**CSCI 1010 - Fall'25 Theory of Computation** Lorenzo De Stefani

# HW7

*Due: n/a*

**Reminder**: Submit your assignment on Gradescope by the due date. Submissions must be typeset. Each page should include work for only one problem (i.e., make a new page/new pages for each problem). See the course syllabus for the late policy.

While collaboration is encouraged, please remember not to take away notes from any labs or collaboration sessions. Your work should be your own. Use of other third-party resources is strictly forbidden.

Please monitor Ed discussion, as we will post clarifications of questions there.

## Problem 1

Prove that the following language is not Turing-recognizable:

$L = \{\langle G_1\rangle, \langle G_2\rangle, \langle G_3\rangle \mid G_1, G_2,$ and $G_3$ are CFG's and $L(G_1) = L(G_2) = L(G_3)\}$.

Hint: You can assume that the following language is undecidable (see page 200 of the textbook):

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

*Solution:* In order to prove that $L$ is not Turing-recognizable, we do a proof by contradiction. Assume that $L$ is Turing-recognizable. We will first show that $\bar{L}$ is Turing-recognizable, which would imply that $L$ is decidable by Theorem 4.11. We will then show that $L$ is undecidable, which contradicts the previous statement.

**Part 1:** We first prove that the language $\bar{L}$, where $\bar{L} = \{w \mid w \text{ does not encode three grammars}\} \cup \{\langle G_1\rangle, \langle G_2\rangle, \langle G_3\rangle \mid G_1, G_2, G_3 \text{ do not generate the same language}\}$, is Turing-recognizable. To do so, we construct a TM $M$ that recognizes $\bar{L}$ as follows:

$M =$ "On input $\langle G_1\rangle, \langle G_2\rangle, \langle G_3\rangle$, where $G_1$, $G_2$, and $G_3$ are CFG's:

1. Check if $G_1$, $G_2$, and $G_3$ are CFG's. If one is not a CFG, accept.
2. Write out strings $w$ in $\Sigma^*$ in lexicographical order (i.e. in order of increasing length, break ties by alphabetical order). For each string $w$:

   (a) Run the decider for $A_{CFG}$ on $\langle G_1, w\rangle$, $\langle G_2, w\rangle$, and $\langle G_3, w\rangle$.
   (b) If the results are different, accept the input.

3. Reject the input.

In this description, $A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ is the language as described on page 198 of the textbook, where its decidability is proved in Theorem 4.7. If the three input grammars generate different languages, then at some point, $M$ will reach a string $w$ that is in some, but not all, of the languages of the three grammars. Here, the results of $A_{CFG}$ will differ. In the case that the grammars do generate the same language, $M$ will be stuck in the loop for step 2, which means that it will never accept the input, even though it may never reach the reject state.

**Part 2:** Now, we must prove that $L$ is undecidable to complete the proof. To do so, we will do a proof-by-contradiction: assume that $L$ is decidable. With the Turing machine $S$ that decides $L$, we will construct a TM $S'$ that decides $EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$ of page 200 in the textbook.

$S'$ = "On input $\langle G, H\rangle$, where $G$ and $H$ are CFGs:

1. Ensure that $G$ and $H$ are CFGs.
2. Run $S$ on $\langle G, G, H\rangle$. If $S$ accepts, accept; otherwise, reject.

Since we assume that $S$ is a decider, it must also be that $S'$ is a decider, a contradiction since the language $EQ_{CFG}$ is undecidable. Thus, it must be that $L$ is undecidable. (Note: A proof of correctness is also necessary here)

**Wrap Up:** We assume that $L$ is Turing-recognizable. With this assumption, we prove that $L$ is decidable, as we show that $\bar{L}$ is also Turing-recognizable in Part 1. However, this contradicts the fact that $L$ is undecidable, which we prove in Part 2. Therefore, it must be that $L$ is actually *not* Turing-recognizable.

## Problem 2

Prove that for any language $A$, $A$ is Turing recognizable if and only if $A \leq_m A_{TM}$.

*Solution:*

- $A$ is Turing-recognizable $\implies A \leq_m A_{TM}$: First, suppose that $A$ is Turing-recognizable and let $M$ be a Turing machine that recognizes $A$. The function $f$ defined by $f(w) =< M, w >$ is a reduction from $A$ to $A_{TM}$ because it is obviously computable and we have that

$$w \in A \iff M \text{accepts } w \iff < M, w > \in A_{TM} \iff f(w) \in A_{TM}$$

- $A \leq_m A_{TM} \implies A$ is Turing-recognizable: We know that $A_{TM}$ is Turing-recognizable, so by Theorem 5.28, $A$ is Turing-recognizable.

## Problem 3

Boolean variables can take on values of TRUE or FALSE. Boolean operators are $\wedge$ (and), $\vee$ (or) and $\neg$ (not). A Boolean formula is an expression with Boolean variables and operators A Boolean formula is satisfiable if some assignment of 0s and 1s to the variables makes the formula evaluate to TRUE. For example, the formula $(\neg x \vee y) \wedge (x \vee \neg z)$ is satisfiable by several assignments including $x = FALSE$, $y = TRUE$, $z = FALSE$.

In the Boolean Satisfiability problem (SAT), given a Boolean formula $\phi$, we aim to decide whether $\phi$ has any satisfying assignments. That is, given the encoding $\langle \phi \rangle$ of the Boolean formula $\psi$ we aim to decide whether $\langle \phi \rangle$ is a member of the language

$$SAT = \{\langle \phi \rangle | \psi \text{ is a satisfiable Boolean formula}\}.$$

As we will see in class, $SAT$ is a very $HARD$ problem. However, we can study variations of the problem that are considerably *easier*.

A *clause* is a type of Boolean formula in which literals (Boolean variables or their negation) are connected by only "or" operators. For example $(x_1 \vee x_2 \vee \neg x_1)$ is a clause. We say that a Boolean formula is in Conjunctive Normal Form (CNF) or clausal normal form if it is a conjunction (and) of one or more clauses. For example $(x_1 \vee x_2 \vee \neg x_1) \wedge (x_1 \vee x_3 \vee \neg x_2) \wedge x_1 \wedge (x_2 \vee \neg x_1)$ is a CNF Boolean formula.

We will consider CNF Boolean Formulas such that each variable appears at most twice (including the times that it appears as negated). For example, in $(x_1 \vee x_2 \vee \neg x_1) \wedge (x_3 \vee x_3 \vee \neg x_2) \wedge x_4 \wedge (x_6 \vee \neg x_7)$ each variable occurs at most twice, while in $(x_1 \vee x_2 \vee \neg x_1) \wedge (x_1 \vee x_3 \vee \neg x_2) \wedge x_1 \wedge (x_2 \vee \neg x_1)$ that is not the case ($x_1$ appears 5 times).

Consider the language. $2RCNFSAT = \{\langle\phi\rangle | \psi$ is a satisfiable CNF Boolean formula in which each variable occurs at most twice $\}$.

Prove that $2RCNFSAT \in P$. That is, provide an algorithm that decides whether an input $\langle\phi\rangle$ is a member of 2RCNFSAT in polynomial time with respect to the size of the input. Here you can assume that the size of the input is the number of literals in $\phi$ (i.e., the sum of occurrences of variables in $\phi$). For example, if $\phi = (x_1 \vee x_2 \vee \neg x_1) \wedge (x_1 \vee x_3 \vee \neg x_2) \wedge x_1 \wedge (x_2 \vee \neg x_1)$, $|\phi| = n = 9$.

Let $\psi$ be a formula with $n$ variables $x_1$, ..., $x_n$. The following describes an algorithm to decide whether $\psi$ is satisfiable or not.

**Solution**

There are multiple ways to approach this problem. Here are two possible solutions.

Solution 1

Consider the following description of a TM, $T$, on input string $\psi$:

1. First we parse the string to check that it is in the correct input format. If not, we reject. If it is, we proceed.
2. Let $x_l$ be a variable in the formula.
3. Parse $\psi$ to see how $x_l$ appears.
   (a) if $x_l$ appears twice as $x_l \vee x_k$ and $\neg x_l \vee x_i$, delete the two clauses and add $x_i \vee x_k$ to the formula. Apply the algorithm to the resulting formula.
   (b) if $x_l$ appears twice as $x_l$ or appears twice as $\neg x_l$, delete the two clauses and apply the algorithm to the resulting formula.
   (c) if $x_l$ appears once, delete the clause it appears in and apply the algorithm to the resulting formula.
4. If the formula has two clauses and is unsatisfiable then reject; if it has two clauses for less and is satisfiable then accept.

We claim that our reduction process preserves satisfiability:

For any $x_i$, since $\psi$ is in CNF and $x_i$ appears at most twice, $\psi$ is of one of the following forms, where $x_l$ does not occur in $\psi'$:

1. $(x_l \vee x_k) \wedge (\neg x_l \vee x_i) \wedge \psi'$
2. $(x_l \vee x_k) \wedge (x_l \vee x_i) \wedge \psi'$ or $(\neg x_l \vee x_k) \wedge (\neg x_l \vee x_i) \wedge \psi'$
3. $(x_l \vee x_i) \wedge \psi'$ or $(\neg x_l \vee x_i) \wedge \psi'$

In each of these cases, our algorithm reduces the formula to a logically equivalent one:

1. $(x_l \vee x_k) \wedge (\neg x_l \vee x_i) \wedge \psi'$ is reduced to $(x_k \vee x_i) \wedge \psi'$. Indeed, if $(x_l \vee x_k) \wedge (\neg x_l \vee x_i) \wedge \psi'$ if and only if $\psi'$ and one of $x_i$ or $x_k$ is true under the given interpretation, which is equivalent to saying $(x_k \vee x_i) \wedge \psi'$ is satisfiable.
2. $(x_l \vee x_k) \wedge (x_l \vee x_i) \wedge \psi'$ or $(\neg x_l \vee x_k) \wedge (\neg x_l \vee x_i) \wedge \psi'$ is reduced to $\psi'$. Indeed, $(x_l \vee x_k) \wedge (x_l \vee x_i) \wedge \psi'$ or $(\neg x_l \vee x_k) \wedge (\neg x_l \vee x_i) \wedge \psi'$ is satisfiable if and only if there is an interpretation under which $\psi'$ and $(x_l \vee x_k) \wedge (x_l \vee x_i)$ or $(\neg x_l \vee x_k) \wedge (\neg x_l \vee x_i)$ are both true. It suffices to set $x_l$ to TRUE or FALSE for the left-hand side to be true, so this holds if and only if there is an interpretation under which $\psi'$ is true if and only if $\psi'$ is satisfiable.
3. $(x_l \vee x_i) \wedge \psi'$ or $(\neg x_l \vee x_i) \wedge \psi'$ are reduced to $\psi'$. Indeed, $(x_l \vee x_i) \wedge \psi'$ or $(\neg x_l \vee x_i) \wedge \psi'$ is satisfiable if and only if there is an interpretation under which $\psi'$ and $(x_l \vee x_k)$ or $(\neg x_l \vee x_k)$ are both true. It suffices to set $x_l$ to TRUE or FALSE for the left-hand side to be true, so this holds if and only if there is an interpretation under which $\psi'$ is true if and only if $\psi'$ is satisfiable.

Moreover, since at each loop, the number of clauses of the formula is reduced by at least one, and the formula has a finite number of clauses, the algorithm must terminate.

Time analysis:

For each $x_i$, the algorithm takes linear time: parsing the input of size $n$, then adding a clause at the end (or not) and deleting one or two clauses, and checking the length of the output.

Since in the worst case we have $n$ distinct $x_i$s. This is repeated $n$ times, so the total time for the reduction is $O(n^2)$

Therefore, this algorithm runs in polynomial time.

Solution 2

Consider the following description of a TM, $T$, on input string $\psi$:

1. First we parse the string to check that it is in the correct input format. If not, we reject. If it is, we proceed.
2. Repeat the following:
   (a) If $\psi$ is empty, accept.
   (b) For each clause in $\psi$, if the clause has at least two literals in it, accept $\psi$.
   (c) Otherwise, find the clause $C_1$ containing only a single literal $x_i$ or $\neg x_i$
   (d) Scan the string to find the clause $C_2$ containing the second instance of variable $x_i$
   (e) If no $C_2$ exists, remove $x_i$ from the boolean expression and go back to (a)
   (f) Otherwise, we will find a $C_2$ other than $C_1$ containing $x_1$. We will call the $x_i$ in $C_1$ $a$ and the $x_i$ in $C_2$ $b$.
   (g) If $a$ and $b$ have the same parity (e.g. $a = x_i, b = x_i$ or $a = \neg x_i, b = \neg x_i$), remove clauses $C_1$ and $C_2$ from the expression and return to (a)
   (h) If $a$ and $b$ have opposite parity and $b$ is the only literal in $C$, reject $\psi$.
   (i) Otherwise, remove $a$ and $b$ from $\psi$ and go to (a)

Proof of correctness:

We first show that if a boolean expression is in the 2RCNFSAT format and each clause contains at least two literals, then the expression is satisfiable.

Consider a 2RCNFSAT expression $\phi$ with $k$ clauses and suppose each clause has at least two literals in it. If it were ever the case that some variable $x$ occurs only once in the expression, we could just set the variable such that the clause containing it is true so t suffices to show the case where all variables occur exactly twice.

We may also ignore any clauses containing the same variable twice because we can always assign one such that it evaluates to true and the remaining clauses will not be affected.

Additionally, it suffices to show the case where each clause contains exactly two literals. This is because removing the additional literals from any clause with more than two literals results in an expression with exactly two literals. If this smaller expression is satisfiable, adding more literals to the or expression will not change that.

Therefore, we need to show that an expression with $k$ clauses, two distinct variables in each clause, and each variable appearing exactly twice is satisfiable. (If you are familiar with graph theory, this is the same as saying that an undirected graph where each vertex has degree 2 is the union of cycles.)

Consider the following greedy assignment: Start with the first clause $C = (x_i/\neg x_i \vee x_j/\neg x_j)$. If $x_i$ is not negated set it to true, otherwise set it to false. Then, go to the other clause $C'$ containing $x_j$ and make the same assignment based on the parity of $x_j$. Repeat until we return to $x_i$. If there are unassigned variables, repeat this process with the clauses containing them. This process is guaranteed to loop back to the original $x_i$ because we know each variable appears exactly twice and in different clauses and there are a finite number of clauses. It also guarantees that each clause has at least one literal that is satisfied. Therefore $\phi$ is satisfiable.

Proof of correctness:

If the string is in the wrong format, we immediately reject so assume $\psi$ is in the 2RCNFSAT format.

In each iteration of the algorithm we perform one of the following:

1. If $\psi$ is empty, the expression is vacuously true so it is correct to accept
2. If all clauses have at least two literals, we have already shown that $\psi$ is satisfiable in the above proof so it is correct for $T$ to accept

here.

3. If $a$ and $b$ are the only variables in their respective clauses and have opposite parity, then $\psi$ contains an expression of the form $x_i \wedge \neg x_i$, which is not satisfiable. Therefore, it is correct to reject here.

4. Otherwise, we perform a reduction that at least one clause. Each of the reductions in the algorithm is the same a reduction described in the first solution. Each of these was shown to preserve satisfiability on the proof of correctness of solution 1.

Since the expression has a finite number of clauses and we always reduce the number of clauses by at least one in each iteration, the algorithm is guaranteed to terminate. In each case where we terminate, we have argued that the $T$ correctly accepts or rejects so $T$ decides 2RCNFSAT.

Time analysis:

In each iteration of the loop we scan through the string at most once in each of (a), (b) and (c) and then remove at most two literals/clauses. Each of these steps require linear time and we perform them at most $n$ times since we remove at least one clause each time and the number of clauses is bounded by the size of the input. Therefore, the algorithm runs in $\mathcal{O}(n^2)$ time, which is polynomial in $n$.