Exam

Web Data Management
Master Parisien de Recherche en Informatique

March 4th, 2020

This is the final exam for the Web Data Management class, which will determine 50% of your grade for this class (the other 50% being given by the project). The exam consists of two independent exercises. **You must write your answer to each exercise on a *separate sheet* of paper.** You can choose to answer the questions in English or in French, as you prefer.

Write your name clearly on the top right of every sheet used for your exam answers, and number every page.

You are provided with an SPARQL "cheat sheet" which gives you a summary about the syntax of SPARQL. You are additionally allowed **one A4 sheet** (i.e., two pages, one on each side) with the content of your choice. You may not use any other written material.

The exam is **strictly personal**: any communication or influence between students, or use of outside help, is prohibited. No electronic devices such as calculators, computers, or mobile phones, are permitted. Any violation of the rules may result in a grade of 0 and/or disciplinary action.

Exercise 1: Provenance for SPARQL queries (10 points)

We assume that we have a namespace `m`, and we consider the following RDF document D_0 in N-Triples notation, seen as a set of RDF facts (ignore for now the comments preceded by `#`):

```
<m:joe> <m:likes> <m:sparql> . #l1
<m:jane> <m:likes> <m:sparql> . #l2
<m:jane> <m:likes> <m:xpath> . #l3
<m:joe> <m:follows> <m:jane> . #f1
<m:jane> <m:follows> <m:joe> . #f2
<m:jack> <m:follows> <m:joe> . #f3
<m:jack> <m:follows> <m:jane> . #f4
```

We consider the SPARQL query Q_0 with two free variables `?x` and `?y` and returning all answer pairs (`?x`, `?y`) such that `?x` follows someone who likes `?y`.

Question 0. Write Q_0 in SPARQL and write the output $Q_0(D_0)$ of Q_0 on D_0, i.e., the answers returned by Q_0 on D_0.

Answer.

```
SELECT ?x ?y WHERE {
  ?x <m:follows> ?z
  ?z <m:likes> ?y
}
```

?x	?y
joe	sparql
joe	xpath
jane	sparql
jack	sparql
jack	xpath

In the rest of the exercise, we will study the *Boolean provenance* of queries. Remember that a *valuation* of X_0 is a function $\nu : X_0 \to \{0, 1\}$, and a *Boolean function* ϕ on X_0 is a mapping from valuations of X_0 to $\{0, 1\}$: given a valuation ν and a Boolean function ϕ, we say that ϕ *evaluates to true on* ν if $\phi(\nu) = 1$, and otherwise that it *evaluates to false on* ν. We will write Boolean functions as Boolean formulas. For instance, $l_1 \wedge (l_2 \vee f_2)$ is a Boolean function. It evaluates to true under the valuation ν_0 mapping l_1, l_2, f_1, f_4 to 1 and l_3, f_2, f_3 to 0.

We annotate each fact of the RDF document D_0 above with a Boolean variable in $X_0 = \{l_1, l_2, l_3, f_1, f_2, f_3, f_4\}$ in the order indicated. Given a valuation ν of X_0, we write $\nu(D_0)$ to be the RDF document consisting of precisely the facts of D_0 that ν maps to 1, formally $\nu(D_0) := \{f \in D_0 \mid \nu(f) = 1\}$.

Question 1. Write $\nu_0(D_0)$ for the example valuation ν_0 above, and write the output of Q_0 on $\nu_0(D_0)$.

Answer.

```
<m:joe> <m:likes> <m:sparql> . #l1
<m:jane> <m:likes> <m:sparql> . #l2
<m:joe> <m:follows> <m:jane> . #f1
<m:jack> <m:follows> <m:jane> . #f4
```

?x	?y
joe	sparql
jack	sparql

Given a SPARQL query Q and a answer **a** for Q on D_0, the *provenance* of **a** for Q on D_0 is the Boolean function ϕ on the variables X_0 such that, for any valuation ν of X_0, we have that ϕ evaluates to true under ν if and only if **a** is an answer to Q on $\nu(D_0)$. For instance, the provenance of the answer (*jane*) for the following SPARQL query on D_0:

```
SELECT ?x WHERE {
  ?x <m:likes> ?y
}
```

is $l_2 \vee l_3$.

Question 2. Write again the output of Q_0 and D_0 like in Question 0, and write next to each answer of the output its provenance for Q_0 on D_0.

Answer.

?x	?y	provenance
joe	sparql	$f_1 \wedge l_2$
joe	xpath	$f_1 \wedge l_3$
jane	sparql	$f_2 \wedge l_1$
jack	sparql	$f_3 \wedge l_1 \vee f_4 \wedge l_2$
jack	xpath	$f_4 \wedge l_3$

Question 3. We say that two SPARQL queries Q and Q' are *equivalent* if, for every RDF document D, we have $Q(D) = Q'(D)$, i.e., Q and Q' have the same output on D.

Give an example of two SPARQL queries Q and Q' that are not equivalent but have the same output on D_0 and for which every tuple $\mathbf{a}$ of their common output has the same provenance for Q and Q'.

Answer. Simply write two non-equivalent queries with no output on D_0, e.g.:

```
SELECT ?x ?y WHERE {
  ?x <m:likes> ?z .
  ?z <m:likes> ?y .
}
```

and

```
SELECT ?x ?y WHERE {
  ?x <m:likes> ?z .
  ?z <m:likes> ?t .
  ?t <m:likes> ?y .
}
```

These two queries are clearly not equivalent, but they all return the same empty set of results on D_0, where vacuously all tuples have the same provenance.

Question 4. A SPARQL query Q is *monotone* if, given two RDF documents D and D', if $D \subseteq D'$ (meaning that every fact of D is in D'), then $Q(D) \subseteq Q(D')$ (meaning that every tuple in the output of Q on D is also in the output of Q on D'). Is the query Q_0 monotone? Give a short proof or a counterexample.

Answer. The query Q_0 is monotone. Indeed, consider RDF documents $D \subseteq D'$, and consider a tuple (a, b) in the output of Q_0 on D. There must exist c such that the following facts are in D:

```
a <m:follows> c .
c <m:likes> b .
```

As $D \subseteq D'$, these facts are then also in D', so that (a, b) is in the output of Q_0 on D'.

Question 5. Given two valuations ν and ν' on X_0, we write $\nu \subseteq \nu'$ if for all $x \in X_0$ such that $\nu(x) = 1$, we have $\nu'(x) = 1$. A Boolean function ϕ on X_0 is *monotone* if, for every valuations $\nu \subseteq \nu'$, if ϕ evaluates to true on ν, then ϕ evaluates to true on ν'.

Show that, for any monotone SPARQL query Q, for any answer tuple $\mathbf{a}$ of Q, the provenance of $\mathbf{a}$ for Q on D_0 is a monotone Boolean function.

Answer. Assume by contradiction that Q has an answer tuple $\mathbf{a}$ whose provenance ϕ is not a monotone Boolean function. This means that there are two valuations $\nu \subseteq \nu'$ such that ϕ evaluates to true under ν but to false under ν'. Now, by definition of the provenance we know that $\mathbf{a}$ is an answer to Q on $\nu(D_0)$, but that it is not an answer to Q on $\nu'(D_0)$. This witnesses that $Q(\nu(D_0))$ is not a subset of $Q(\nu'(D_0))$, and as $\nu(D_0) \subseteq \nu'(D_0)$ by definition, this contradicts the monotonicity of Q.

Question 6. Give an example of a SPARQL query Q_6 having an answer $\mathbf{a_6}$ such that the provenance ϕ for Q_6 on D_0 is a Boolean function ϕ which evaluates to true under *exactly* one valuation.

Answer. We simply write a query Q_6 that follows the structure of the database:

```
SELECT ?joe ?jane ?jack ?sparql ?xpath WHERE {
  ?joe <m:likes> ?sparql .
  ?jane <m:likes> ?sparql .
  ?jane <m:likes> ?xpath .
  ?joe <m:follows> ?jane .
  ?jane <m:follows> ?joe .
  ?jack <m:follows> ?joe .
  ?jack <m:follows> ?jane .
}
```

The answer tuple $\mathbf{a_6} := (\mathit{joe}, \mathit{jane}, \mathit{jack}, \mathit{sparql}, \mathit{xpath})$ is an answer on D_0 but on none of its strict subsets, so its provenance (i.e., $l_1 \wedge l_2 \wedge l_3 \wedge f_1 \wedge f_2 \wedge f_3 \wedge f_4$) only has one satisfying valuation, namely the one mapping every variable of X_0 to 1.

Another solution is to write some (non-monotone) Boolean SPARQL query that tests that there are no facts at all, so that the empty tuple has as provenance the Boolean function which is only satisfied by the Boolean function mapping each variable to 0.

Question 7. We say that a Boolean function ϕ on X_0 *implies* a Boolean function ψ on X_0 if, for every valuation ν of X_0, if ϕ evaluates to true under ν then ψ also does.

Can there be a SPARQL query Q_7 having two answer tuples $\mathbf{a}$ and $\mathbf{b}$ whose respective provenances ϕ and ψ are different Boolean functions such that ϕ implies ψ? Give an example, or prove that this is impossible.

Answer. Consider the query:

```
SELECT ?w ?z WHERE {
  ?w <m:follows> ?x .
  ?x <m:follows> ?y .
  ?y <m:follows> ?z .
}
```

Consider the answer tuples $\mathbf{a} = (\mathit{jane}, \mathit{joe})$ and $\mathbf{b} = (\mathit{jack}, \mathit{joe})$. The provenance ϕ of $\mathbf{a}$ is:

$$f_2 \wedge f_1 \wedge f_2$$

and the provenance ψ of $\mathbf{b}$ is:

$$f_3 \wedge f_1 \wedge f_2$$

And indeed ϕ and ψ are different but ϕ implies ψ as required.

Question 8. We consider the problem of computing the provenance of a SPARQL query, defined as follows. We fix a SPARQL query Q of the form `SELECT ?x1 ... ?xn WHERE { ... }` where the variables of the `SELECT` clause do not use any operator, and where the body of the `WHERE` clause only consists of RDF facts where the predicate of each fact is a constant and where the subject and object of the facts can be constant or variables (without any operator or other feature of the SPARQL syntax). We want, given an RDF document D with facts annotated by variables, to compute all answers of Q on D and their provenance. We study the *data complexity* of this problem, with the input being the annotated RDF document D.

Design an algorithm to solve this problem with polynomial data complexity, and argue that it is correct.

Answer. There are constantly many variables used in Q, so we can enumerate all possible ways to map them to entities of D: there are polynomially many ways. For each way, we check if the resulting set of facts is a subset of D. If it is, then we see the answer tuple that this gives us (from the variables exported in the `SELECT` statement), and we add a disjunction to the provenance of this tuple with the conjunction of the annotations of all the facts used in D for this mapping. This process is in polynomial time overall.

What is more, the result is indeed the provenance of each tuple, as it claims that we have one of the sets of facts that allows the query to match.

Question 9. Open question: Some informations usually tracked by semiring provenance are not tracked in the definition of provenance proposed above, in particular the *number of uses of a given fact* and the *number of ways to use a set of facts.*

Explain these shortcomings by showing examples of queries and documents where some answer tuples has the same provenance in the sense above, but should have a different provenance if we accounted for these additional informations.

Propose ways in which we could define an extended notion of provenance for SPARQL (or some fragment thereof) to capture these additional informations, and explain how they could be computed.

Which SPARQL features would be problematic for this extended provenance notion?

Answer. For the number of uses of a given fact, consider the query Q:

```
SELECT ?x WHERE {
  ?x <m:follows> ?z .
  ?z <m:follows> ?y .
}
```

and the query Q':

```
SELECT ?x WHERE {
  ?x <m:follows> ?y .
}
```

and the singleton set D of RDF facts:

```
<m:a> <m:follows> <m:a> . # x1
```

For the answer (a), both queries have x_1 as provenance, but the first query is using the fact twice, and the second one only once. Intuitively, the first query should have the provenance x_1^2, and the second query should have provenance x_1.

For the number of ways to use a set of facts, we can consider:

```
SELECT ?x WHERE {
  ?x <m:follows> ?y .
  ?x <m:follows> ?z .
  ?y <m:follows> ?z .
  ?z <m:follows> ?y .
}
```

and the same query with `?y` replaced by the constant `<m:b>`, on the set of RDF facts:

```
<m:a> <m:follows> <m:b> . # x1
<m:a> <m:follows> <m:c> . # x2
<m:b> <m:follows> <m:c> . # y1
<m:c> <m:follows> <m:b> . # y2
```

For both queries, the Boolean provenance of `<m:a>` is $x_1 \wedge x_2 \wedge y_1 \wedge y_2$ as we need all facts to get an answer. However, the first query has two ways to use this set of facts (mapping `?y` and `?z` to `<m:b>` and `<m:c>` or vice-versa), whereas the second query has only one way to do so. Intuitively, the second query should have as provenance $x_1 x_2 y_1 y_2$, and the first one $2x_1 x_2 y_1 x_2$.

These notions can be formalized, and generalized (for SPARQL queries whose body only contains RDF facts) to a notion of *provenance polynomial* where, for any monomial on the set of variables X, the coefficient of this monomial in the polynomial is the number of ways to map the existential variables of the query such that we can obtain the indicated output by mapping the facts of the query body precisely to this multiset of facts in the RDF document. For a fixed query, this notion of provenance can still be computed in PTIME as in the previous question, by considering all possible matches of all query variables.

RDF features that make the query use an unbounded number of facts in the matches (in an unbounded number of ways), in particular property paths, do not work under this definition: for the document D above, for the query:

```
SELECT ?x WHERE {
  ?x <m:follows>* ?y .
}
```

the provenance should intuitively be $1 + x_1 + x_1^2 + \ldots$. Solutions to this problem could be proposed using formal power series.

Please write your answer to the second exercise on a separate sheet of paper.

Exercise 2: Web search and Big Data management (10 points)

Question 1. Explain the difference between *coverage* and *freshness*, as alternative objectives in a crawler strategy, and how they may impact that strategy.

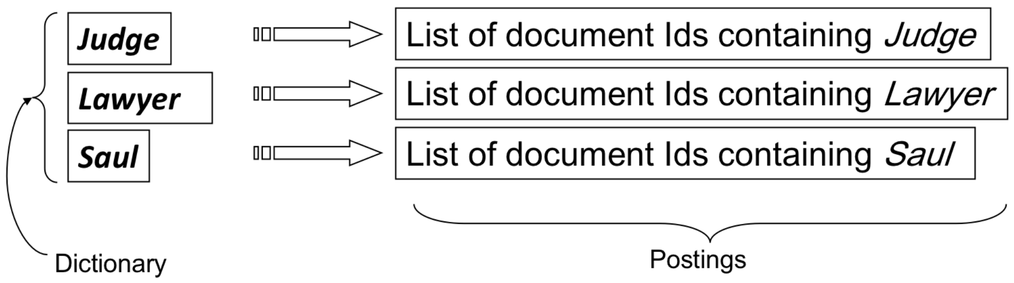
Question 2. Explain *binary freshness* in the context of a crawler, and name one drawback of this freshness indicator.

Question 3. Explain *link pollution* and how a search engine could avoid it.

Question 4. Explain why Web graphs must be compressed, and name two specificities of adjacency lists (of the Web graph) that can help in this regard.

Question 5. Explain why inverted indexes must be compressed, and explain one technique that can help in this regard.

Question 6. Boolean queries are particularly helpful/frequent for text retrieval in law research services. In a simple version of an inverted index designed to answer Boolean queries, we would keep, for each word (in the dictionary), the list of documents in which it appears (its *posting list*). Let us assume only document Ids are kept in posting lists, as in the following example:


Judge
Lawyer
Saul
List of document Ids containing Judge
List of document Ids containing Lawyer
List of document Ids containing Saul
Dictionary
Postings

1. Posting lists may be very large. Which condition is necessary in order to be able to process queries such as *Judge AND Saul* efficiently (say, in linear time in the size of the posting lists)?
2. What would be a good logic to process a query such as *Judge AND Saul AND Lawyer*?
3. Assuming that we only keep the dictionary in main memory, but not the posting lists, which additional information do we need to keep in the dictionary for the previous logic to work?
4. How could that logic be extended to handle OR queries, such as *Judge OR Lawyer*?

Question 7. Open question: We want to build a search engine for *sports* only. How would you do that (lay down the main changes with respect to a general search engine design)?

Question 8. To build a search engine for *sports*, how could we compute a topic-specific PageRank? Is the sports PageRank score of a page always smaller than the general PageRank score of that page?

Question 9. Explain the fault tolerance and crash recovery logic of a Big Data Management System.

Question 10. Name one reason why Spark can be considered as "Hadoop compatible", so that it can be appealing as an alternative data processing engine for Hadoop users.