CS3230: Design and Analysis of Algorithms (Fall 2014)

Tutorial Set #7

[For discussion during Week 9]

S-Problems are due (outside Prof. Leong's office): Friday, 10-Oct, before noon.

OUT: 29-Sep-2014 **Tutorials:** Tue & Wed, 14, 15 Oct 2014

IMPORTANT: Read "Remarks about Homework".

Submit solutions to S-Problem(s) by deadline given above.
Prepare your answers to all the D-Problems in every tutorial set.

When preparing to present your answers,

- Think of a CLEAR EXPLANATION
- Illustrate with a good worked example;
- Describe the main ideas,
- Can you sketch why the solution works;
- Give analysis of running time, if appropriate
- Can you think of other (perhaps simpler) solutions?

Helpful Hints Series: Make definitions very clear. Read the questions slowly and carefully.

Please assume everywhere below that $\leq_P$ represents polynomial time Karp reduction (unless otherwise specified).
Please recall that decision problem are also known as languages.
Please recall that $\vee$ represents logical OR, $\wedge$ represents logical AND and $\neg$ represents logical NOT operation.

Routine Practice Problems *-- do not turn these in -- but make sure you know how to do them.*

R1. Can a decision problem A in P be NP-complete if P not equal to NP?

Answer. We have seen the following theorem in lectures.
Theorem. Suppose Y is an NP-complete problem. Then Y is solvable in poly-time iff P = NP.
This implies that the answer to the above question is No.

R2. NP stands for [No-problem/Non-polynomial/Non-deterministic-polynomial time]?

Answer. Non-deterministic-polynomial time.

R3. Do we know this for sure: A polynomial time solution does not exist for CIRCUIT-SAT?

Answer. No, we do not know this for sure. If a polynomial time algorithm exists for CIRCUIT-SAT (which we have seen to be an NP-complete problem) then P=NP. This is widely believed to be unlikely, however it is still possible. Whether P=NP, is an open question (a million dollar question, literally ☺, and a great challenge for you guys to settle ☺).

R4. Suppose $A \leq_P B$, what are the implications?

1) If B is solvable in polynomial time, then A is solvable in polynomial time?
2) If A is not solvable in polynomial time, then B is not solvable in polynomial time?
3) If B is not solvable in polynomial time, then A is not solvable in polynomial time?
4) If A is solvable in polynomial time, then B is solvable in polynomial time?

Answer. 1) and 2). This is true both for Karp and Cook reductions.
2) is just the contrapositive of 1) and hence the same statement.

Assume $A \leq_P B$ via a Karp reduction. This means there exists a polynomial time algorithm (transformation) R (below R(x) represents the output of R on input x) such that

$$x \in A \iff R(x) \in B .$$

Assume B is solvable in polynomial time using algorithm Alg-B. A polynomial time algorithm Alg-A to decide A is as follows:

```
Alg-A(x) {
        Set y = R(x)
        If B(y) = true
                Return true
        Elseif B(y) = false
                Return false
}
```

Since R and B are polynomial time algorithms, Alg-A is also a polynomial time algorithm.

Now assume $A \leq_P B$ via a Cook reduction. This means, by definition of Cook reduction, that there exists a polynomial time algorithm R, which makes oracle calls B(y) (returning true or false) for various strings y during its computation. Each oracle call B(y) is counted as a unit time. Assume worst case running time of R on input x is bounded by p(|x|) (where p is a fixed polynomial).

Assume B is decidable using algorithm Alg-B in time q(|y|) for input y (where q is a fixed polynomial). Replace oracle calls B(y) (in R) by Alg-B(y) to get new algorithm R'. Note that since running time of R, on input x, is bounded by p(|x|), the length of any string y for which an oracle call B(y) is made in R, is also bounded by p(|x|) (R does not even have time to write larger strings and hence cannot make oracle calls for larger strings). Also note that R can make at most p(|x|) oracle calls (since each call is counted as unit time). Hence the worst case running time of R' on input x will be bounded by p(|x|) + p(|x|) q(p(|x|)) , which is another polynomial in |x| (since addition, multiplication and composition of polynomials gives another polynomial) .

S-Problems: (To do and submit by due date given in page 1)
Solve this S-problem(s) and submit for grading.

IMPORTANT: Write your NAME, Matric No, Tutorial Group in your Answer Sheet.

S1. [Understanding transformations]

An independent set in a graph G = (V,E) (V is the set of vertices and E is the set of edges of G) is a subset $I \subseteq V$ such that for all $u, v \in I: (u, v) \notin E$.

A clique in a graph G = (V,E) is a subset $J \subseteq V$ such that for all $u, v \in J: (u, v) \in E$.

Given a graph G and a number k, transform it to a graph H such that G has an independent set of size at least k if and only if H has a clique of size at least k.

Answer. Let $H = (V', E')$ where $V' = V$ and $E' = \bar{E}$ (the complement of E). That is for all $u, v \in V: (u, v) \in E \iff (u, v) \notin E'$. It is clear that independent sets in G become cliques in H. Hence G has an independent set of size at least k if and only if H has a clique of size at least k.

We can note that this transformation from G to H can be done in time that is polynomial in the size of input (say a binary encoding of G using an adjacency table which can be done using $\Theta(n^2)$ bits.)

D-Problems: Solve these D-problems and prepare to discuss them in tutorial class. You may be called upon to present your solution *or your best attempt at a solution*. Your solution presentation does NOT need to be fully correct, given your best attempt. The TA will help clarify and correct any issues or errors.

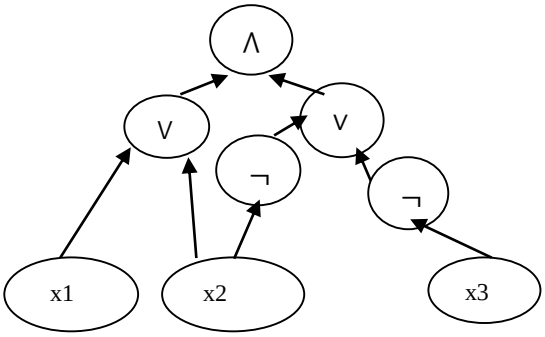
D1. [Transformations]

a) Transform the CNF formula D = $(x1 \vee x2) \wedge (\overline{x3} \vee \overline{x2})$ to a circuit C with AND/OR/NOT gates such that C is satisfiable if and only if (iff) D is satisfiable.

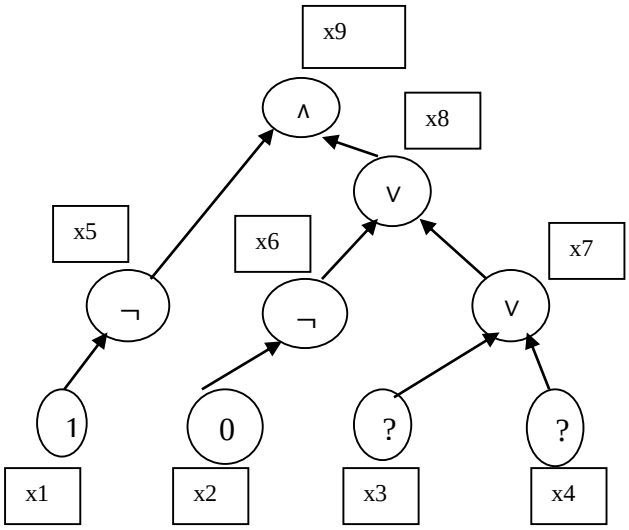
Answer. Many students have given an answer to this question by first explicitly calculating whether D is satisfiable or not. This answer is correct and this could be done since D is a small formula. However please note that this will not be possible when we are given a large formula (say using a million variables and clauses). For that we will need a transformation which does not require to first decide if the given formula is satisfiable or not.

Such a transformation and the resulting circuit C is as follows. Note that here we even additionally satisfy a stronger requirement (which is not asked in the question), which is that the truth table of the circuit C is the same the truth table of the formula D (and of course they have the same set of input variables).

Please note that this example is also helpful for you to solve Programming Assignment 2, Parts A/B/C.



b) Transform the following circuit C to a CNF formula D such that D is satisfiable iff C is satisfiable.



Answer. Again many students have given an answer to this question by first explicitly calculating whether C is satisfiable or not. This answer is correct and this could be done since C is a small circuit. However please note that this will not be possible when we are given a large circuit (say using a million gates and variables). For that we will need a transformation which does not require to first decide if the given circuit is satisfiable or not.

Such a transformation and the resulting formula D is as follows.

Please note that this example is also helpful for you to solve Programming Assignment 2, Parts D/E (which will come later).

The idea is to allot a new variable to each node in the circuit and then encode the gates and constants in the circuit using the following formula. First we will give a formula D' which is not a CNF formula and will later transform it to a CNF formula D.

D' =
$(x1 = 1)$
$\wedge (x2 = 0)$
$\wedge (x5 = \overline{x1})$
$\wedge\ (x6 = \overline{x2})$
$\wedge (x7 = \ x3 \vee\ x4\ )$
$\wedge (x8\ = x6 \vee\ x7)$
$\wedge (x9 = \ x5\ \wedge x8)$
$\wedge (x9 = 1)$

It is easily seen from our construction that D' is satisfiable iff circuit C is satisfiable since formula D' essentially encodes the constraints specified by the gates and constants of the circuit C.

We have seen in the lecture notes that:

1. $(a = 1)$ is equivalent to (a)
2. $(a = 0)$ is equivalent to $(\bar{a})$
3. $(a = \neg b)$ is equivalent to $(a \vee b) \wedge (\bar{a} \vee \bar{b})$
4. $(a = b \vee c)$ is equivalent to $(a \vee \bar{b}) \wedge (a \vee \bar{c}) \wedge (\bar{a} \vee b \vee c)$
5. $(a = b \wedge c)$ is equivalent to $(\bar{a} \vee b) \wedge (\bar{a} \vee c) \wedge (a \vee \bar{b} \vee \bar{c})$

Using these rules we get finally get our CNF formula D (from D'):

$D = (x1)$
$\wedge (\overline{x2})$
$\wedge (x5 \vee x1) \wedge (\overline{x5} \vee \overline{x1})$
$\wedge\ (x6 \vee x2) \wedge (\overline{x6} \vee \overline{x2})$
$\wedge (x7 \vee \overline{x3}) \wedge (x7 \vee \overline{x4}) \wedge (\overline{x7} \vee x3 \vee x4)$
$\wedge (x8 \vee \overline{x6}) \wedge (x8 \vee \overline{x7}) \wedge (\overline{x8} \vee x6 \vee x7)$
$\wedge\ (\overline{x9} \vee\ x8) \wedge (\overline{x9} \vee x5) \wedge (x9 \vee \overline{x5} \vee \overline{x8})$
$\wedge (x9)$

c) Transform a CNF formula C, in which each clause has at most 3 literals to a 3-CNF formula D in which each clause has exactly three literals, with the condition that D is satisfiable if and only if C is satisfiable.

Answer: Assume there is a clause with two literals e.g. $(x1 \vee \overline{x2})$. Introduce a fresh variable say, $x3$. Replace this clause with $(x1 \vee \overline{x2} \vee x3) \wedge (x1 \vee \overline{x2} \vee \overline{x3})$.

We can note that any truth assignment that makes $(x1 \vee \overline{x2} \vee x3) \wedge (x1 \vee \overline{x2} \vee \overline{x3})$ true, also makes $(x1 \vee \overline{x2})$ true. Similarly any truth assignment that makes $(x1 \vee \overline{x2} \vee x3) \wedge (x1 \vee \overline{x2} \vee \overline{x3})$ false also makes $(x1 \vee \overline{x2})$ false.

Now assume there is a clause with just one literal e.g. $(x4)$. Introduce a new variable say x5. Replace this clause with $(x4 \vee x5) \wedge (x4 \vee \overline{x5})$.

Again we can note that any truth assignment that makes $(x4 \vee x5) \wedge (x4 \vee \overline{x5})$ true, also makes $(x4)$ true. Similarly any truth assignment that makes $(x4 \vee x5) \wedge (x4 \vee \overline{x5})$ false also makes $(x4)$ false.

This produces two new clauses with two variables each. Now we can repeat as we do in the first paragraph for these two clauses (using fresh new variables).

We do the same with every clause with less than three variables (every time using fresh new variables).

D2. [Transitivity of reductions]

Show that if $X \leq_P Y$ and $Y \leq_P Z$ then $X \leq_P Z$, where $\leq_P$ represents Karp reduction.

Answer: Let R be a polynomial time Karp reduction from X to Y. Let the worst case running time of R on input x be bounded by p(|x|), where p is a fixed polynomial. From definition of Karp reduction we have,

$$(x \in X) \Leftrightarrow (R(x) \in Y) .$$

Let T be a polynomial time Karp reduction from Y to Z. Let the worst case running time of T on input y be bounded by q(|y|), where q is a fixed polynomial. From definition of Karp reduction we have,

$$(y \in Y) \Leftrightarrow (T(y) \in Z) .$$

This implies

$$(x \in X) \Leftrightarrow (R(x) \in Y) \Leftrightarrow (T(R(x)) \in Z) .$$

Hence $T(R(.))$ (T composed with R) serves as a Karp reduction from X to Z. The worst case running time of this reduction on input x is bounded by $q(p(|x|))$ which is a polynomial in $|x|$ (since composition of two polynomials is a polynomial). Note that the length of R(x) is bounded by p(|x|) since the running time of R(x) is bounded by p(|x|) and hence in this time it cannot output a string of larger length.

D3. [Closure properties of P]

Let $A, B \in P$ be two decision problems where P represents the class of all decision problems that are solvable in polynomial time. Show that

1) $A \cup B \in P$,
2) $A \cap B \in P$,
3) $\overline{A} \in P$.

Answer. Let Alg-A be polynomial time algorithm for *A*. Let Alg-B be polynomial time algorithm for *B*.

The following algorithm R is a polynomial time algorithm for $A \cup B$.

```
R(x) {
        if Alg-A(x) = true
                return true
        elseif Alg-B(x) = true
                return true
        else
                return false }
```

It is easily seen that R(x) is true iff $(x \in A$ or $x \in B)$. Also it is easily seen that since Alg-A and Alg-B are polynomial time algorithms, R is a polynomial time algorithm.

The following algorithm T is a polynomial time algorithm for $A \cap B$.

```
T(x) {
        if Alg-A(x) = true and Alg-B(x) =true
                return true
        else
                return false }
```

It is easily seen that T(x) is true iff $(x \in A \ and \ x \in B)$. Also it is easily seen that since Alg-A and Alg-B are polynomial time algorithms, T is a polynomial time algorithm.

The following algorithm S is a polynomial time algorithm for $\bar{A}$.

```
S(x) {
        if Alg-A(x) = true
                return false
        else
                return true }
```

It is easily seen that S(x) is true iff $(x \notin A)$. Also it is easily seen that since Alg-A is a polynomial time algorithm, S is a polynomial time algorithm.

D4. [Cook v/s Karp reductions]

Show that every decision problem (language) L has a Cook reduction to $\bar{L}$ (the complement of L). Can you show the same with Karp reduction (assume neither L nor $\bar{L}$ is empty)?

Answer. The following algorithm R is the desired Cook reduction from L to $\bar{L}$.

```
R(x) {
        if L̄(x) = true
                return false
        else
                return true
}
```

It is easily seen that R(x) is true iff $x \notin \bar{L}$, that is iff $(x \in L)$. There is only one invocation of the oracle for $\bar{L}$ (which is counted as unit time) and hence R is a polynomial time algorithm.

Doing the same is not easy with Karp reductions. It is actually an important open question and a nice challenge for you guys ☺.