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 $f : (V,E) \to (V',E')$ is a map $f : V \to V'$ 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.
The (oriented) cycles form an infinite antichain.

Question 2 One can show that following problem:

**INPUT:** two graphs \( H, G \);

**QUESTION:** is \( H \) an induced subgraph of \( G \)?

is \( \text{NP} \)-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.

Let \( G = (V, E) \).

Let \( u_1, \ldots u_n \) be the vertices of \( H \). \( H \) is an induced subgraph of \( G \) if and only if we can find \( n \) vertices \( v_1, \ldots, v_n \) in \( G \) such that:

(a) \( v_1, \ldots, v_n \) are pairwise distinct;
(b) for all \( i, j \) with \( 1 \leq i, j \leq n \), there is an edge from \( v_i \) to \( v_j \) in \( G \) if and only if there is an edge from \( u_i \) to \( u_j \) in \( H \).

In order to test that, we simply run \( n \) nested ‘for’ loops, for all vertices \( v_1, \ldots, \) for all vertices \( v_n \) (remember: \( n \) is a constant). Inside the loop nest, we check:

(a) that for all \( i, j \) with \( 1 \leq i, j \leq n \), \( v_i \neq v_j \); this takes constant time since \( n \) is a constant (assuming vertex comparison takes constant time; on a Turing machine, that really takes \( \log |V| \) time);
(b) that for all \( i, j \) with \( 1 \leq i, j \leq n \), if there is an edge from \( u_i \) to \( u_j \) in \( H \) then there is an edge from \( v_i \) to \( v_j \) in \( G \), and otherwise there is no edge from \( v_i \) to \( v_j \) in \( G \); this again takes constant time (resp., \( |V|^2 \) time with an adjacency matrix representation on a Turing machine).

Hence we obtain an algorithm that runs in time \( |V|^n \) (resp., \( |V|^{n+2} \) on a Turing machine), which is a polynomial since \( n \) is constant.

2 \( m \)-partite cographs

For every \( m \in \mathbb{N} \), let \( \Sigma_m \) be the signature consisting of:

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

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 \( \text{edge}_R(1, \text{edge}_S(1, 2), \text{edge}_S(2, 3)) \) 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 \( \lambda: V \to \{1, \cdots, m\} \). 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 \( ((\{\star\}, \emptyset), \{\star \mapsto i\}) \) with one vertex \( \star \) colored \( i \), and no edge;

- for every binary relation \( R \subseteq \{1, \cdots, m\}^2 \), \( [\text{edge}_R(t_1, \cdots, t_n)] \) 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, \( [\text{edge}_R()] \) is the empty graph.

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

**Question 3** Find a well-quasi-ordering \( \leq \) on \( \Sigma_m \) such that, for all \( s, t \in TM_m \) such that \( s \leq t \), \( [s] \) is an induced subgraph of \( [t] \). Justify.

We take equality for \( = \). This is a well-quasi-ordering because \( \Sigma_m \) is finite (of cardinality \( m + 2^m \)).

We show by induction of the definition of \( \leq \) in the lecture notes that \( s \leq t \) implies that \( [s] \) is an induced subgraph of \( [t] \); more: that there is an embedding of \( [s] \) into \( [t] \) that respects colors.

- **Rule (T-add).** If \( t = g(t_1, \cdots, t_n) \) and \( s \leq t_k \) for some \( k, 1 \leq k \leq n \), then, first, \( g = \text{edge}_R \) for some binary relation \( R \), since constants are never applied to a non-empty list of arguments. By induction hypothesis \( [s] \) is an induced subgraph of \( [t_k] \) through an embedding \( f \) that respects colors. Then \( f \) is also an embedding of \( [s] \) into \( [t] \), which is obtained by adding new vertices and new edges to \( [t_k] \). The new edges are all between disjoint graphs, so no new edge is added inside \( [t_k] \).

- **Rule (T-inc), first case.** If \( s \) and \( t \) are colors, then they are the same color \( i \), and then \( \star \mapsto \star \) defines an embedding that respects colors.

- **Rule (T-inc), first case.** If \( s = \text{edge}_R(s_1, \cdots, s_m) \) and \( t = \text{edge}_R(t_1, \cdots, t_n) \), with the same \( R \) and where \( s_1 \cdots s_m \leq_T t_1 \cdots t_n \). In particular, there is a (monotonic) injection \( g: \{1, \cdots, m\} \to \{1, \cdots, n\} \) such that \( s_k \leq_T t_{g(k)} \) for every \( k, 1 \leq k \leq m \). By induction hypothesis, there is an embedding \( f_k \) from \( [s_k] \) into \( [t_{g(k)}] \) for every \( k \). Since the graphs \( [s_k] \) have pairwise disjoint...
vertex sets, we can define a graph homomorphism \( f: \llbracket s \rrbracket \to \llbracket t \rrbracket \)
by: \( f(u) = f_k(u) \), where \( k \) is the unique index such that \( u \) is a vertex of \( \llbracket s_k \rrbracket \). This respects colors by definition.

We claim that this is an embedding. Imagine we have an edge between two vertices inside the image of \( f \).

- If they are both in the same graph \( \llbracket t_{q(k)} \rrbracket \), then \( k \) is determined uniquely because those graphs are pairwise disjoint and \( g \) is injective. Then they are edges of the form \( f(u) = f_k(u) \) and \( f(v) = f_k(v) \) with \( u, v \in \llbracket s_k \rrbracket \), and since there is an edge from \( f_k(u) \) to \( f_k(v) \) and \( f_k \) is an embedding, there is an edge from \( u \) to \( v \) (in \( \llbracket s_k \rrbracket \), hence also in \( \llbracket s \rrbracket \)).

- If they are in distinct—hence disjoint—graphs \( \llbracket t_{q(k)} \rrbracket \) and \( \llbracket t_{q(\ell)} \rrbracket \), then, reading off the semantics of \( t \), they must form a pair of a vertex \( f_k(u) \) of some color \( i \), with \( u \in \llbracket s_k \rrbracket \), and of a vertex \( f_\ell(v) \) of some color \( j \), with \( v \in \llbracket s_\ell \rrbracket \), and where \( (i, j) \in R \). Since \( f_k \) and \( f_\ell \) preserve colors, \( u \) has color \( i \) and \( v \) has color \( j \), so there is an edge from \( u \) to \( v \) in \( \llbracket s \rrbracket \).

**Question 4**

An \( m \)-partite cograph is any graph that one can write as \( \llbracket t \rrbracket \) for some \( t \in TM_m \). Show that the induced subgraph relation is wqo on the set of \( m \)-partite cographs.

The map \( t \mapsto \llbracket t \rrbracket \) is monotonic by **Question 3**. It is surjective by definition. By Kruskal’s Theorem, \( \leq_T \) is wqo on the set of all trees with function symbols from \( \Sigma_m \) (ordered by equality, as in **Question 3**). Hence the subset \( TM_m \) is also wqo under \( \leq_T \) (Lemma 1.13, item 2). It follows that its image by \( t \mapsto \llbracket t \rrbracket \) is wqo under induced subgraph, by Lemma 1.13, item 4.

**Question 5**

Given any \( m \)-partite cograph \( G \) and an induced subgraph \( H \) of \( G \), show that \( H \) is also an \( m \)-partite cograph.

By assumption, \( G = (V, E) \) is the graph underlying a colored graph of the form \( \llbracket t \rrbracket \) for some \( t \in TM_m \). An easy induction on \( t \) shows that \( V \) is simply the set of leaves of \( t \). (Two distinct leaves can have the same color: this is not a DAG.) Up to isomorphism, \( H \) is entirely determined by a subset \( W \) of \( V \). We define a new term \( t|_W \) from \( t \) by only keeping the leaves of \( t \) that are in \( W \). Then \( H = \llbracket t|_W \rrbracket \), as an easy induction shows.

Formally, let \( f \) be an embedding of \( H \) into \( G \) (and \( W \) \( \overset{\text{def}}{=} f^{-1}(V) \)). We define \( s|_W \) by induction on the subterm \( s \) of \( t \), and we show that \( \llbracket s|_W \rrbracket \) is the subgraph of \( \llbracket s \rrbracket \) obtained as the inverse image of \( \llbracket s \rrbracket \) by \( f \) (namely, its vertex set is the inverse image of the vertex set of \( \llbracket s \rrbracket \) by \( f \), and then its set of edges is defined uniquely). This is immediate.
**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.

Let $\mathcal{G}_m$ be the set of all $m$-partite cographs, preordered by induced subgraph. The set $B$ of $m$-partite cographs $G$ that make $P$ true is downwards-closed in $\mathcal{G}_m$. Hence its complement $A \defeq \mathcal{G}_m \setminus B$ is upwards-closed. By Question 4 and characterization of wqos II (Proposition 1.7), there is a finite subset $A_0$ of $A$ such that, for every $G \in \mathcal{G}_m$, $P$ is true of $G$ if and only if no element of $A_0$ is an induced subgraph of $P$.

The algorithm takes $G$ as input, and loops through the elements $H$ of $A_0$. If some $H \in A_0$ is an induced subgraph of $G$, then we reject. Otherwise, we accept. Note that, since $H$ is fixed, we can test whether $H$ is an induced subgraph of $G$ in time polynomial in the size of $G$, by Question 2.

Note that Question 5 is not needed here.

**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?

Yes: 3-colorability is a hereditary property of graphs, then apply Question 6.

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

The 1-partite cographs are exactly the disjoint unions of cliques, where a clique is a graph where there is an edge from any vertex to any other vertex (no self-loop), as an easy induction on tree models shows. Note that we do not just have cliques: edge computes disjoint unions.

Such a graph is 3-colorable if and only all its cliques have cardinality at most 3. Note that any clique embeds into any clique of larger cardinality.

Hence there is just one minimal non-3-colorable 1-partite cograph, up to isomorphism: the clique on 4 vertices.
It follows that the algorithm of the previous question runs in time $O(n^4)$—already for 1-partite cographs.

3 Upwards-closed and Downwards-closed subsets of $\Sigma^*$

In this section, we fix a finite alphabet $\Sigma$, ordered by equality. We quasi-order $\Sigma^*$ 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.

Given that $w = a_1 a_2 \cdots a_n$ where each $a_i$ is in $\Sigma$, we can compute $\downarrow w$ as $a_1^? a_2^? \cdots a_n^?$.

**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.

Given that $w = a_1 a_2 \cdots a_n$ where each $a_i$ is in $\Sigma$, we can compute $\uparrow w$ as $\Sigma^* a_1^? \Sigma^* a_2^? \cdots \Sigma^* a_n^?$. If $n = 0$, this should be read as $\Sigma^*$.

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

Every upwards-closed subset $U$ is a finite union of sets $\uparrow w$, hence is a finite union of regular languages. This is clearly computable, provided we represent $U$ as one of its basis.

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

We use the previous question, then compute a regular expression for the complement of the regular expression we have just obtained. The classical algorithm converts the regular expression to a finite automaton, determinizes it, complements the latter, and then converts back the resulting automaton to a regular expression.
def VJGL (O_U):
    A := ∅
    for each x ∈ X:
        if x ∈ U and x ∉ ↑ A:
            A := A ∪ {x}
        if U ⊆ ↑ A:
            break # exit for loop
    return A

Figure 1: The VJGL algorithm

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 ∈ X, one can decide whether x ≤ y;
- the map ↓: x ↦ ↓ x is computable (from X to the set of downwards-closed subsets of X);
- the function □: A ↦ □ X \setminus ↑ 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 = Σ∗, 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 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 O_U as input, such that O_U(D) returns true if D intersects U, and false otherwise. The ‘piece of data’ above can be taken as simply that function O_U. The VJGL algorithm mentioned below—see Figure 1—is therefore simply a second-order function taking O_U as input.)

Question 13 Show that you can decide, given x ∈ X, whether x ∈ U.

Since x ∈ U if and only if ↓ x intersects U, it suffices to call O_U on ↓ x.

Question 14 Show that you can decide, given a finite subset A of X, whether U ⊆ ↑ A.
It suffices to test whether $U$ does not intersect the complement of $\uparrow A$. For that, we compute $-O_U(\bar{\uparrow}(A))$.

**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 $O_U$. As remarked above, this can be expressed more simply by saying that VJGL takes $O_U$ 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.

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

- It computes a finite basis of $U$. At each turn of the loop, $A$ contains a subset of $U$, so $\uparrow A \subseteq U$. When (if ever) it returns, we also have $U \subseteq \uparrow A$, so $U = \uparrow A$.

  The algorithm terminates because $U$ has a finite basis $A_0$, and eventually all the points of $A_0$ will have been enumerated. At this point every point of $A_0$ (or an even lower point) will be in $A$, and then $U = \uparrow A_0 \subseteq \uparrow A$, causing the algorithm to stop and return $A$.

  **Erratum:** the algorithm does not terminate if $U$ is empty and $X$ is infinite. This can be repaired by first testing whether $O_U(\bar{\uparrow}(\emptyset))$ holds: if so, $U$ is non-empty and we proceed, otherwise we immediately return $\emptyset$.

**Question 16** Let $\Sigma$ be a finite alphabet (ordered by equality again). Imagine that $U$ is an upwards-closed subset of $\Sigma^*$, 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 $O_U$. 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 $O_U$ as input, and returns a finite basis for $U$.)

- $X = \Sigma^*$ satisfies all the assumptions. (Notably, to decide $w \leq_* w'$, we can just test whether $w$ is in the regular language $\downarrow w'$, for example.) Then we simply run the VJGL algorithm.

**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 $G$. 

8
Let $U$ be $\uparrow L(G)$. We apply the algorithm of the previous question—really, the VJGL algorithm. We define the oracle $O_U$ so that, given a regular expression for a downwards-closed language $L$, decides whether $L \cap U \neq \emptyset$. Since $L$ is downwards-closed, $L \cap U \neq \emptyset$ if and only if $G \cap L \neq \emptyset$. The oracle simply computes $G \cap L$ and tests whether it is non-empty.