Wqos: homework assignment and midterm exam

Jean Goubault-Larrecq
LSV, ENS Paris-Saclay

Turn your solution in by October 18th, 2019!

Note: be as clear as possible. If I cannot understand you, this won't help you obtain a good grade. Specific hints: (1) write, do not scribble; (2) if I use a specific notation, use the same; (3) use the definitions and results of the lecture notes—if you know an equivalent definition from the literature but the equivalence is not proved in the lecture notes, do not use it (as a last resort, include the proof of equivalence); (4) justify every claim you make, by a proof, by a definition, by a previous question, or by a theorem (preferentially use theorem *names*, such as "Higman's Lemma", rather than numbers); (5) find the simplest possible proof argument.

Also, the final exam may contain sequels to some of the questions asked here—who knows.

Section 2 depends on Section 1, and Section 4 partly depends on Section 3. Those are the only dependencies.

1 Induced subgraphs

A *graph* G is a pair (V, E), where V is the finite set of vertices of G, and $E \subseteq V \times V$ is the finite set of edges of G. We will always assume that V is a subset of some fixed infinite subset such as $\mathbb{N}$ (otherwise the class of graphs would not even be a set).

A graph *homomorphism* $f\colon (V, E) \to (V', E')$ is a map $f\colon V \to V'$ such that for every $(s, t) \in E$, $(f(s), f(t)) \in E'$. It is *injective* if and only if it is injective as a map from V to V'. It is an *embedding* if and only if for all $s, t \in V$, $(s, t) \in E$ is *equivalent* to $(f(s), f(t)) \in E'$.

We say that a graph G is a *subgraph* of a graph G' if and only if there is an injective homomorphism from G into G'. It is an *induced subgraph* if and only if there is an embedding of G into G'. For example, the graph with two vertices and no edge is a subgraph of the graph with the same two vertices and one edge between them, but is not an *induced* subgraph.

It is helpful to think of graphs up to isomorphism, that is, up to (bijective) renaming of their vertices. Up to isomorphism, an induced subgraph of $G = (V, E)$ is uniquely determined by a subset V' of V, as $(V', E \cap (V' \times V'))$.

Question 1 Show that the induced subgraph relation is not a well-quasi-ordering.

Question 2 One can show that following problem:

> **INPUT:** two graphs H, G;
>
> **QUESTION:** is H an induced subgraph of G?

is **NP**-complete. Show that, for *fixed* H, the following problem:

> **INPUT:** a graph G;
>
> **QUESTION:** is H an induced subgraph of G?

can be solved in polynomial time. Give explicitly the degree of the polynomial, too.

2 m-partite cographs

For every $m \in \mathbb{N}$, let Σ_m be the signature consisting of:

- constants (i.e., functions of arity 0) i, $1 \leq i \leq m$;
- function symbols $edge_R$ of variable arity, for every binary relation $R \subseteq \{1, \cdots, m\}^2$.

Implicitly, all of those are pairwise distinct. In particular, $R \neq i$ for all binary relations R and constants i.

Let TM_m be the set of terms built on those symbols, restricted in such a way that the constants i are always applied to an empty list of argument; instead of writing $i()$, we simply write i. Hence $edge_R(1, edge_S(1, 2), edge_S(2, 3))$ is in TM_m, but not $1(2, 3)$.

The elements of TM_m are called *tree models* on the m colors $1, \ldots, m$.

A *colored graph* (with colors in $\{1, \cdots, m\}$) is a pair of a graph $G = (V, E)$ with a (*coloring*) map $\lambda \colon V \to \{1, \cdots, m\}$. Its *underlying graph* is G. We will sometimes confuse a colored graph with its underlying graph, and that will allow us to make sense of the 'induced subgraph' and 'subgraph' relations on colored graphs.

The *semantics* $[\![t]\!]$ of a tree model t is a colored graph defined as follows:

- for every constant $i \in \{1, \cdots, m\}$, $[\![i]\!]$ is the colored graph $((\{*\}, \emptyset), \{* \mapsto i\})$ with one vertex $*$ colored i, and no edge;
- for every binary relation $R \subseteq \{1, \cdots, m\}^2$, $[\![edge_R(t_1, \cdots, t_n)]\!]$ is the graph obtained by taking the disjoint union of the colored graphs $[\![t]\!]_k$, $1 \leq k \leq n$, and for each pair $(i, j) \in R$, for each pair of positions $k \neq \ell$ $(1 \leq k, \ell \leq n)$ in the argument list, adding an edge from each i-colored vertex of $[\![t_k]\!]$ to each j-colored vertex of $[\![t_\ell]\!]$. As a very special case, $[\![edge_R()]\!]$ is the empty graph.

In the sequel, we fix $m \in \mathbb{N}$.

Question 3 Find a well-quasi-ordering $\leq$ on Σ_m such that, for all $s, t \in TM_m$ such that $s \leq_T t$, $[\![s]\!]$ is an induced subgraph of $[\![t]\!]$. Justify.

Question 4 An *m-partite cograph* is any graph that one can write as $[\![t]\!]$ for some $t \in TM_m$. Show that the induced subgraph relation is wqo on the set of m-partite cographs.

Question 5 Given any m-partite cograph G and an induced subgraph H of G, show that H is also an m-partite cograph.

Question 6 Let P be any hereditary property of graphs. By 'hereditary' I mean that if P is true of a graph G, and H is an induced subgraph of G, then P must also be true of H. Show that the following problem:

INPUT: an m-partite cograph G;

QUESTION: is P true of G?

is decidable in polynomial time, for every hereditary property of graphs. Remember that both m and P are fixed parameters, not inputs to the problem.

Question 7 3-colorability is the following question:

INPUT: a graph G;

QUESTION: is there a way of assigning each vertex of G a color from $\{1, 2, 3\}$ in such a way that every edge connects vertices of different colors?

This problem is well-known to be **NP**-complete. Can we decide it in polynomial time when the input is required to be an m-partite cograph?

Question 8 What are the minimal non-3-colorable 1-partite cographs? By minimal, we mean with respect to the induced subgraph relation.

3 Upwards-closed and Downwards-closed subsets of Σ^*

In this section, we fix a finite alphabet Σ, ordered by equality. We quasi-order Σ^* by the embedding quasi-ordering $\leq_*$. (Although $\leq$ is $=$, really, we refrain from writing $=_*$, which would probably be confusing.) As usual, $\uparrow$ denotes upward closure and $\downarrow$ denotes downward closure.

We also assume that you have basic knowledge on regular languages and finite automata. For every regular expression L, we will write $L^?$ for the regular expression $L + \epsilon$, where $a \in \Sigma$, denoting the language $L \cup \{\epsilon\}$.

Question 9 Given any finite word $w \in \Sigma^*$, show that $\downarrow w$ is a regular language. More: show that one can compute a regular expression denoting $\downarrow w$ from any word w given in input.

Question 10 Given any finite word $w \in \Sigma^*$, show that $\uparrow w$ is a regular language. More: show that one can compute a regular expression denoting $\uparrow w$ from any word w given in input.

Question 11 Show that every upwards-closed subset U of Σ^* is a regular language. Given a basis for U, show that one can even compute a regular expression whose language is U.

Question 12 Deduce that every downwards-closed subset D of Σ^* is a regular language, and again, that given a basis of the complement of D, we can compute a regular expression whose language is D.

4 The Valk-Jantzen-GL algorithm

Let X be a wqo.

We imagine that we can represent the elements of X on a computer, and also the downwards-closed subset D of X, and that:

- given $x, y \in X$, one can decide whether $x \leq y$;
- the map $\downarrow : x \mapsto \downarrow x$ is computable (from X to the set of downwards-closed subsets of X);
- the function $\complement : A \mapsto X \smallsetminus \uparrow A$ is computable (from the set of finite subsets of A to the set of downwards-closed subsets of X).

As an example, the whole purpose of Section 3 was to convince you that those assumptions are true when $X = \Sigma^*$, provided that you represent downwards-closed subsets of Σ^* as regular expressions.

Imagine you are given some piece of data, and you know that this piece of data represents an upwards-closed subset U of X, but you don't know a basis of U. Instead, we assume that:

> (H) we are given access to an oracle $\mathcal{O}_U$ that, given any downwards-closed subset D of X, decides whether D intersects U.

('Oracle' is computability-speak. If you are an ML or Haskell programmer, what I am saying is that you are given a function $\mathcal{O}_U$ as input, such that $\mathcal{O}_U(D)$ returns true if D intersects U, and false otherwise. The 'piece of data' above can be taken as simply that function $\mathcal{O}_U$. The VJGL algorithm mentioned below—see Figure 1—is therefore simply a second-order function taking $\mathcal{O}_U$ as input.)

Question 13 Show that you can decide, given $x \in X$, whether $x \in U$.

Question 14 Show that you can decide, given a finite subset A of X, whether $U \subseteq \uparrow A$.

Question 15 We run the algorithm of Figure 1. I am *not* claiming that it is practical! The algorithm takes the piece of data representing U as input, and has access to the oracle $\mathcal{O}_U$. As remarked above, this can be expressed more simply by saying that VJGL takes $\mathcal{O}_U$ as input.

The tests $x \in U$ and $U \subseteq \uparrow A$ at lines 4 and 6 stand for the algorithms you have described in **Question 13** and **Question 14** respectively.

```
def VJGL ($\mathcal{O}_U$):
    $A := \emptyset$
    for each $x \in X$:
        if $x \in U$ and $x \notin \uparrow A$:
            $A := A \cup \{x\}$
            if $U \subseteq \uparrow A$:
                break # exit for loop
    return $A$
```

Figure 1: The VJGL algorithm

What does algorithm VJGL compute? You should justify your answer: why it computes what you claim it does, and also why it terminates.

Question 16 Let Σ be a finite alphabet (ordered by equality again). Imagine that U is an upwards-closed subset of Σ^*, which you do not know. However, imagine that, for every regular expression L, you are allowed to test whether the language of L intersects U, through an oracle $\mathcal{O}_U$. Show that this gives you complete knowledge of U, in the sense that there is a way to compute a finite basis of U from just those inputs. (Formally, define an algorithm that takes the oracle $\mathcal{O}_U$ as input, and returns a finite basis for U.)

Question 17 For this question, you need to know that given a context-free grammar G and a regular expression L, we can compute a context-free grammar—which we write as $G \cap L$—whose language is the intersection of the languages of G and L. (Beware that we cannot in general compute the intersection of two context-free grammars.) It is also decidable whether the language of a context-free grammar given as input is empty or not. Show that there is an algorithm that, given a context-free grammar G as input, computes a regular expression L whose language is the set of words that contains a subword in the language of G.