5CCS2FC2: Foundations of Computing II

Tutorial Sheet 3

Solutions

2.5 (i) Show that the language A_{TM} is recursively enumerable by constructing a sound and complete algorithm that recognises all words $\langle M, w\rangle$, where M encodes a TM that accepts w.

(ii) Hence, or otherwise, show that its complement $\overline{\mathsf{A}_{TM}}$ is *not* recursively enumerable.

<u>SOLUTION:</u>

(i) It is enough to simulate the machine encoded by M on the input word w, and return `true` if M accepts w.

```
public static boolean ATM(String M, String w) {

        // Simulate M on input w with a
        // Universal TM
        boolean ans = UTM(M,w);

        if (ans == true) {
                return true;
        } else {
                return false;
        }
}
```

The algorithm may not terminate if M does not terminate on input w, but will always give the correct answer whenever $\langle M, w\rangle \in \mathsf{A}_{TM}$.

(ii) By saw in lectures that a language and its complement cannot both be undecidable *and* recursively enumerable.

If $\overline{\mathsf{A}_{TM}}$ were also recursively enumerable, there would be an algorithm that would always tell us whether $w \in \overline{\mathsf{A}_{TM}}$, *i.e.*, if $w \notin \mathsf{A}_{TM}$.

We could run this algorithm in parallel with the algorithm `ATM` to always decide whether w belonged to A_{TM} or $\overline{\mathsf{A}_{TM}}$, thereby deciding the Accepting Problem.

2.6 *(Tricky!)*

(i) Show that the language $\overline{\mathsf{EQ}_{TM}}$ is not recursively enumerable by reducing A_{TM} to its complement EQ_{TM}. (In other words, that EQ_{TM} is not co-recursively enumerable.)

(ii) Show that the language $\overline{\mathsf{EQ}_{TM}}$ is also not co-recursively enumerable by reducing A_{TM} to $\overline{\mathsf{EQ}_{TM}}$. (In other words, that EQ_{TM} is not recursively enumerable.)

(It follows that $\overline{\mathsf{EQ}_{TM}}$ and EQ_{TM} are 'harder' than any recursively enumerable or co-recursively enumerable problem. There are not even any sound-and-complete algorithms for either problem)

<u>SOLUTION:</u>

(i) First let us understand why a reduction from A_{TM} to EQ_{TM} would show that EQ_{TM} is not co-recursively enumerable.

- We know that A_{TM} is undecidable but recursively enumerable, and so its complement $\overline{\mathsf{A}_{TM}}$ must not be recursively enumerable.
- A reduction from A_{TM} to EQ_{TM} would also establish that $\overline{ATM}$ is reducible to $\overline{EQTM}$, which is to say that $\overline{\mathsf{EQ}_{TM}}$ is at least as hard as $\overline{\mathsf{A}_{TM}}$. Therefore $\overline{EQTM}$ cannot be recursively enumerable.
- By definition, if EQ_{TM} were co-recursively enumerable, then its complement $\overline{\mathsf{EQ}_{TM}}$ would have to be recursively enumerable. Hence, we would have shown that EQ_{TM} is not co-recursively enumerable.

Returning to the question, we want to show that EQ_{TM} is at least as hard as A_{TM} so that any (hypothetical) algorithm for EQ_{TM} could be used as a subroutine to solve A_{TM}.

So, again, suppose that there is some algorithm `EQTM` that takes as an input a pair of encodings $\langle M_1, M_2\rangle$ and returns `true` if and only if $L(M_1) = L(M_2)$.

Given an input pair $\langle M, w\rangle$ we want to construct a pair of Turing Machines $\langle M_1, M_2\rangle$ such that

$$\langle M, w\rangle \in \mathsf{A}_{TM} \qquad \Longleftrightarrow \qquad \langle M_1, M_2\rangle \in \mathsf{EQ}_{TM}$$

which is to say that M accepts w if and only if $L(M_1) = L(M_2)$.

One approach would be to choose M_1 to be any machine that accepts *all* words, so that $L(M_1) = \Sigma^*$ (set of all strings). And then design M_2 so that its language will accept everything if and only if M accepts w.

```
public static boolean ATMc(String M, String w) {

        String M1 = "[code for M_all]";
        String M2 = "[code for M_w]";

        return EQTM(M1,M2);
}
```

```
public static boolean M_all(String s) {
        // Always accept the input s
        return true;
}
```

```
public static boolean M_w(String s) {

        // Simulate M on hard-coded
        // input w (ignore input s)
        boolean ans = UTM(M,w);

        // Accept s if and only if
        // M accepts w
        if (ans == true) {
                return true;
        } else {
                return false;
        }

}
```

(ii) The reasoning for why reducing A_{TM} to $\overline{\mathsf{EQ}_{TM}}$ shows that EQ_{TM} cannot be recursively enumerable is the same as above.

To answer the question, we want to show that $\overline{\mathsf{EQ}_{TM}}$ is at least as hard as A_{TM} so that any (hypothetical) algorithm for $\overline{\mathsf{EQ}_{TM}}$ could be used as a subroutine to solve A_{TM}.

So, suppose that there is some algorithm `coEQTM` that takes as an input a pair of encodings $\langle M_1, M_2 \rangle$ and returns `true` if and only if $L(M_1) \neq L(M_2)$.

Given an input pair $\langle M, w \rangle$ we want to construct a pair of Turing Machine encodings $\langle M_1, M_2 \rangle$ such that

$$\langle M, w \rangle \in \mathsf{A}_{TM} \qquad \Longleftrightarrow \qquad \langle M_1, M_2 \rangle \in \overline{\mathsf{EQ}_{TM}}$$

which is to say that M accepts w if and only if $L(M_1) \neq L(M_2)$.

Following the same approach as above, we could choose M_1 to be any machine that accepts *all* words, so that $L(M_1) = \Sigma^*$. But we would then need to design M_2 so that its language will accept everything if and only if M does *not* accept w, which includes the case where M does not terminate on w. This would require us to *also* have a subroutine that answered the halting problem.

An alternative would be to choose M_1 to be any machine that rejects *all* words, so that $L(M_1) = \emptyset$. We can then choose M_2 as before, since this have a non-empty language if and only if M accepts w.

```
public static boolean ATM(String M, String w) {

        String M1 = "[code for M_empty]";
        String M2 = "[code for M_w]";

        return coEQTM(M1,M2);
}
```

```
public static boolean M_empty(String s) {
        // Always reject the input s
        return false;

}
```

3.1 Determine whether the following are true or false?

3.1 $10^{15}n \in O(n)$,

3.2 $n^2 \in O(n)$,

3.3 $n^2 \in O(n \log n)$,

3.4 $n^2 \in O(n \log^2 n)$,

3.5 $n \log n \in O(n^2)$,

3.6 $2^{(2n+1)} \in O(4^n)$,

3.7 $n^{\log \log n} \in O(n^{10})$.

SOLUTION:

3.1 $10^{15}n \in O(n)$ — True, since we can choose $k = 10^{15} + 1$, so that $10^{15}n < (10^{15} + 1)n$ for example.

3.2 $n^2 \in O(n)$ — False, since for any fixed value of k, we have that $n^2 > k \cdot n$ for sufficiently large n.

3.3 $n^2 \in O(n \log n)$ — False, since for any fixed value of k, we have that $n > k \cdot \log n$ for sufficiently large n.

3.4 $n^2 \in O(n \log^2 n)$ — False, since for any fixed value of k, we have that $n > k \cdot \log^2 n$ for sufficiently large n.

3.5 $n \log n \in O(n^2)$, — True, since we can choose $k = 1$, so that $n \log n < k \cdot n^2$, for example.

3.6 $2^{(2n+1)} \in O(4^n)$ — True, since we can rewrite $2^{(2n+1)} = 4^n \times 2^1$ and choose $k = 3$, for example.

3.7 $n^{\log \log n} \in O(n^{10})$ — False, since for any fixed value of k, there is eventually some n such that $\log \log n > 10 + k$ (for $n > 2^{1024(2^k)}$), at which point we have that $n^{\log \log n} > n^{10+k} = n^{10} \times n^k > k \cdot n^{10}$ (since $n^k > k$).

3.2 For each of the following formulas F construct a graph G_F and choose an integer k such that

$$F \text{ is satisfiable} \quad \Longleftrightarrow \quad G_F \text{ contains a clique of size } k$$

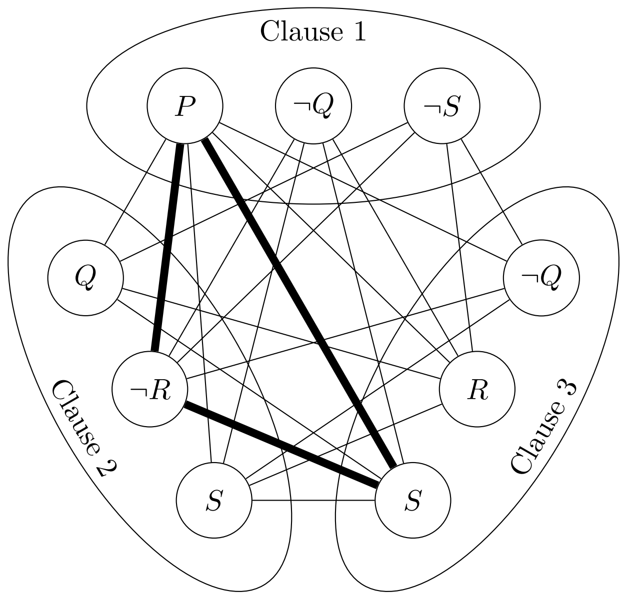
(i) $F = (P \vee \neg Q \vee \neg S) \wedge (Q \vee \neg R \vee S) \wedge (\neg Q \vee R \vee S)$

(ii) $F = (P \vee \neg Q \vee \neg R) \wedge (P \vee Q \vee \neg R) \wedge (\neg P \vee \neg Q \vee R)$

(iii) $F = (P \vee Q \vee \neg R) \wedge (P \vee Q \vee R) \wedge (\neg P \vee Q \vee \neg R) \wedge (P \vee \neg Q \vee R)$

Use this property to identify which of the above formulas are satisfiable.
SOLUTION:

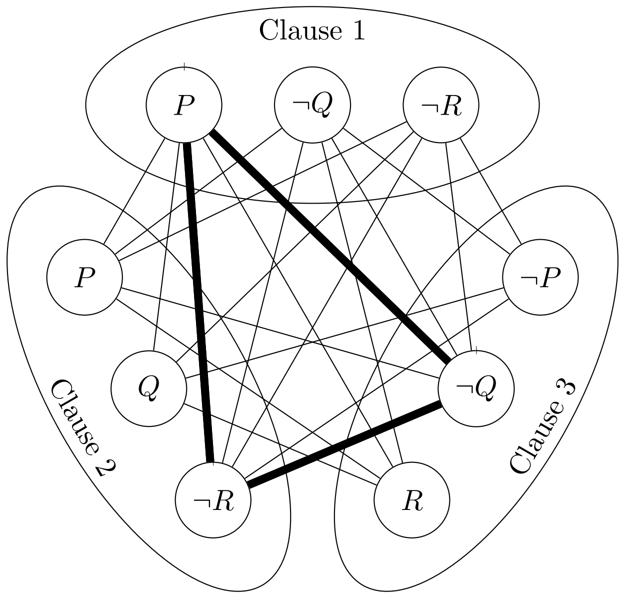
(i) Following the procedure described in class, we have that



There are many 3-cliques in the graph, the one identified in the above diagram corresponds to the following satisfing assignment:

$$\boxed{P = 1, \qquad Q = 1 \text{ or } 0, \qquad R = 0, \quad \text{and} \quad S = 1}$$

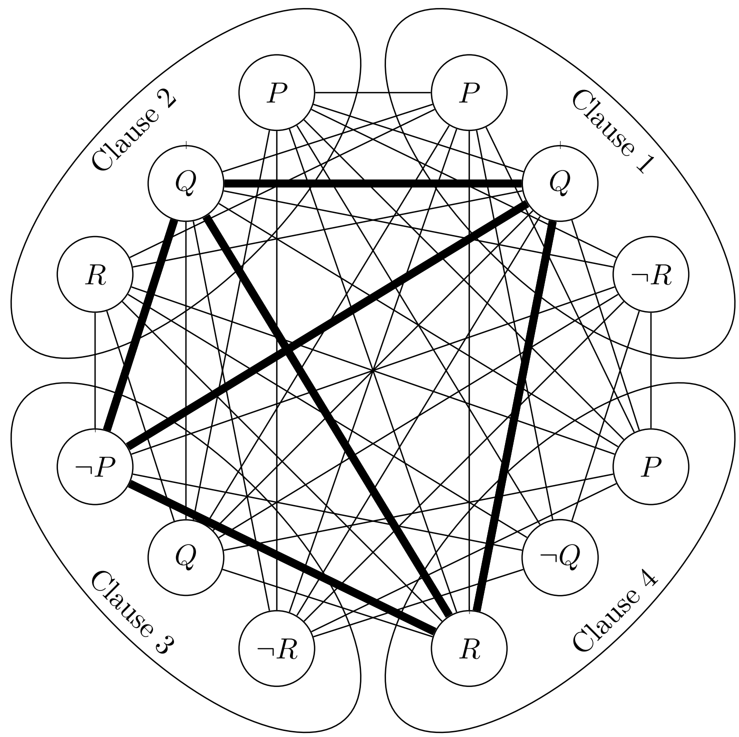
(ii) The graph G_F is illustrated below:



The 3-clique highlighted above corresponds to the following satisfying assignment:

$$\boxed{P = 1, \qquad Q = 0, \quad \text{and} \quad R = 0}$$

(iii) The graph G_F is illustrated below:



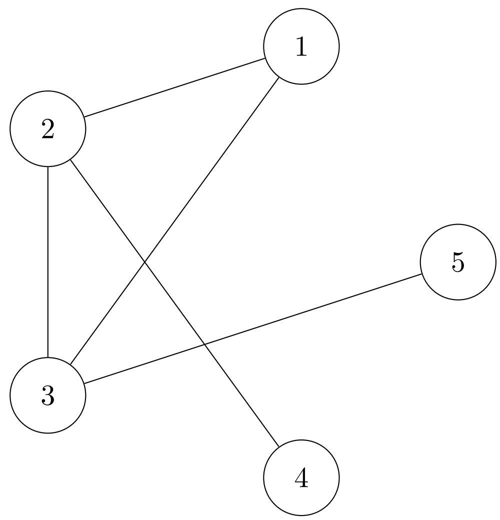
There are several 3-cliques in this graph, but these *do not* correspond to satisfying assignments for the formula F, since they do not select a vertex from each clause.

Hence, we must choose $k = 4$ to guarentee that any if the graph has a clique of size k then the formula is satisfying. One such 4-clique is shown above. This corresponds to the following satisfying assignment,

$$\boxed{P = 0, \qquad Q = 1, \quad \text{and} \quad R = 1}$$

3.3 Construct a propositional formula that is satisfiable if and only if the following graph $G = (V, E)$ can be coloured using only two colours, where

$$V = \{1, 2, 3, 4, 5\}$$
$$E = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 5)\}$$



SOLUTION: Suppose for instance that the two colours are $C = \{R, B\}$. We first need to specify that every vertex is coloured in one of two colours, which we can do with the following formula:

$$(P_{1,R} \vee P_{1,B}) \wedge (P_{2,R} \vee P_{2,B}) \wedge (P_{3,R} \vee P_{3,B}) \wedge (P_{4,R} \vee P_{4,B}) \wedge (P_{5,R} \vee P_{5,B})$$

Each clause says that "vertex i must either be coloured Red or Blue".

Furthermore, we need to also specify that no vertex is to be coloured in more than one colour:

$$\neg(P_{1,R} \wedge P_{1,B}) \wedge \neg(P_{2,R} \wedge P_{2,B}) \wedge \neg(P_{3,R} \wedge P_{3,B}) \wedge \neg(P_{4,R} \wedge P_{4,B}) \wedge \neg(P_{5,R} \wedge P_{5,B})$$

Each clause says that "it is not the case that vertex i is coloured *both* Red and Blue".

Finally, we also need to specify that each edge connects two vertices of different colour:

$$\neg(P_{1,R} \wedge P_{2,R}) \wedge \neg(P_{1,B} \wedge P_{2,B}) \wedge \neg(P_{1,R} \wedge P_{3,R}) \wedge \neg(P_{1,B} \wedge P_{3,B})$$
$$\wedge \neg(P_{2,R} \wedge P_{3,R}) \wedge \neg(P_{2,B} \wedge P_{3,B}) \wedge \neg(P_{2,R} \wedge P_{4,R}) \wedge \neg(P_{2,B} \wedge P_{4,B})$$
$$\wedge \neg(P_{3,R} \wedge P_{5,R}) \wedge \neg(P_{3,B} \wedge P_{5,B})$$

The first clause says that "it is not the case that vertex 1 and vertex 2 are both coloured Red", for example.

This set of formulas is satisfiable if and only if there is a 2-colouring of the graph G. However, since the graph contains a cycle of length

3 ($1 \to 2 \to 3 \to 1$), we must not be able to colour the graph with only 2 colours. Hence, the set of formulas we have derived must not be satisfiable.