, n \rangle$
iv. If *H* rejects, then accept. Otherwise, continue the loop

To solve the problem of M_{loop} possibly never finding a counter example, we will use *H* again to see M_{loop} comes to an answer. We define a new Turing Machine $M_{finalcalc}$ as follows:

On input <*z*> :

i. Ignore the input
ii. Run *H* on $\langle M_{loop}, \varepsilon \rangle$
iii. If *H* accepts, then print "Counter Example exists"

Turing Machine $M_{finalcalc}$ will state the answer to the *3x+1* problem

6.19 In a Turing Machine associated with an oracle, an oracle provides flexibility or a countable property not normally afforded by a non-oracle Turing Machine. If the terminology of a Turing Machine is not associated with the concept of an oracle, then it will be undecidable rather than when decidable relative to the A_{TM}. However, if we say that the PCP is decidable, then an oracle must be involved, meaning the PCP problem must then decidable with respect to A_{TM}.

7.21 A language *L* is NP-Complete if the following are true (Def 7.34):

- *L* is in NP
- Every language *B* is reducible to *A* in polynomial time.

We will first prove that $DOUBLE - SAT \in NP$. We define a Non-Deterministic Turing Machine labeled M_{Double}, which behaves as follows:

On input <ϕ> :

i. Non-deterministically guess two Boolean assignments $b_0 \; and \; b_1$ that are different from each other
ii. If both $b_0 \; and \; b_1$ satisfy ϕ, then accept. Otherwise, reject.

We have thus constructed an NTM to decide *DOUBLE-SAT,* which means that $DOUBLE - SAT \in NP$. We now expand upon the first proof to show that $SAT \leq_p DOUBLE - SAT$. We will define some function $f(\phi)$, which maps an instance of ϕ *of SAT* to an instance of ϕ *of DOUBLE-SAT*. It works as follows:

$f(\phi) = \phi \wedge (x_1 \vee x_2), where \; x_1, x_2 \notin \phi$

This reduction/mapping will take polynomial time. If ϕ is unsatisfiable, then ϕ' is also unsatisfiable, because we have only conducted an addition item. However, if ϕ has satisfying assignments b_0/b_1, then ϕ' has at least three satisfying assigments, which all correspond to the three different ways that one can extend *the b* variables to *the x* variables. Thus, we have proven *that DOUBLE-SAT* is NP-Complete.

7.27 To prove *3COLOR* is NP-Complete, we will first prove that $3COLOR \in NP$. This is easily verifiable in that a color can be found in polynomial time. We will now prove that $\forall \; 3SAT, 3SAT \leq_p 3COLOR$. We will define $3SAT = \{\langle x \rangle \mid x \text{ is a satisfiable 3 Chomsky Normal Form Formula } (3cnf)\}$ and a "3 Chomsky Normal Form" is one where all clauses have three literals. We will let $x = y_1 \wedge y_2 \wedge \cdots \wedge c_n$ be a $3cnf$ over the variables $z_1, z_2, \ldots z_q$. We build a graph G with $2q + 6n + 3$ nodes, contaning a variable gadget for each z_i, one clause gadget for each clause, and one palette gadget defined as follows:

1. The nodes of the palette gadget are labeled $T, F, and \; W$
2. Label the nodes of each variable gadget "+" and "-" such that they cannot reach to the *W* node in

the palette gadget
3. Create a gadget for each clause
4. Connect the *F* and *W* nodes to the top of the clause gadget in the palette
5. Connect the top of the bottom triangles to *the W* node
6. For every clause h_i, connect the j^{th} $\left(1 \leq j \leq 3\right)$ bottom node of the clause gadget to the literal node that appears in its j^{th} location.

To demonstrate that the construction is correct, we will demonstrate that if ϕ is satisfiable, then the graph is 3-colored, with colors *T, F,* and *W*. We first color the palette with the appropriate labels. For each variable, color the "+" node as *T* and "-" node as *F* if the variable is true in satisfying the assignment. If otherwise, reverse the color assignment. We should have a proper 3-coloring.

We are also able to obtain a satisfying assignment if given a 3-coloring by taking the colors assigned to the "+" nodes of each variable. If we do this, we can see that neither node of the variable gadget can be colored using the color *W*, as all variable nodes are connected to *W*. In addition, if both of the bottom nodes of the clause gadget are colored with *F*, then the top node must also be the same color, which also means each clause must contain a true literal. Therefore, $3SAT \leq_p 3COLOR$.

Since we have proven the two conditions of NP-Complete, we can say that $3COLOR \in NP - Complete$