CS5311 Homework #5 Solutions (Spring 2002)

Due: Thursday, May 2, 2002

100 points

1. This problem is about the Reachability problem: Given a directed graph $G = (V, E)$ and two vertices a and b, is there a directed path from a to b?

   (a) [**5 points**] Define **NL**-complete using log-space reduction.

   Answer: A language L is **NL**-complete if and only if L is in **NL** and every language $Q \in$ **NL** reduces to L using no more than $O(\log_2 |x|)$ space, where $x \in Q$. ∎

   (b) [**20 points**] Show that Reachability is in **NL**.

   Answer: Let $G = (V, E)$ be the given directed graph with start vertex $a \in V$ and end vertex $b \in V$. What we need to do is to construct a non-deterministic Turning machine that returns "yes" if there is a direct path from a to b. Here is an algorithm that implements the non-deterministic Turing machine:

   Start := a;
   for $i := 1$ **to** $|V|$ **do**
       begin
           non-deterministically guess the next vertex *End* on the path;
           if (*Start* $\rightarrow$ *End* is an edge) **then**
               if *End* $= b$ **then**
                   a path is found, says "yes" and stops;
               else
                   Start := *End*;
               end if
           else
               no such path exists, says "no" and stops;
           end if
       end
   no such path exists, says "no" and stops;

   Let us analyze the space consumption. There are six variables (*i.e.*, *Start*, *End*, a, b, i and $|V|$) that must be written on a work tape. Since the maximum value of each variable is $|V|$, the number of bits for each variable is $\log_2 |V|$, and the total memory consumption is $O(\log_2 |V|)$. Thus, the above algorithm has a log-space bound and Reachability is in **NL**. ∎

   (c) [**20 points**] Show that Reachability is **NL**-complete.

   Answer: We shall show that any language $Q \in$ **NL** is log-space reducible to Reachability. Hence, Reachability is **NL**-complete. Since Q is in **NL**, there is a non-deterministic Turing machine $\mathcal{M}$ that accepts $x \in L$ with a space bound $O(\log_2 |x|)$. From $\mathcal{M}$ and input x we will construct a directed graph $G = (V, E)$ with start vertex a and end vertex b such that $x \in L$ if and only if there is a directed path from a to b (*i.e.*, Reachability). Note that this construction must have a space bound $O(\log_2 |x|)$.

   The graph is easy to construct. Consider the configuration graph of $\mathcal{M}$ accepting x. We assume $\mathcal{M}$ has only one final state. Otherwise, we can create a dummy one and construct a transition from each existing final state to the new one on a new symbol, which is added to the end of input string. Moreover, we will also assume that only one work tape is used. Otherwise, we can simulate multiple tapes using one tape. In this way, each configuration can be simplified as a triple: (p, h, i), where p is a state, h is the tape head position of the work tape, and i is the tape head position of the input tape. The following is the construction of $G = (V, E)$:

- Each vertex in V corresponds a configuration of $\mathcal{M}$.
- If there is an edge from configuration (p, h, i) to configuration (q, k, j), add a directed arc from the corresponding vertices.
- The start vertex a is the configuration that contains the initial state.
- The end vertex b is the configuration that contains the only final state.

In fact, you can consider the above construction as a renaming process of the configuration graph of $\mathcal{M}$ accepting x. Therefore, the meaning of a configuration graph shows that $\mathcal{M}$ accepts x if and only if there is a directed path from a to b. What remains to do is to show that this construction can be done in log-space. This part is also easy as long as we can generate configurations in a systematic way. The following is a naive way:

```
for i := 1 to |Q| do    // no. of states
    for j := 1 to |x| do    // the work tape of M is less than |x|
        for k := 1 to |x| do    // input length
            begin
                C1 = (i, j, k);   // the 1st configuration
                for p := 1 to |Q| do
                    for q := 1 to |x| do
                        for r := 1 to |x| do
                            begin
                                C2 = (p, q, r);   // the 2nd configuration
                                if C1 ⊢ C2 then
                                    set the entry of C1 and C2 of the adjacency matrix to 1
                                else
                                    set the entry of C1 and C2 of the adjacency matrix to 0
                                end if
                            end
            end
```

In the above, the first set of three **for** loops fix the generation of the adjacency matrix to one row and the three inner **for** loops generate the entries of that row. For each pair of configurations C_1 and C_2 found so far, we determine if the Turing machine $\mathcal{M}$ can go from C_1 to C_2, and write 1 or 0 to the output tape. Thus, at the end of the above code, the output tape contains the adjacency matrix of the configuration graph of Turing machine $\mathcal{M}$ accepting x. If the condition $C_1 \vdash C_2$ can be implemented in log-space, the above can run in log-space because it has a constant number of variables.

Determining if $C_1 \vdash C_2$ is true is easy. Consider $C_1 = (i, j, k)$ and we are in state i and input tape head position k. We can re-scan the input tape and find the content in tape square k. From i, k and the Turing machine instruction, we can determine if C_1 can reach $C_2 = (p, q, r)$. Since this is another set of loops cycling through states and tape heads, it can be implemented in log-space.

Finally, we can use three **for** loops to find and write the start vertex (*i.e.*, initial configuration) and another set to find and write the end vertex (*i.e.*, final configuration). Alternatively, we can use two more work tapes. In the outer **for** loops, if $C_1 = (i, j, k)$ is an initial (*resp.*, final) configuration, it is written to the first (*resp.*, second) work tape. After the adjacency matrix is completely written to the output, we copy the start and end vertices to the output tape. The space consumption is $O(\log_2 |x|)$. Consequently, the above program reduces a Turing machine $\mathcal{M}$ accepting input string x to an instance of Reachability problem using log-space, and Reachability is **NL**-complete. ∎

2. **[10 points]** Show that if a **P**-complete problem can be solved in log-space, then all problems in **P** can be solved in log-space.

   Answer: Let L be a **P**-complete language and let Q be a language in **P**. Then, there is a Turing machine $\mathcal{M}$ that reduces Q to L in log-space. If L can be solved in log-space, then there is a Turing machine $\mathcal{N}$ that accepts all strings in L in log-space. The composition of $\mathcal{M}$ and $\mathcal{N}$ can be done in log-space using the proof that the composition of two log-space reductions is still log-space. We proved this fact in class. Therefore, Q can be reduced to L in log-space and solve that instance of L in log-space. The total space used is still log-space. As a result, all languages in **P** can be solved in log-space. ∎

3. **[10 points]** Show that if **PSPACE-complete** ∩ **NP-complete** $\neq \emptyset$ holds, then we have **PSPACE** = **NP** and **PSPACE-complete** = **NP-complete**. Here, **PSPACE**, **PSPACE-complete**, **NP** and **NP-complete** are the classes of **PSPACE**, **PSPACE**-complete, **NP** and **NP**-complete languages.

   Answer: Let L be a language in **PSPACE-complete** ∩ **NP-complete**. Since we know that **NP** ⊆ **PSPACE**, to prove **NP** = **PSPACE** we only need to establish **PSPACE** ⊆ **NP**. Note that since $L \in$ **PSPACE-complete** ∩ **NP-complete**, $L \in$ **NP-complete** and $L \in$ **NP**. Now, let $Q \in$ **PSPACE**. Since $L \in$ **PSPACE-complete** ∩ **NP-complete**, $L \in$ **PSPACE-complete** and $Q \leq_{\text{p}} L$. Since $L \in$ **NP**, Q must also be in **NP**. This implies that for every Q in **PSPACE**, Q is also in **NP**. Hence, **PSPACE** ⊆ **NP**.

   The second part of the question is easy:

$$\begin{aligned} L \in \textbf{PSPACE}\text{-complete} &\Leftrightarrow (\forall)_{Q\in\textbf{PSPACE}}[Q \leq_{\text{p}} L] \\ &\Leftrightarrow (\forall)_{Q\in\textbf{NP}}[Q \leq_{\text{p}} L] \quad (\text{since } \textbf{PSPACE} = \textbf{NP}) \\ &\Leftrightarrow L \in \textbf{NP}\text{-complete} \end{aligned}$$

   Therefore, **PSPACE-complete** ∩ **NP-complete**. ∎

4. **[15 points]** In the definition of **P**-complete, we use log-space reduction, which has a polynomial time bound. Now define **linearP**-complete as follows:

A language $L \in$ **P** is **linearP**-complete if and only if every language Q in **P** can be reduced to L with a linear time bound (*i.e.*, $O(|x|)$, where $x \in Q$ is an input), denoted as $Q \leq_{\text{linear}} L$.

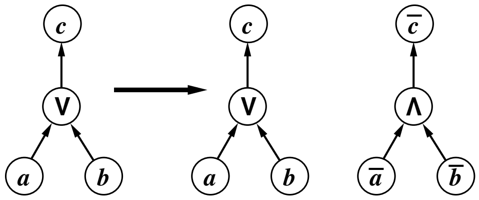
   How large/small is this **linearP**-complete class? Since a linear time bound is less than a polynomial time bound, it is obvious that **linearP**-complete is a proper subset of **P**-complete. Could this set be too small?

   Answer: This class is empty. Suppose **linearP**-complete is non-empty and contains a language L. Since $L \in$ **P**, there is a Turing machine $\mathcal{M}$ that takes $O(|x|^k)$ time to accept a $x \in L$. Because L is **linearP**-complete, every language $Q \in$ **P** is linear time reducible to L (*i.e.*, $O(|x|)$ for $x \in Q$ with a Turing machine $\mathcal{N}$. As a result, we can reduce Q to L and solve L in time $O(|x| + |f(x)|) = O(|x| + |x|^k) = O(|x|^k)$, where f is the reduction function computed by Turing machine $\mathcal{N}$. Note that the output length of the linear reduction carried out by Turning machine $\mathcal{N}$ is $O(|x|)$ because $\mathcal{N}$ can only scan no more than $O(|x|)$ tape squares. Hence, $O(|f(x)|) = O(|x|)$ and every problem in P can be solved in time $O(|x|^k)$, where k is a constant dependent on a problem in **linearP**. This is a contradiction because there are problem in **P** requires more than $O(|x|^k)$ time. Hence, **linearP** is empty. ∎

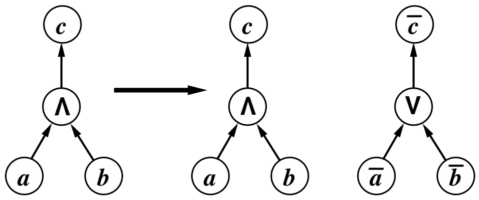
5. **[20 points]** We proved in class that the CIRCUIT VALUE problem is **P**-complete. Show that the CIRCUIT VALUE problem remains **P**-complete even if the circuit does not contain the negation operator $\neg$.

   Answer: Let the CIRCUIT VALUE version without $\neg$ be called the MONOTONIC CIRCUIT VALUE problem. Since it is a special case of the CIRCUIT VALUE problem which is in **P**, the MONOTONIC CIRCUIT VALUE problem is also in **P**. What remains to be shown is that the CIRCUIT VALUE problem is log-space reducible to the MONOTONIC CIRCUIT VALUE problem.

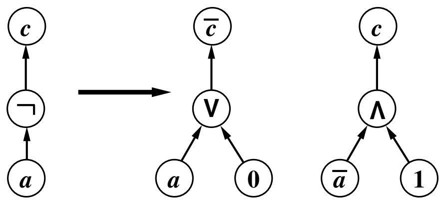
   First, let use see how we can eliminate all $\neg$ gates in an instance of the CIRCUIT VALUE problem. We shall use the *double-rail* logic, which is easier to understand although may not be as intuitive as applying de Morgan's law directly. The double-rail logic computes both the required result and its negation. Given a circuit, the double-rail logic is constructed in the following way. For each $\vee$ gate $a \vee b \to c$, the new circuit contains both the original $a \vee b \to c$ and its negation $(\neg a) \wedge (\neg b) \to \neg c$ as shown below:



   For each $\wedge$ gate $a \wedge b \to c$, the new circuit contains both the original $a \wedge b \to c$ and its negation $(\neg a) \vee (\neg b) \to \neg c$ as shown below.

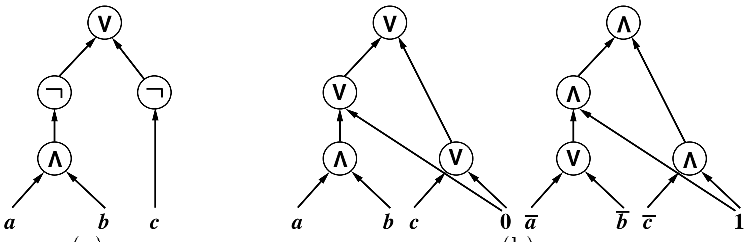


   Finally, we introduce two input values, 0 and 1. Each $\neg$ gate is replaced with a twisted-pair. More precisely, for each $\neg$ gate $\neg a \to c$, the new circuit contains a $\vee$ gate $a \vee 0 \to \neg c$ and a $\wedge$ gate $(\neg a) \wedge 1 \to c$ as shown below.



   Suppose we have a circuit as shown in (a) below. The corresponding double-rail logic is shown in (b). It is easy to see that the number of input lines, the number of output lines and the number of gates are doubled. In other words, the size of the circuit is doubled; but, the depth is not. It is not difficult to show that if a gate in the original circuit has an output 1 if and only if the corresponding gate that uses the original input line is 1 and if and only if the corresponding gate that uses the negated input is 0. You can prove this by induction on the depth. Keep in mind the double-rail version does not contain any $\neg$ gate.

   To make sure this construction can be carried out in log-space, we need to find some way to represent the input and the output circuits. Suppose the input circuit, the one with $\neg$, has n gates. Then, each gate's number can be represented with $k = \lceil \log_2(n+1) \rceil$ bits. The input tape which contains the



(a) (b)

circuit has n blocks, each of which has $k+2$ bits. The first two bits indicate the type of the gate (*i.e.*, input, $\vee$, $\wedge$ or $\neg$), and the next k bits provide the "successor" of this gate. The number of this gate is simply the block number. Therefore, to figure out what gate i is, we can read in block i and examine the first two bits. The next k bits tell where the output of this gate should be sent to.

As for the output tape of this reduction, it will contain $2+2n$ blocks. Block 0 contains the 0 input gate and block 1 contains the 1 input gate. For gate i of the input circuit, its corresponding gate in the output is in block $2i+2$, and its negation is in block $2i+3$. Since we have $2n+2$ blocks on the output tape, each block has $\lceil \log_2(2n+3) \rceil$ bits.

Once the input and output formats are fixed, the reduction is fairly easy. The reduction algorithm first writes 0 and 1 to block 0 and block 1 to the output tape. Note that the gates of the input circuit are topological sorted. Then, we simply read in the input gates, perform the transformation, and write the output to the output tape. It is not difficult to see that this reduction can be done in log-space. Write a program to verify it by yourself. Therefore, CIRCUIT VALUE is log-space reducible to MONOTONIC CIRCUIT VALUE, and the latter is **P**-complete. ∎