Brandan Smith Nov. 28, 2019

Comp 330: Assignment 4

4.4 For A_{ECFG} to be decidable we need to construct a Touring Machine that accepts or rejects for all inputs in finite time.

M = "On input $\langle G \rangle$ where G is a CFG:
1. Run Touring Machine S (where S is the machine from Theorem 4.7 showing that $A_{CFG} = \{\langle G, w \rangle | G \text{ is a CFG that generates string } w\}$ is a decidable language) on input $\langle G, \varepsilon \rangle$.
2. If S accepts then accept, if S rejects then reject."

4.13 For A to be decidable we need to construct a Touring Machine that accepts or rejects for all inputs in finite time.

M = "On input $\langle S, R \rangle$ where S and R are regular expressions:
1. Noting that $L(R) \subseteq L(S)$ iff $\overline{L(S)} \cap L(R) = \emptyset$, given the equivalence between a regular expression and a DFA, construct DFA E s.t. $L(E) = \overline{L(S)} \cap L(R)$
2. Run Touring Machine S (where S is the machine from theorem 4.4 showing that $E_{DFA} = \{\langle A \rangle | A \text{ is a DFA and } L(A) = \emptyset\}$ is a decidable language) on input $\langle E \rangle$.
3. If S accepts then accept, if S rejects then reject.

4.28 For C to be decidable we need to construct a Touring Machine that accepts or rejects for all inputs in finite time.

M = "On input $\langle G, x \rangle$ where G is a CFG and x is a string of alphabet of G:
1. Noting that x is a substring of $y \in L(G)$ iff $\Sigma^* x \Sigma^* = y$, given that the intersection of a regular language and a context free language is context free, construct $L(NE) = L(\Sigma^* x \Sigma^*) \cap L(G)$.
2. Run TM S (where S is the machine from theorem 4.8 showing that $E_{CFG} = \{\langle G \rangle | G \text{ is a CFG and } L(G) = \emptyset\}$ on input $\langle NE \rangle$
3. If S accepts then reject, if S rejects then accept.

5.16 Given mapping $BB: \mathbb{N} \to \mathbb{N}$ is computable if some Turing Machine M on every input w, halts with just $BB(w)$ on its tape.

Let's assume BB is a computable function. This implies there must exist some TM S that halts for input $1^{BB(k)}$. Which further implies that S has input (resulting from the problem description) of 1^k

M = "On input $\langle k \rangle$ where k is $\mathbb{N}$:
1. Write k 1's to the tape.
2. Run a machine D which doubles each input.
3. Run S on the contents of tape.

This construction implies M halts with $BB(2k)$ 1's on the tape. However when implementing M we took $(k+n)$ steps. k comes from writing k 1's to the tape, and n comes from running both machines from steps 2 and 3. This implies M needs $(k+n)$ steps to halt. Therefore by construction $BB(k+n) \geq BB(2k)$ (the number of states is at least the number of 1's upon halting. But for large k the relation doesn't hold because $2k$ grows faster than $k+c$ with respect to k. $\Rightarrow BB(k+n) < BB(2k)$ for large k. Therefore we have a contradiction and so our assumption cannot hold.
$\Rightarrow$ $BB(k)$ is not computable.

5.19

For the SPCP to have a match the first and last tiles of a sequence of tiles must have the top and bottom strings being identical (i.e. $\frac{t_i}{b_i}$ s.t. $t_i = b_i$ for the first and last of a sequence of tiles). This is because the lengths of top and bottom are the same so if they are not the same strings then there exists no further combinations to make them become the same. (This logic extends to every tile in the match).

M = "On input $\langle P \rangle$ where P is an SPCP tileset:
1. Run Turing Machine S (where S is the machine that checks all tiles in input tileset P, and accepts if any $t_i = b_i$, rejects if end of P is reached) on input P.
2. If S accepts then accept, if S rejects then reject."

5.20 To show undecidability we would like to reduce an undecidable problem to our current problem. Take $\{0,1\}^*$ (undecidable as it describes all possible Natural Numbers which is infinitely defined). For all $x \in \{0,1\}^*$ we can define $bin(x)$ as the binary number expressed by string x and then map $\{0,1\}^*$ to $1^{bin(x)}$. As there exists a reduction from $\{0,1\}^*$ to a subset of 1^* ($1^{bin(x)}$) we have shown that $\exists$ a subset of 1^* that is undecidable.

5.21 So if there exists a match to PCP then $t_{i_1} t_{i_2} \dots t_{i_k} = b_{i_1} b_{i_2} \dots b_{i_k}$.
The CFG has two leftmost derivations that contain the strings:

$S \Rightarrow T \overset{*}{\Rightarrow} t_{i_1} t_{i_2} \dots t_{i_k} a_{i_k} \dots a_{i_2} a_{i_1}$

$S \Rightarrow B \overset{*}{\Rightarrow} b_{i_1} b_{i_2} \dots b_{i_k} a_{i_k} \dots a_{i_2} a_{i_1}$.

Now to show that if CFG is ambiguous then there exists a match to PCP.
There are two ways to generate $\omega a_{i_1} a_{i_2} \dots a_{i_k}$ for any ordering of a's which forces ω to be either all t_i's or all b_i's. However for any given ordering a's the ω is constructed by a specific ordering of t_i's or b_i's such that $\omega = \omega$ for both leftmost derivations so by construction $t_{i_1} t_{i_2} \dots t_{i_k} = b_{i_1} b_{i_2} \dots b_{i_k}$.

$AMBIG_{CFG}$ is undecidable because CFG is ambiguous iff PCP has a match.

IF A_{TM} decidable, then halting is decidable. Assume $H \equiv HALT$.

5.31 Construct TM F such that it takes input $x \in \mathbb{N}$ and loops over the $3x+1$ problem starting from x.

Then we can construct TM M

M = "On input x where $x \in \mathbb{N}$:
1. Run H with input (F, x).
2. If H accepts then $x = x+1$, If H rejects then reject."

3X1 = "On input M, $M \in TM$:
1. Run H with input $(M, 1)$
2. If H accepts then accept, If H rejects then 'reject.'"

3X1 is a Touring Machine that Decides if EVERY $x \geq 1$ reduces down to a single 1 thus 3X1 is decidable iff H exists

6.19 As seen in class, the PCP can be reduced from A_{TM} which implies that PCP is decidable iff A_{TM} is decidable. The construction was such that a set of tiles could fully describe the execution of a Touring Machine step by step. So a Tourng Machine that tells us that A_{TM} is decidable consequently also tells us that PCP is decidable.

Note: This can be extended to Many undecidable if they can be reduced from A_{TM} (which is a lot of problems).

7.21

Showing NP: The exact same as checking 1, a Non-deterministic machine can run on 2 or more inputs and accept if at least 2 of them satisfy ϕ (in linear time too!).

Showing Completeness: SAT $\leq_p$ DOUBLE-SAT

M = "On input ϕ where ϕ is a boolean expression:
1. $\phi' = \phi \wedge (y \vee \bar{y})$ where y is a new variable.
2. Output ϕ' "

Using M above, given input $\phi \in$ SAT, ϕ' outputted will have 2 unique solutions because y = True, and y = False both satisfy the newly added clause.

So ϕ has a solution iff ϕ' has at least 2 solutions.

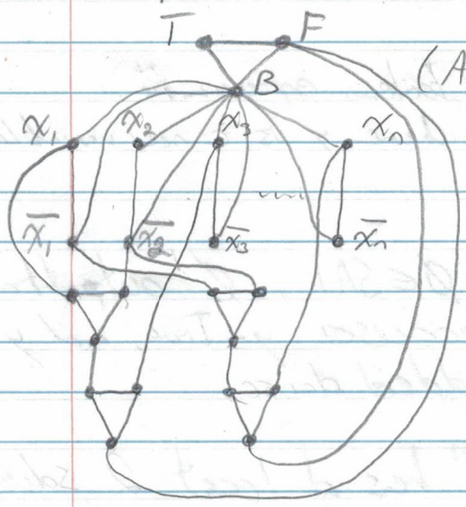
$\Rightarrow$ DOUBLE-SAT is NP-Complete

7.27

Showing NP is trivial by a certifier ✓ checking for each edge that its two connected nodes have a different colour.

Completeness: $3SAT \leq_P 3\text{-}CoL$

Input: 3CNF ϕ with variables $x_1, x_2, \ldots, x_n$



(Assigning B as the third colour)

(The connection to the or-gadgets can be done specifically to whichever 3-SAT we are looking at. For this example I will draw the graph for say $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_n)$)

In words the construction is as follows:

1) Construct the palette coloured as T, F and B.
2) Construct a variable sub-graph for each variable in ϕ.
3) Connect each node of each variable to the B node in the palette.
4) Construct two or-gadgets for each clause in ϕ s.t. the output of one is the input of the other (∇).
5) For each clause in ϕ connect the three corresponding variable nodes x_k or $\bar{x}_k$ to the three inputs of an available augmented or-gadget.
6) Connect the outputs of each augmented or-gadget to the F node in the palette.

If there exists a 3-CoL on the constructed graph, then we equivalently have a solution of 3-SAT.

$\Rightarrow$ 3-CoL is NP-Complete.