TP 05 - Dependencies and the Chase

17th March 2024

Exercise 1. We apply the Chase algorithm to D, denote $D' := \text{chase}(\Sigma, D)$. We claim that this algorithm run in polynomial time:

- v is the number of values in the database. We iterate only on tables appearing in Σ. For the i-th table, we add at most v^{k_i} values, where k_i is the arity of T_i. We know that k_i is constant since T_i is a table appearing in a formula of Σ which is of fixed size.
- One can compute a homomorphism in polynomial time. Therefore, one can add an atom in D in polynomial time.
- Furthermore, we add at most $\sum_i v^{k_i}$ atoms in the database, which is polynomial in D.
- One can check that an atom does not appear in the database in linear time in D.

Finally, one can check whether $D' \models q$ holds in polynomial time in D': we compute the set $\{\, \varphi(\overline{z}) \mid \overline{z} \in D \,\}$ and check whether it is not empty. (We assume that q is of the form $q := \exists \overline{z}, \varphi\left(\overline{z}\right)$).

Exercise 2. Datalog is in ExpTime using the same majoration as in the previous exercise; the difference is that the k_i are not constant anymore, which makes it ExpTime.

Let's prove that Datalog is ExpTime-complete by showing that a deterministic Turing machine running in time $O(2^{n^k})$ for a constant k can be simulated by a Datalog program. Let $M = (Q, \Gamma, \delta, q_0, F)$ be a deterministic Turing machine running in time $O(2^{n^k})$ for some constant k.

Tables To encode the given Turing machine in a Datalog program, we introduce the following tables:

- A table H of arity $2 \times n^k$, representing the head of the machine. $H(s, p)$ holds when the head is in position p after s steps, where p and s are represented in big-endian binary over n^k bits (since the machine runs in time $O(2^{n^k})$, p and s are at most 2^{n^k} and can therefore be represented over n^k bits).
- For each state $q \in Q$, we introduce a table Q_q of arity $1 \times n^k$, representing the states. $Q_q(s)$ holds when the machine is in the state q after s steps (q and s are also represented in binary)
- For each symbol $\gamma \in \Gamma$, we introduce a table S_γ of arity 2, representing the symbols. $S_\gamma(s, c)$ holds when the tape cell in position c contains the symbol γ after s steps (s and c are also represented in binary).

Handling numbers We now need a way to manipulate binary numbers represented over n^k bits. Sadly, we cannot simply build a table for the relationship of numbers over n^k digits since this would require more than polynomial time in the input size when n^k is very large. To fix this issue, we will use tables and dependencies to do the computation for us, starting from numbers of length 1 (which we can initialize in constant time) and building up a recurring scheme toward n^k. We introduce three tables for each number length $l \in [\![1, n^k]\!]$:

- A table Z_l (for zero) of arity l such that $Z_l(z)$ holds when z is the zero of length l.
- A table M_l (for max) of arity l such that $M_l(m)$ holds when m represents $2^l - 1$, the maximum value represented over l digits.
- A table P_l (for plus 1) of arity $2 \times l$ such that $P^l(a, b)$ holds when $b = a + 1$ (with a and b of length l).

Tables for $l = 1$ are initialized with:

$$\begin{cases} Z_1(0) \\ M_1(1) \\ P_1(0,1) \end{cases}$$

We add the following dependencies:

$$\begin{aligned} Z_i(z) &\to Z_{i+1}(0, z) \\ M_i(m) &\to M_{i+1}(1, z) \\ P_i(a, b) &\to P_{i+1}(0, a, 0, b) \\ P_i(a, b) &\to P_{i+1}(1, a, 1, b) \\ M_i(a) \wedge Z_i(b) &\to P_{i+1}(0, a, 1, b) \end{aligned}$$

Finally, we add a table G (for *greater than*) of arity $2 \times n^k$ comparing two number over n^k bits. We introduce the dependencies:

$$\begin{aligned} \text{True} &\to G(a, a) \\ G(a, b) \wedge P_{n^k}(b, c) &\to G(a, c) \end{aligned}$$

corresponding to reflexivity and transitivity (G is the reflexive and transitive closure of P_{n^k}).

Initial configuration of the Turing machine Let $\gamma_1, \ldots, \gamma_n \in \Gamma^n$ be the input word of the Turing machine. We denote by i_{n^k} the representation over n^k bits of the number i, in the form of a tuple of n^k binary variables. To represent the initial configuration of the Turing machine, we add the following entries to the tables:

$$\begin{cases} Q_{q_0}(0_{n^k}) \\ H(0_{n^k}, 0_{n^k}) \\ S_{\gamma_i}(0_{n^k}, i_{n^k}) & \text{forall } i \in [\![1, \ldots, n]\!] \\ G((n+1)_{n^k}, c) \to S_{\sqcup}(0_{n^k}, c) \end{cases}$$

Transitions Recall that $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$. Foreach $q \in Q, \gamma \in \Gamma$, we note $\delta(q, \gamma) =: (q', \gamma', f)$ we add the dependencies:

$$\underbrace{P_{n^k}(s, s')}_{s'=s+1} \wedge \underbrace{H(s,p) \wedge S_\gamma(s,p) \wedge Q_q(s)}_{\text{TM is in state } q \text{ and head is in position } p \text{ at step } s} \longrightarrow \underbrace{S_{\gamma'}(s', p)}_{\text{then } \gamma \text{ is replaced by } \gamma' \text{ at step } s+1}$$

$$\underbrace{P_{n^k}(s, s') \wedge H(s,p) \wedge S_\gamma(s,p) \wedge Q_q(s)}_{\text{idem}} \longrightarrow \underbrace{Q_{q'}(s')}_{\text{then the TM is in state } q \text{ at step } s+1}$$

We do the same for setting the head position. If $f = \rightarrow$, we add the dependency:

$$\underbrace{P_{n^k}(s, s')}_{s'=s+1} \wedge \underbrace{P_{n^k}(p, p')}_{p'=p+1} \wedge \underbrace{H(s,p) \wedge S_\gamma(s,p) \wedge Q_q(s)}_{\text{as previously}} \longrightarrow \underbrace{H(s', p')}_{\text{then the head is in position } p+1 \text{ at step } s+1}$$

If $f = \leftarrow$, we add a similar dependency instead:

$$\underbrace{P_{n^k}(s,s')}_{s'=s+1} \wedge \underbrace{P_{n^k}(p',p)}_{p'=p-1} \wedge \underbrace{H(s,p) \wedge S_\gamma(s,p) \wedge Q_q(s)}_{\text{as previously}} \longrightarrow \underbrace{H(s',p')}_{\text{then the head is in position } p-1 \text{ at step } s+1}$$

For the sake of the length of the proof, we assume that the machine never moves its head to a position $p < 0$, which guarantees that such a p' always exists.

Finally, for each step, we need to copy the symbols which are not changed by the machine, to make sure that they are still defined at the next step. This is done using two rules, for each $\gamma \in \Gamma$:

$$\underbrace{P_{n^k}(s,s')}_{s'=s+1} \wedge \underbrace{S_\gamma(s,p)}_{\gamma \text{ is written in } p} \wedge \underbrace{H(s,p') \wedge P_{n^k}(p',p'') \wedge G(p'',p)}_{\text{head is in } p' \text{ such that } p'<p\ (p'+1=p''\leqslant p \implies p'<p)} \longrightarrow \underbrace{S_\gamma(s',p)}_{\gamma \text{ stays in } p}$$

This dependency guarantees that cells on the left of the head are re-written. We do the same for cells on the right:

$$\underbrace{P_{n^k}(s,s')}_{s'=s+1} \wedge \underbrace{S_\gamma(s,p)}_{\gamma \text{ is written in } p} \wedge \underbrace{H(s,p') \wedge P_{n^k}(p'',p') \wedge G(p,p'')}_{\text{head is in } p' \text{ such that } p'>p\ (p'-1=p''\geqslant p \implies p'>p)} \longrightarrow \underbrace{S_\gamma(s',p)}_{\gamma \text{ stays in } p}$$

Accepting Finally, we introduce a table A (for *accepting*) of arity 0, such that $A()$ holds when the Turing machine accepts. We connect it to the other tables using the following dependencies, for each $q_f \in F$:

$$Q_{q_f}(s) \to A()$$

The number of dependencies is polynomial in the input, therefore the problem can be polynomially reduced. Therefore, Datalog is ExpTime-hard.

Exercise 3.

1. $\{C\} \to \{B\}$ can be rewritten as:

$$\forall a, a', b, b', c, d, d', e, e', f, f', \quad R(a,b,c,d,e,f) \wedge R(a',b',c,d',e',f') \to b = b'$$

2. $\{D,E\} \to \{F\}$ can be rewritten as:

$$\forall a, a', b, b', c', d, e, f, f', \quad R(a,b,c,d,e,f) \wedge R(a',b',c',d,e,f') \to f = f'$$

3. $\{A\} \twoheadrightarrow \{C,F\}$ can be rewritten as:

$$\forall a, b, b', c, c', d, d', e, e', f, f', \quad R(a,b,c,d,e,f) \wedge R(a,b',c',d',e',f') \to R(a,b',c,d',e',f)$$

Exercise 4. Let (a,b,c,d,e,f) be a tuple over R. During the decomposition of R, it is split into (a,b,c,d',e',f') and (a,b',c',d,e,f). We will now show that using the dependencies of Σ, one can deduce from $R(a,b,c,d',e',f')$ and $R(a,b',c',d,e,f)$ that $R(a,b,c,d,e,f)$. We start with:

R					
a	b	c	d'	e'	f'
a	b'	c'	d	e	f

Applying $\{A\} \twoheadrightarrow \{C,F\}$ to the first two tuples, we obtain:

R

a	b	c	d'	e'	f'
a	b'	c'	d	e	f
a	b	c'	d'	e'	f
a	b'	c	d	e	f'

Then, we apply $\{D, E\} \to \{F\}$ to the pair of tuples (a, b, c, d', e', f') and (a, b, c', d', e', f), which yield $f = f'$. This gives the following updated table:

R

a	b	c	d'	e'	f
a	b'	c'	d	e	f
a	b'	c	d	e	f
a	b	c'	d'	e'	f

Finally, we apply $\{C\} \to \{B\}$ to the pair of tuples (a, b, c, d', e', f) and (a, b', c, d, e, f), which yield $b = b'$. This gives the following updated table:

R

a	b	c	d'	e'	f
a	b	c'	d	e	f
a	b	c	d	e	f
a	b	c'	d'	e'	f

We have reached a fixed point and the tuple (a, b, c, d, e, f) is in the table. Therefore, the decomposition is lossless.

Exercise 5. If a set of TGDs Σ is Datalog, then its TGDs feature no existentially quantified variables. Therefore, the graph associated with Σ contains no special edge at all, which shows that it is weakly acyclic.

Exercise 6. Let t be the following TGD :

$$t : \forall \overline{x} \forall \overline{y},\ B[\overline{x}, \overline{y}] \to \exists \overline{z},\ H[\overline{x}, \overline{z}]$$

Suppose that during a chase we apply the TDG t with a homomorphism h. This leads to the introduction of new tuples in the table, with fresh values $\overline{v}$ for all the variables in $\overline{z}$.

If we apply the same TDG with a homomorphism h' such that:

$$\forall v \in \overline{x} \cup \overline{y},\ h(v) = h'(v)$$

The variables in $\overline{z}$ can be unified with the values $\overline{v}$ introduced earlier. No tuples are added to the table and therefore one cannot apply two times the same TGD with the same homomorphism during a chase.

Exercise 7. Let's show that C has a bounded active domain. First, only the TGD with a head containing a position in C can introduce new values its active domain. That's why we will only consider shuch TGD in the following.

In such TGD, the positions occurring in their body only occurs in $P \cup C$. Thus, during the application of a TGD t, two cases are possible :

- All the position occurring in the body of t is in P. Since P has a bounded active domain, we can only find a finite number of homomorphisms in P to apply t. Moreover, t can only be applied once per homomorphism (as we saw earlier). Thus, only a finite number of new values are introduced by such TGDs.
- The body of t contains position occurring in C. Since C is a subset of a weakly acyclic set of TGDs, the introduction of fresh values by t cannot lead to another application of itself, (otherwise, the graph would contain a cycle with a special edge). Thus, combined with the fact that TGDs cannot be applied twice with the same homomorphism. One can only add a finite number of values to the position in C by applying such TGDs.

Finally, if P has a bounded active domain then C does too.

Exercise 8. Let Σ be a weakly acyclic set of TGDs. The associated graph of Σ can be split into strongly connected component $(\sigma_i)_{i\in[\![0;n]\!]} \in \Sigma^n$. Moreover, theses component form a directed acyclic graph. So, one can find a topological ordering $\leqslant_\Sigma$ over theses strongly connected components.

We suppose without any loss of generality that :

$$\sigma_0 \leqslant_\Sigma \sigma_1 \leqslant_\Sigma \cdots \leqslant_\Sigma \sigma_{n-1} \leqslant_\Sigma \sigma_n$$

This means that for any $i \in [\![0;n]\!]$ all predecessors[1] of σ_i are in $\bigcup_{j\in[\![0;i[\![} \sigma_j$. By induction, all position occurring in Σ have a bounded active domain. Indeed :

- The set of predecessors of σ_0 is empty. Since the chase algorithm begin with a finite number of values for each position, with the previous question, σ_0 has a bounded active domain.
- If, for one $i \in [\![0;n]\!]$, all the previous sets $(\sigma_i)_{i\in[\![0;i[\![}$ have a bounded active domain, then with the previous question, σ_i has too.

Finally, the chase of Σ terminates in a finite amount of time because only a finite number of values are introduced by the TGDs.

[1] as defined in the previous question