Examen final du cours (Calculabilité & Complexité (L3)

Les documents (notes, polycopiés, ..) et calculatrices (téléphone, tablette, ..) ne sont pas autorisés.

Date : 13 janv. 2019 à 10h30 / Durée : 2 heures

Note: when you give a reduction \( r : L \rightarrow L' \) between two problems/languages, be sure to define the value of \( r(x) \) in all cases, to argue the correctness of the reduction, and to argue its complexity.

Exercise 1: Closure properties of \( \mathsf{NP} \)

1. Among the following four statements, say which are true and which are false (justify your answers):

   1. If \( L_1 \) and \( L_2 \) are in \( \mathsf{NP} \) then also \( L_1 \cap L_2 \) is.
   2. If \( L_1 \) and \( L_2 \) are in \( \mathsf{NP} \) then also \( L_1 \cup L_2 \) is.
   3. If \( L_1 \) and \( L_2 \) are \( \mathsf{NP} \)-hard then also \( L_1 \cap L_2 \) is.
   4. If \( L_1 \) and \( L_2 \) are \( \mathsf{NP} \)-hard then also \( L_1 \cup L_2 \) is.

Solution:
The first two statements are correct (easy proof omitted) while the last two are incorrect. For example, if \( L_1 \) is a \( \mathsf{NP} \)-hard problem written with the alphabet \( A = \{a, b, \ldots\} \), we can consider a disjoint copy \( A' = \{a', b', \ldots\} \) of \( A \) and define \( L_2 \) as \( L_1 \) where \( a, b, \ldots \) are replaced with \( a', b', \ldots \) correspondingly. Then \( L_2 \) is \( \mathsf{NP} \)-hard too but \( L_1 \cap L_2 = \emptyset \) is not. If we now take \( L_1' = L_1 \cup A^* A' (A \cup A')^* \) and \( L_2' = L_2 \cup A^* A (A \cup A')^* \), we again have two \( \mathsf{NP} \)-hard languages. Now \( L_1' \cup L_2' \cup \emptyset \) is \( (A \cup A')^* \) hence is not \( \mathsf{NP} \)-hard.

Exercise 2: A \( \mathsf{PSPACE} \)-complete problem

2. Show that the following problem is \( \mathsf{PSPACE} \)-complete:

   **Input:** A Turing Machine \( \mathcal{M} \), a word \( x \), a number \( k \) written in unary.
   **Question:** Does \( \mathcal{M} \) accepts \( x \) using workspace at most \( k \)?

Solution:
Let us call the problem at hand \( \mathsf{PM}_k \). The size of its input is \( n = |\mathcal{M}| + |x| + k \).
\( \mathsf{PM}_k \) is in \( \mathsf{PSPACE} \): it is enough to simulate \( \mathcal{M} \) on \( x \) while maintaining two counters: one checking that the simulation does not use more than \( k \) cells in the auxiliary memory, and one checking that \( \mathcal{M} \) does not perform more than \( |Q| \cdot \log |x| \cdot |\Sigma|^k \) steps, hence does not
loop (this second check guarantees that the simulation terminates). The amount of memory
used by this simulation is $k + \log k + \log |Q| + \log \log |x| + k \log |\Sigma|$, hence is in $O(n)$.

$P_{Mk}$ is PSPACE-hard: If $L$ is any language in PSPACE there is a TM $M_L$ and a polynomial
$p_L$ such that $M_L$ decides $L$ using space $p_L(|x|)$. Thus $r_L : x \mapsto M_L, x, 1^{p_L(|x|)}$ is a reduction
proving $L \leq P_{Mk}$ since outputting $M_L$ (fixed) and copying $x$ is easy, while constructing
$1^{p_L(|x|)}$ from $x$ can be done in logspace.

**Problem 3: Stable subsets of graphs**

When $G = (V, E)$ is an undirected graph, a subset of vertices $X \subseteq V$ is called a *stable set*
if no edge $u \in E$ has both extremities in $X$. In other words, if the subgraph $G/X$ obtained
by restricting $G$ to vertices from $X$ has no edge at all. (A stable set is sometimes called an
*independent set*, or an *anticlique*.)

For example, $X = \{a, b, f\}$ is a stable of the graph $G$ below.

```
g: a b c d e f
  a  c
  d  e
  f
```

The *Stable* problem is, given an undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$ as inputs,
to decide whether $G$ has a stable subset of cardinal at least $k$. We assume that $k$ is written in
unary, so that the size $n$ of the input is in $O(|V|^2 + k)$.

3. Show that *Stable* is in NP.

**Solution:**

A nondeterministic algorithm is to guess a subset $X$ of $V$ and check its size and stability.
Obviously, $X$ is polynomial-sized and checking that it is stable and has size $\geq k$ can be
done in polynomial time.

4. Show that it is possible to solve *Stable* in time $n^{O(k)}$, i.e., in polynomial time *when $k$ is fixed*.

**Solution:**

A possible algorithm is to enumerate all subsets of $V$ having size $k$ and check each of them
for stability. There are $\binom{|V|}{k}$ such subsets, which is $\leq |V|^k$. Since checking some $X$
takes polynomial time $|V|^{O(1)}$, the whole procedure is in $|V|^{k+O(1)}$, which is in $n^{O(k)}$.

5. Let *Stable*$_{\text{bin}}$ be the variant of *Stable* where $k$ is given in binary, so that the size $n$ of an input
is in $O(|V|^2 + \log k)$. Show that *Stable* and *Stable*$_{\text{bin}}$ are logspace inter-reducible (i.e., there
exist reductions in both directions).

**Solution:**

Reducing *Stable* to *Stable*$_{\text{bin}}$ just requires replacing a string $1^k$ with a string denoting $k$ in
binary. This can be done in logspace by maintaining an auxiliary counter (in binary) that
is incremented each time we read one “1” symbol in the unary input. When all the input has been read, the counter contains the binary representation of \( k \) and has size \( \log k \).

Reducing \text{Stable}_{\text{bin}} \) to \text{Stable} is almost as simple. If \( k \leq |V| \), we replace the binary representation of the input \( k \) with a string \( 1^k \); the reduction is logspace in this case since \( |V| \leq n \).

If \( k > |V| \), it may be impossible to produce \( 1^k \) using only logspace but since no stable can have size \( > |V| \), the reduction just has to produce some fixed negative \text{Stable} instance (or replace \( k \) with \( 1^{|V|+1} \)).

6. Show that \( 3\text{SAT} \leq \text{Stable} \). Deduce the complexity of \text{Stable}.

Solution:
With a \( 3\text{SAT} \) instance \( \phi = C_1 \land C_2 \land \cdots \land C_m \) where each clause \( C_i \) is a disjunction \( l_{i,1} \lor l_{i,2} \lor l_{i,3} \) of three literals over some set \( V = \{ x_1, \ldots, x_p \} \) of variables, we associate a \text{Stable} instance \( G_{\phi}, k_{\phi} \) with \( k_{\phi} \) defined as \( m \) and \( G_{\phi} \) defined as \((V, E_1 \cup E_2)\) defined as follows:

\[
V \overset{\text{def}}{=} \{ (i, j) \mid i = 1, 2, \ldots, m \land j = 1, 2, 3 \},
E_1 \overset{\text{def}}{=} \{ ((i, j), (i', j')) \mid 1 \leq i \leq m \land 1 \leq j < j' \leq 3 \},
E_2 \overset{\text{def}}{=} \{ ((i, j), (i', j')) \mid 1 \leq i, i' \leq m \land l_{i,j} = \neg l_{i',j'} \}.
\]

The reduction is illustrated on the following example, where edges are drawn straight when from \( E_1 \) and bent when from \( E_2 \) (unless already in \( E_1 \)):

Let us prove that the reduction is correct: If \( \phi \) is satisfiable, say via some valuation \( v \), we select in each clause \( C_i \) one literal \( l_{i,r} \) satisfied by \( v \) and let \( X = \{ (i, r_i) \mid i = 1, \ldots, m \} \).

Two vertices in \( X \) cannot be connected by an \( E_1 \) edge since they come from different triangles. And they cannot be connected by an \( E_2 \) edge since \( E_2 \) connects vertices with opposite literals while \( X \) contains vertices with literals validated by a common \( v \). Hence \( X \) is stable.

Reciprocally, if \( G_{\phi} \) has a stable \( X \) of size \( m \), then \( X \) must contain one vertex per triangle, hence one literal per clause. Now there is a valuation that validates all these literals (hence \( \phi \) is satisfiable) since \( X \) cannot contain vertices with opposite literals (or it would not be stable w.r.t. \( E_2 \) edges).

We conclude by observing that the reduction is clearly logspace.

Now, with question 3 and since \( 3\text{SAT} \) is \( \text{NP} \)-complete, we deduce that \text{Stable} is \( \text{NP} \)-complete.
We are interested in Stable$_{\text{tree}}$, the restriction of Stable to graphs that are trees. Recall that a tree is an undirected graph such that there is one and only one simple path between any two vertices $s, t \in V$. (A simple path between $s$ and $t$ is a list of pairwise distinct vertices $v_0, \ldots, v_\ell$ with $v_0 = s$, $v_\ell = t$, and such that for all $0 \leq i < \ell$, $\{v_i, v_{i+1}\}$ is an edge.)

A tree can be transformed into a rooted directed tree (a data structure commonly used in programs) by choosing any vertex as the root and by orienting every edge away from the root. Below we show a tree $T$ and the two rooted trees obtained by choosing $a$ or $e$ as the root.

![Tree Diagram]

In a rooted directed tree with an edge $u \rightarrow v$, we say that $v$ is a child of $u$, and $u$ is the parent of $v$. A vertex is a leaf if it is has no children.

7. Show that if in a rooted directed tree $T$ there is a stable set of size $k$, then there is a stable set that contains all the leaves and that has size at least $k$.

**Solution:**

Let $T = (V, E)$ be a rooted tree and $X$ be a stable of size $k$ in $T$. If $u$ is a leaf in $T$ that does not appear in $X$, we modify $X$ by adding $u$ and removing the father of $u$ (if $u$ has one). The new $X$ is still stable (a leaf is only connected to its father) and its cardinality has not decreased. Repeating this process as long as some leaf is not in $X$, we finally obtain a stable of size $\geq k$ that contains all leaves.

8. Using the previous question, give an algorithm deciding Stable$_{\text{tree}}$ in polynomial-time. Justify its correctness and your complexity analysis.

**Solution:**

Our algorithm first checks that its input $G$ is a tree, and transforms it into a rooted tree $T$ (easily done in polynomial-time). Then is uses the following recursive function for computing a stable set of $T$ of maximum size and accepts if $|X_{\text{max}}| \geq k$.

```
function MaxStable(T):
    if T is empty then return Ø
    else let $X_1$ = all leaves of $T$
        let $T_1$ = $T$ without its leaves and their parents
        in return $X_1 \cup \text{MaxStable}(T_1)$
```

The algorithm is polynomial-time since each recursive call is on a strictly smaller tree and since computing $X_1$, $T_1$ and building $X_1 \cup \cdots$ is polynomial-time. To prove correctness, we note that there is a stable set of $T$ of maximum size that contains $X_1$ (by the previous question). Since $X_1$ joined with any stable of $T_1$ is a stable of $T$, and since if $X$ is a stable of $T$ that contains $X_1$, then $X \setminus X_1$ is a stable of $T_1$, one deduces (proof by induction on $T$) that $X_1$ joined with a maximum-sized stable of $T_1$ is a maximum-sized stable of $T$. 

Page 4