— *Homework* —

Data Bases and Data Mining

Hugo SALOU

1 Exercise 2.3: *Window functions* [♥].

Question 1.1. *For each evaluation, return the name of the reviewer, the title of the film evaluated, the rating given to the film by the reviewer and the average of all the scores assigned by this reviewer.*

We can use the following query:

```
select name_reviewer, title_movie, stars_rating,
  avg(stars_rating) over (partition by id_reviewer) as average
from ratings natural join reviewers natural join movies
```

Question 1.2. *For each evaluator, give the number of films reported, the average rating they gave, the smallest and the best note. Order the result by the number of films reported.*

We can use the following query:

```
select name_reviewer, count(id_movie), avg(stars_rating),
  min(stars_rating), max(stars_rating)
from ratings natural join reviewers
group by id_reviewer
order by count(id_movie) desc
```

Question 1.3. *For each film, display its average rating and ranking rank in order of average ratings among all the films.*

We can use the following query:

```
with movie_avg as (
  select id_movie, avg(stars_rating) as avg_rating
  from ratings group by id_movie
)
select id_movie, avg_rating,
  rank() over (order by avg_rating desc) as ranking
from movie_avg
```

Question 1.4. *For each reviewer, and each film they rated, display the highest rating they gave to that film, as well as the average of all the ratings that were given to this film. The response will be ordered by reviewer name and movie title.*

We can use the following query:

```
select name_reviewer, title_movie, max(stars_rating) as max_rating,
  avg(stars_rating) over (partition by id_movie) as avg_rating
from ratings natural join reviewers natural join movies
group by id_reviewer, id_movie
order by name_reviewer, title_movie
```

2 Exercise 3.9 [♥].

Let $R(\mathrm{Att}_R), S(\mathrm{Att}_S)$ ***with*** $\mathrm{Att}_S \subseteq \mathrm{Att}_R$. ***Let*** $\Delta := \mathrm{Att}_R \setminus \mathrm{Att}_S$.

> **Note.** I also tried this exercise, by using Codd's theorem for the first question as $R \div S \equiv q_{\forall \mathrm{Att}_S, R \wedge S}$ (as the $\forall$ is transformed into a $\neg\exists\neg$). Unfortunately, I was not able to completely simplify the resulting query to the one expected.

Question 2.1. *Prove the identity* $R \div S = \pi_\Delta(R) \setminus \pi_\Delta\big((\pi_\Delta(R) \bowtie S) \setminus R\big)$.

I will say that a tuple $x \in D^\Delta$ is *invalid* when $xy \notin R$ for all $y \in S$. The set of invalid values is given by

$$P := \pi_\Delta((\pi_\Delta(R) \times S) \setminus R).$$

The tuples in the division $R \div S$ are exactly the valid tuples (the not invalid ones) of $\pi_\Delta(R)$. Thus:

$$R \div S = \pi_\Delta(R) \setminus P.$$

Question 2.2. *Prove the identity* $R \div S = \bigcap_{t \in S} \pi_\Delta(\sigma_{S=t}(R))$.

By definition,

$$R \div S := \left\{ u \in D^\Delta \;\middle|\; \forall t' \in S, \exists t \in R, t_{|\Delta} = u \text{ and } t_{|\mathrm{Att}_S} = t' \right\}.$$

We have that

$$\begin{aligned}\bigcap_{t\in S}\pi_\Delta(\sigma_{S=t}(R)) &= \left\{u\in D^\Delta \;\middle|\; \forall t\in S, u\in\pi_\Delta(\sigma_{S=t}(R))\right\}\\ &= \left\{u\in D^\Delta \;\middle|\; \forall t\in S, \exists t'\in\sigma_{S=t}(R), t'_{|\Delta}=u\right\}\\ &= \left\{u\in D^\Delta \;\middle|\; \forall t\in S, \exists t'\in R, t'_{|\Delta}=u \text{ and } t_{|\mathsf{Att}_S}=t'\right\}\\ &= R\div S.\end{aligned}$$

3 Exercise 3.10 [♥].

Define by induction a function* sch *that takes an relation algebra query and outputs the relation schema of a corresponding relation.

A relation schema is given by a relation name and its attributes' names. However, for queries, the relation name really does not matter, so we can always use a fresh relation name (except, for example, for R where we use R). So, I will only define the names of attributes in the output of a relational algebra query.

Define $\mathsf{sch}(q)$ by induction on query q:

- $\mathsf{sch}(\langle j:d\rangle) := \{j\}$;
- $\mathsf{sch}(R) := \mathsf{Att}_R$;
- $\mathsf{sch}(\sigma_{j=j'}(q)) := \mathsf{sch}(q)$;
- $\mathsf{sch}(\pi_I(q)) := I\cap\mathsf{sch}(q)$;
- $\mathsf{sch}(\rho_f(q)) := f(\mathsf{sch}(q)) = \{f(j)\mid j\in\mathsf{sch}(q)\}$;
- $\mathsf{sch}(q\cup q') := \mathsf{sch}(q) = \mathsf{sch}(q')$;
- $\mathsf{sch}(q\smallsetminus q') := \mathsf{sch}(q) = \mathsf{sch}(q')$;
- $\mathsf{sch}(q\bowtie q') := \mathsf{sch}(q)\cup\mathsf{sch}(q')$.

In the above, I assumed a well-written query (*e.g.* no $\langle j:d\rangle\cup\langle j':d'\rangle$ with $j\neq j'$).

4 Exercise 4.5: *Query evaluation* [♥].

In this exercise, we study the theoretical complexity of query evaluation.

Question 4.1. *Write an algorithm running in polynomial space for FO query evaluation.*

Firstly, assuming the database instance is available globally, we can use the following algorithm for FO query evaluation (algorithm 1).

Algorithm 1 FO query Evaluation

```
1:  procedure EVAL(λ, φ)
2:      if φ = R(τ_1, ..., τ_n) then return true iff (λ(τ_1), ..., λ(τ_n)) ∈ R^I
3:      else if φ = ψ ∨ ϑ then return EVAL(λ, ψ) || EVAL(λ, ϑ)
4:      else if φ = ψ ∧ ϑ then return EVAL(λ, ψ) && EVAL(λ, ϑ)
5:      else if φ = ¬ψ then return not EVAL(λ, ψ)
6:      else if φ = ∃x, ψ then
7:          for all v ∈ D do
8:              if EVAL(λ[x := v], ψ) = true then
9:                  return true
10:         return false
11:     else                ▷   φ = ∀x, ψ
12:         for all v ∈ D do
13:             if EVAL(λ[x := v], ψ) = false then
14:                 return false
15:         return true
```

Here λ is a valuation, and $\textsc{Eval}(\lambda, \phi)$ returns `true` iff $I, \lambda \vDash \phi$. Thus, for closed formulas, we simply have to use the "empty" valuation.

The space complexity of algorithm 1 is $\mathrm{O}(|\phi| \log |D|)$ as we simply consider one path in the evaluation tree (and we wait for the first path to finish before going to a different branch), and for the "**for all**" instructions, we can use binary counting, thus only requiring $\mathrm{O}(\log |D|)$ space.

Question 4.2. *Show that FO query evaluation is **PSPACE**-hard.*

To do that, we will use that the Quantified Boolean Formula problem (QBF) is **PSPACE**-hard and we will proceed by reduction.

Consider ϕ a quantified boolean formula, and consider the following database instance I:

- $D := \{\texttt{true}, \texttt{false}\}$;
- $\textsc{True}^I = \{(v : \texttt{true})\}$ with only one attribute v such that v does not occur (not even as a bound variable) in ϕ.

Then we transform ϕ into $\tilde{\phi}$ by replacing all occurrences of variables x by the relational algebra query $\textsc{True}(v : x)$. This can be done in polynomial time. We have the following equivalence:

$$I, \lambda \vDash \tilde{\phi} \qquad \Longleftrightarrow \qquad \lambda \vDash \phi,$$

which can be proven by a simple induction on ϕ. The only interesting case in this induction is for variables and we have

$$I, \lambda \vDash \underbrace{\textsc{True}(v : x)}_{\tilde{x}} \Longleftrightarrow (v : \lambda(x)) \in \textsc{True}^I \Longleftrightarrow \lambda(x) = \texttt{true} \Longleftrightarrow \lambda \vDash x.$$

We can thus conclude by reduction that first-order query evaluation is **PSPACE**-hard, and thus, **PSPACE**-complete with the first question.

5 Exercise 4.7: *Trakhtenbrot's theorem* [♥♥].

We consider the halting problem of a deterministic Turing machine on the empty word ε:

$\textsc{Halt}_\varepsilon$	**Input.** A deterministic Turing machine M **Output.** Does M halt on input ε?

Theorem 1. The SATRC problem is undecidable where

SATRC	**Input.** A closed first-order formula ϕ **Output.** Does ϕ admit a finite model?

Question 5.1. *Explain which reduction to apply for proving the Trakhtenbrot's theorem (give a diagram if that helps).*

We will construct a reduction $\textsc{Halt}_\varepsilon \leq \textsc{SatRC}$:

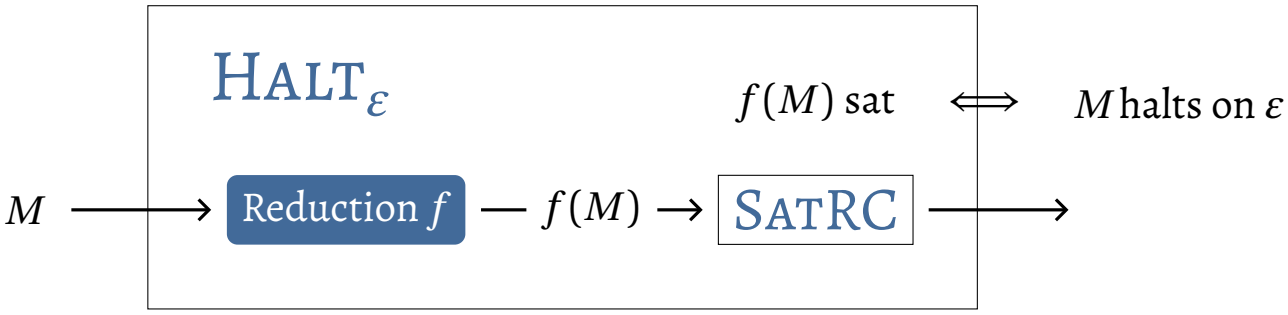

Figure 1 | *Diagram for a reduction* $\textsc{Halt}_\varepsilon \leq \textsc{SatRC}$

In figure 1, we write "$f(M)$ sat" to say that first-order formula $f(M)$ admits a finite model.

Question 5.2. *Write a formula saying that the relation symbol $<$ is interpreted as a strict linear order $<^I$.*

> **Note.** In the rest, I will use $\Rightarrow$ to denote implication. It'll behave greedily on the right (like $\forall$ and $\exists$), but not on the left. For example,
>
> $$a \wedge b \Rightarrow c \vee \exists u, d(u)$$
>
> means
>
> $$a \wedge (b \Rightarrow (c \vee (\exists u, d(u)))).$$

Writing $x \not< y$ for $\neg(x < y)$, we define the following formulae:

- ▷ irreflexive $:= \forall x, x \not< x$,
- ▷ asymmetric $:= \forall a\, b, a < b \Rightarrow b \not< a$,
- ▷ transitive $:= \forall a\, b\, c, a < b \wedge b < c \Rightarrow a < c$,
- ▷ linear $:= \forall x\, y, x < y \vee x = y \vee x > y$,

and finally define

$$\text{strict-linear-order} := \underbrace{\text{irreflexive} \wedge \text{asymmetric} \wedge \text{transtive}}_{\text{strict-order}} \wedge \text{linear}.$$

Question 5.3. *Write a formula saying that the constant symbol* 0 *represents the minimum of* $<$.

We can use the formula:

$$\text{minimum} := \forall x, \underbrace{x \neq 0}_{\neg(x=0)} \Rightarrow 0 < x.$$

Question 5.4. *We now introduce the relation symbols* $T_0(\cdot,\cdot)$, $T_1(\cdot,\cdot)$, $H_q(\cdot,\cdot)$ *for all states* q *of* M. *The intuitive meaning of* $T_i(x,t)$ *is that there is the letter* i *at cell* x *at time* t. *The predicate* $H_q(x,t)$ *means that the machine is in the state* q *at time* t *and the head is at position* x *at time* t. *Write a formula saying that there is an terminating execution of* M *on* ε.

For every transition $\tau := [(q,a) \to_M (q',a'), m]$,[a] we define the formula:

$$\text{move}(t,\tau) := \text{update}(t,\tau) \wedge \text{keep-others}(t,\tau)$$

where

- $\text{update}(t,\tau) := \forall x, H_q(x,t) \wedge T_a(x,t) \Rightarrow T_{a'}(x,t+1) \wedge H_q(m(x),t+1)$
- $\text{keep-others}(t,\tau) := \forall x, \neg H_q(x,a) \Rightarrow \bigwedge_{\tilde{a} \in \{0,1,\square\}} T_{\tilde{a}}(x,t) \Leftrightarrow T_{\tilde{a}}(x,t+1)$
- $T_\square(x,t) := \neg T_0(x,t) \wedge \neg T_1(x,t)$.

The intuitive meaning of these formulae is: formula update updates the state of the Turing machine and the value on the tape at the head's position; and formula keep-others states that all other cells on the tape are preserved (and no new cell is created).

We will define a formula

$$\text{valid}(t) := \text{head}(t) \wedge \text{head-unique}(t) \wedge \forall x, \text{cell-valid}(t,x)$$

where

[a] This notation means that M was in state $q \in Q$ and its head read $a \in \{0,1,\square\}$ and that, after the transition, M will write $a' \in \{0,1,\square\}$ at the head position, then move its head according to $m \in \{x \mapsto x-1, x \mapsto x+1\}$, and finally switch to state $q' \in Q$.

- $\mathsf{head}(t) := \exists x, \bigvee_{q \in Q} H_q(x,t)$;
- $\mathsf{head\text{-}unique}(t) := \forall x\, x', \bigwedge_{q,q' \in Q} H_q(x,t) \wedge H_{q'}(x',t) \Rightarrow x = x' \wedge \mathsf{eq}_{q,q'}$;
- $\mathsf{eq}_{q,q'} := \begin{cases} \top & \text{if } q = q' \\ \bot & \text{otherwise;} \end{cases}$
- $\mathsf{cell\text{-}valid}(x,t) := \neg\big(T_0(x,t) \wedge T_1(x,t)\big)$.

The meaning of valid is to ensure that the Turing machine stays in a valid state:

- the Turing machine is in some state q and the head is somewhere on the tape (ensured by $\mathsf{head}(t)$),
- the Turing machine cannot be in two states at once nor have a head at two different positions (ensured by $\mathsf{head\text{-}unique}(t)$),
- the cells on the tape can either be 0, 1 or $\square$, and cannot be have both 0 and 1 written (ensured by $\mathsf{cell\text{-}valid}(x,t)$).

We also define a formula

$$\mathsf{init} := H_{q_{\text{initial}}}(0,0) \wedge \forall x, T_{\square}(x,0)$$

defining the initial state of the Turing machine.

We used $x+1, x-1$ previously (with functions $(-)+1$ and $(-)-1$), but we need to define a few axioms saying that they behave according to $+_{\mathbb{N}}$ and $-_{\mathbb{N}}$:

$$\mathsf{coh}_+ := \forall x, x < x+1 \wedge \forall y, x < y \Rightarrow x+1 \leq y$$

and

$$\mathsf{coh}_- := \big(\forall x, x \neq 0 \Rightarrow x-1 < x \wedge \forall y, y < x \Rightarrow y \leq x-1\big) \wedge 0-1 = 0,$$

where we write $x \leq y$ for $x < y \vee x = y$. These definitions come from:

$$x +_{\mathbb{N}} 1 = \min\{y \mid x < y\} \qquad 0 -_{\mathbb{N}} 1 = 0 \qquad \underbrace{x -_{\mathbb{N}} 1 = \max\{y \mid y < x\}}_{\text{if } x \neq 0}.$$

Note. Rereading the course notes, it seems that we do not allow functions in terms. But, there is an easy fix: whenever we have $T_a(x, t+1)$, it suffices to replace $t+1$ by a fresh variable u, and to append the coherence rule for $(-)+1$ specialized at x/u:

$$T_a(x, t+1) \qquad \rightsquigarrow \qquad \exists u, T_a(x, u) \wedge \widetilde{\mathsf{coh}_+}[t, u],$$

where $\widetilde{\mathsf{coh}_+}[t, u] = \mathsf{coh}_+[t, u] \wedge \forall v, v \neq u \Rightarrow \neg \mathsf{coh}_+[t, u]$. Formula $\mathsf{coh}_+[t, u]$ is obtained by removing the $\forall x$ and replace x by t and $x+1$ by u in the rest of the formula.

Intuitively, $\mathsf{coh}_+[t, u]$ says that u behaves like $t+1$, and $\widetilde{\mathsf{coh}_+}[t, u]$ states that u is the unique element behaving like $t+1$.

We can finally define a formula saying that there is an execution of M on ε that halts.

$$\text{exec-halt} := \exists T, \text{init} \wedge \text{make-moves}(T) \wedge \text{all-valid}(T) \wedge \text{halts}(T)$$

where (in the following, τ, τ' are transitions of M)

- ▷ $\text{make-moves}(T) := \forall t, t < T \Rightarrow \text{has-move}(t) \wedge \text{move-unique}(t)$;
- ▷ $\text{has-move}(t) := \bigvee_\tau \text{move}(t, \tau)$;
- ▷ $\text{move-unique}(t) := \bigwedge_{\tau,\tau'} \text{move}(t, \tau) \wedge \text{move}(t, \tau') \Rightarrow \text{eq}_{\tau,\tau'}$;
- ▷ $\text{all-valid}(T) := \forall t, t \leq T \Rightarrow \text{valid}(t)$;
- ▷ $\text{halts}(T) := \neg \bigvee_\tau \text{move}(T, \tau)$.

Question 5.5. *Conclude.*

We define

$$f(M) := \text{strict-linear-order} \wedge \text{minimum} \wedge \text{exec-halt}$$

and we have the equivalence:

$$f(M) \text{ admits a finite model} \iff M \text{ has a execution that halts on } \varepsilon.$$

The fact that we have a *finite* model tells us that we do not consider an execution whose length is infinite.

To prove " $\Longrightarrow$ ", we would have to prove that all the previous formulae have the correct behaviour in a finite model, and thus we can use the sequence of moves satisfying make-moves, to obtain the execution.

To prove " $\Longleftarrow$ ", we would have to show that $[\![0, T]\!]$ is a finite model of $f(M)$ where T is the length of the execution that halts on ε.

This concludes the reduction $\textsc{Halt}_\varepsilon \leq \textsc{SatRC}$, and as $\textsc{Halt}_\varepsilon$ is undecidable, so is SatRC.

6 Exercise 5.9 [♥].

Prove that testing that a FO query is domain independent is undecidable.

Firstly, we can clearly see that both problems DomDep and DomInd are equivalent: if one is decidable, so is the other (we simply negate the output). Thus we will show that DomDep is undecidable, proving the undecidability of DomInd in the process.

DomInd	**Input.** A database query q **Output.** Is q domain independent?

DomDep	**Input.** A database query q **Output.** Is q domain dependent?

We will proceed by reduction to show that $\textsc{SatRC} \leq \textsc{DomDep}$. Consider a closed first-order formula ϕ. Define $f(\phi) := q_\phi \bowtie \pi_x(x = y)$ where q_ϕ is defined in the proof of Codd's theorem (the definition of q_ϕ does not use the hypothesis of domain-independence: it is only used in the proof that ϕ and q_ϕ have the same semantics).

Let us show

$$\phi \text{ admits a finite model} \iff f(\phi) \text{ is domain dependent.}$$

Firstly, we will show the implication " $\Longrightarrow$ ".

- If ϕ is satisfiable, then $[\![f(\phi)]\!]^I = \{(x : v) \mid v \in I\}$ for all I satisfying ϕ, and $[\![f(\phi)]\!]^I = \emptyset$ otherwise.
- If ϕ is unsatisfiable, then for all I, $[\![f(\phi)]\!]^I = \emptyset$. Thus $f(\phi)$ is domain independent.

Secondly, we show the opposite implication " $\Longleftarrow$ ".

- If ϕ is domain dependant, then necessarily there is an instance satisfying q_ϕ and thus ϕ.
- If ϕ is domain independent, then necessarily q_ϕ is always false, thus ϕ does not admit a finite model.

Thus SatRC $\leq$ DomDep and SatRC is undecidable thus so is DomDep, and so is DomInd.