Foundations of Databases

Complexity of Query Languages

Free University of Bozen – Bolzano, 2004–2005

Thomas Eiter

Institut für Informationssysteme

Arbeitsbereich Wissensbasierte Systeme (184/3)

Technische Universität Wien

`http://www.kr.tuwien.ac.at/staff/eiter`

(revised 2)

Issues

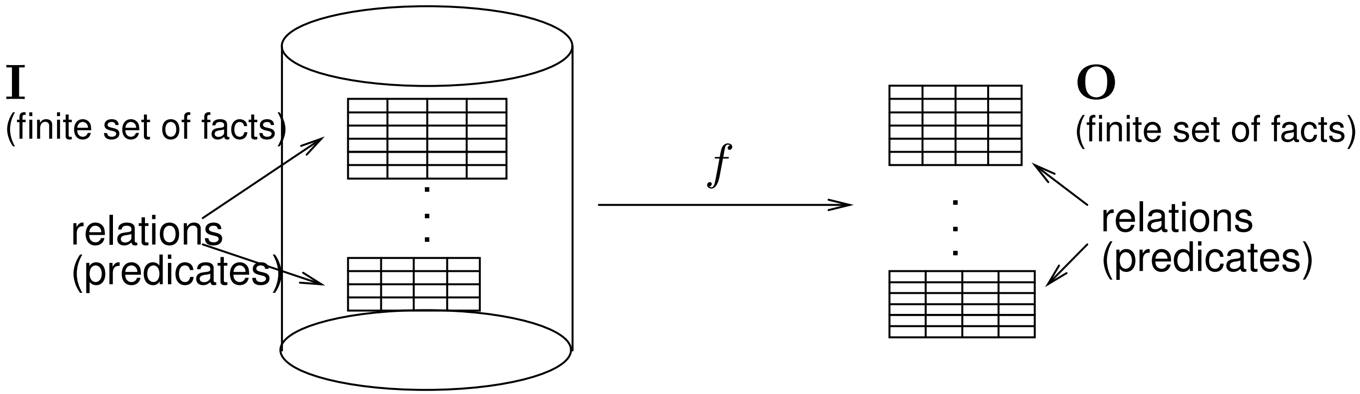
- How difficult is it to evaluate queries?
- For this, we must take into account:
  - How are query inputs / outputs represented ?
  - How difficult is the *intrinsic* complexity of a query, viewed as an abstract mapping ?
  - How difficult is the evaluation of a query defined by some query expression in some query language ?
  - What kind of use cases are of interest ?

Database Queries

In an abstracting view, a *database query* can be viewed as a (partial) function

$$f: \quad inst(\mathbf{EDB}) \longrightarrow inst(\mathbf{ODB})$$

such that every constant occurring in $f(\mathbf{I})$ occurs also in $\mathbf{I}$.



- Intuitively, f assigns to every input database $\mathbf{I}$ output relations, in which no new constant may occur.
- Usually, $\mathbf{ODB}$ consists of a single relation

Examples

- **Transitive closure Query:** Directed graph $G = (N, E)$
  - **EDB** = $\{ e \}$ (e binary)
  - **ODB** = $\{ tc \}$ (tc binary)

  Assumption: the vertex set N of G is implicit through e

$$f_{TC} : \quad inst(\mathbf{EDB}) \rightarrow inst(\mathbf{ODB}),$$
$$f_{TC}(\mathbf{I}) = \quad \{tc(a, b) \mid \exists \text{ path from } a \text{ to } b \text{ in graph } G \text{ with edges } \mathbf{I}(e)\}$$

Examples /2

- **"Even-Query":** A Boolean (Yes/No) query
  - **EDB** = $\{\, p \,\}$, p unary,
  - **ODB** = $\{even\}$, $even$ is 0-ary (a propositional atom)

$$f_{Even} : inst(\mathbf{EDB}) \rightarrow inst(\mathbf{ODB}),$$

$$f_{Even}(\mathbf{I}) = \begin{cases} \{even\}, & \text{if } |\mathbf{I}(p)| \text{ is even,} \\ \{\}, & \text{if } |\mathbf{I}(p)| \text{ is odd.} \end{cases}$$

Note: for propositional atom a, $a \in \mathbf{J}$ is equivalent to $\mathbf{J}(a) = \{\langle\ \rangle\}$

Examples /3

- **Graph 3-Uncolorability Query**: Boolean Query (Un)directed graph $G = (N, E)$
  - **EDB** = $\{\ e\ \}$ (e binary)
  - **ODB** = $\{\ uncol\ \}$ (tc 0-ary)

  Assumption: the vertex set N of G is implicit through e

$$f_{3uncol} : inst(\mathbf{EDB}) \rightarrow inst(\mathbf{ODB}),$$

$$f_{3uncol}(\mathbf{I}) = \begin{cases} \{uncol\}, & \text{graph } G \text{ with edges } \mathbf{I}(e) \text{ has no 3-coloring,} \\ \{\}, & \text{otherwise.} \end{cases}$$

Data Independence

- Databases should provide abstract interfaces and hide internal representation of the data (i.e., how they are stored).
- This is known as the *logical data independence principle*
- At the level of queries, this is formalized by the notion of *genericity*.

Defn. A query f is *generic*, if it commutes with automorphisms χ on $\mathbf{dom}$ (that, is renamings $\chi(c)$ of the elements c in $\mathbf{dom}$), i.e.,

$$f(\chi(\mathbf{I})) = \chi(f(\mathbf{I})), \quad \text{for every } \mathbf{I} \in inst(\mathbf{EDB})$$

$$\boxed{\begin{array}{ccc} \mathbf{I} & \xrightarrow{f} & f(\mathbf{I}) \\ \downarrow \chi & & \downarrow \chi \\ f(\mathbf{I}) & \xrightarrow{f} & f(\chi(\mathbf{I})) = \chi(f(\mathbf{I})) \end{array}}$$

Example

- The queries f_{TC}, f_{Even} are generic.
- The query which selects all facts from $\mathbf{I}$ containing the constant $'Jeff'$ is not generic.

Constants in queries can be treated differently:

- By relaxing genericity to C =*genericity*, $C \subseteq \mathbf{dom}$, which requests that χ is the identity on C
- By moving constants from query expressions to new designated input relations.

  Example: $\{x \mid G(a,x)\} \rightsquigarrow \{x \mid \exists y G(y,x) \wedge C_a(y)\}$, provide in the input $C_a = \{\langle a \rangle\}$.

Computability

- A further requirement for a query f is *computability*, in terms of a Turing machine M.
- For each "input" $\mathbf{I} \in inst(\mathbf{EDB})$,
  - M does not terminate if $f(\mathbf{I})$ is undefined.
  - M halts on input $\mathbf{I}$ with output $\mathbf{O} = f(\mathbf{I})$ on its tape if $f(\mathbf{I})$ is undefined.
- A problem to detail here is how $\mathbf{I}$ and $\mathbf{O}$ are represented on the tape of a Turing machine.

Notation:

- $\mathbf{Q}^{\star}$ denotes the collection of all computable queries
- $\mathbf{Q}$ denotes the collection of all generic computable queries

Database Instance Representation

- Any database instance $\mathbf{I}$ of a schema $\mathbf{R} = \{R_1, \ldots, R_m\}$ must for a Turing Machine be represented by a string $enc(\mathbf{I})$
- There are different possibilities
- They are based on encodings enc of the constants $\mathbf{dom} = \{c_0, c_1, c_2, \ldots\}$ to binary strings (e.g., $enc(c_i)$ is i in binary, with no leading bits)
  - For tuples, $enc(\langle a_1, \ldots, a_k \rangle)$ is e.g. $[enc(a_1), \ldots, enc(a_k)]$
  - For a relation $R \in inst(\mathbf{R})$, $enc(\mathbf{I}(R))$ is e.g. $enc(t_1), \ldots, enc(t_k)$, where $t_1, \ldots, t_k$ are the tuples in R in lexicographic ordering
  - Finally, $enc(\mathbf{I}(R))$ = $enc(\mathbf{I}(R_1)); \ldots; enc(\mathbf{I}(R_m))$

Enumeration of the domain

- Notice: Above, we assumed that there is an enumeration of $\mathbf{dom}$
- Different enumerations α, α' will yield different encodings enc_α, $enc_{\alpha'}$
- Under genericity, the particular enumeration α of $\mathbf{dom}$ ($= \{c_0, c_1, c_2, \ldots\}$) is not relevant.
- Thus in particular, wlog for a generic query the active domain consists of $C_n = \{c_0, c_1, \ldots, c_n\}$ represented by $0, 1, 2, \ldots, n$ (in binary)
- Relations over C_n are also often stored as bitmaps (serialized 0-1 matrices)

Example: $G = \{\langle 0, 0\rangle, \langle 1, 2\rangle, \langle 2, 0\rangle\}$ on C_2

0-1 matrix $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$ serialized (row by row): 100|001|100

Query Language

Defn. A *(database) query language* is a pair

$$\mathcal{L} = \langle Exp, \mu \rangle,$$

where

- Exp is a set of expressions E in a formal language (the *query expressions*)
- $\mu : Exp \longrightarrow \mathbf{Q}^\star$ is a *meaning function*, which assigns every $E \in Exp$ a database query $\mu(E)$

Remark: For $\mathcal{L}$ being effective, it is required that valid query expressions $E \in Exp$ can be recognized by a (fast) algorithm, and that $\mu(E)$ is a computable from E.

Examples

- $\mathcal{L}$ = Relational Algebra
  - Exp consists of all tuples $\langle \mathbf{R}, O, E \rangle$ where E is an expression in Relational Algebra over relations from $\mathbf{R}$
  - $\mu(\langle \mathbf{R}, O, E \rangle)$ is the query mapping

$$\mu(\langle \mathbf{R}, O, E \rangle) : inst(\mathbf{R}) \rightarrow inst(\{O\})$$

such that $\mathbf{I}$ is mapped to the result of $E(\mathbf{I})$.

Examples /2

- $\mathcal{L}$ = datalog:
  - Exp consists of all tuples $\langle \mathbf{R}, O, P \rangle$ where P is a datalog program with associated edb relations $\mathbf{R}$ and output relation 0
  - The meaning μ is given by

$$\mu(\langle \mathbf{R}, O, P \rangle) = f : \mathbf{EDB} \rightarrow \{O\},$$

where

$$f(\mathbf{I}) = P(\mathbf{I})(O)$$

Query Complexity Classes

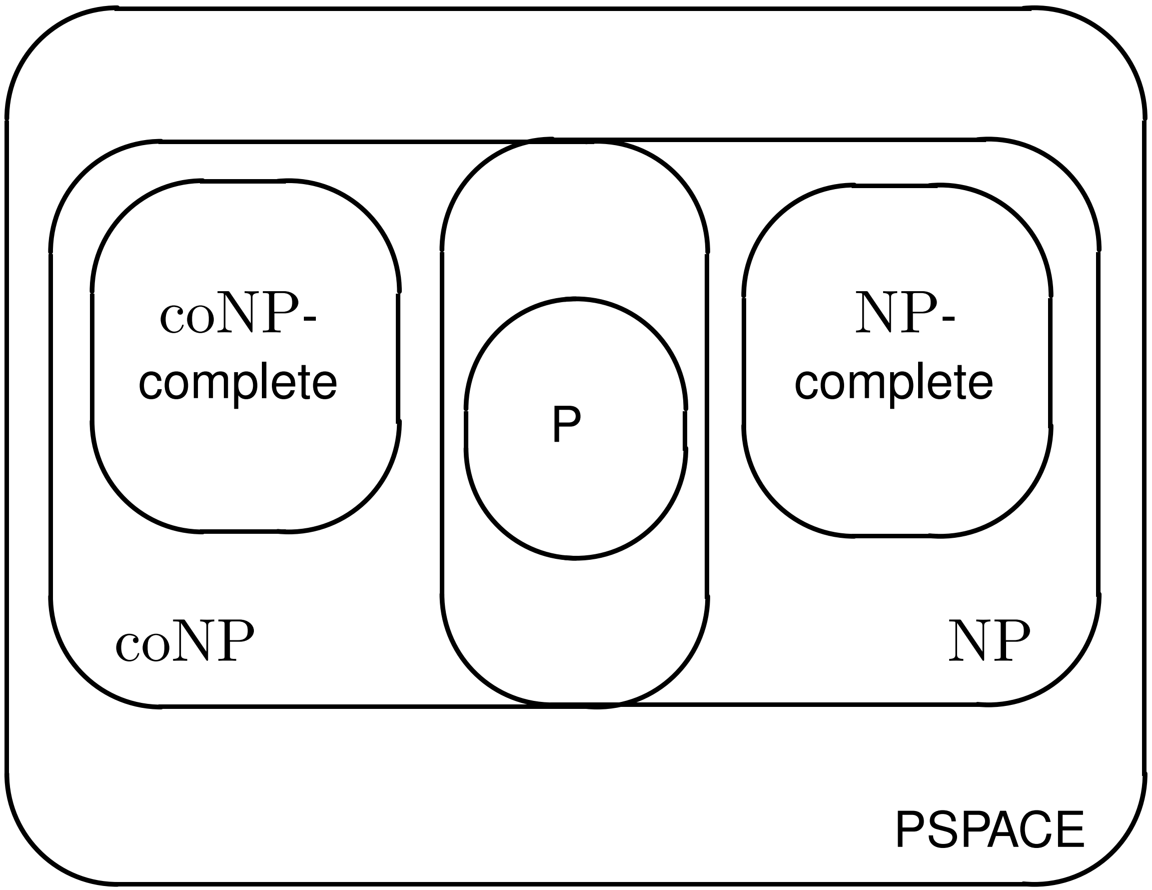
Query Output Tuple Problem (QOT): Given a database query $f : \mathbf{EDB} \to \mathbf{ODB}, \mathbf{I} \in inst(\mathbf{EDB})$, and a fact $R(\vec{c})$, decide whether $R(\vec{c}) \in f(\mathbf{I})$.

Defn. $\mathbf{QC}$ denotes the class of all generic queries f for which the QOT problem has complexity in the class C

In particular:

- $\mathbf{QP}$ = class of all generic queries f for which QOT is polynomial (f is fixed)
- $\mathbf{QcoNP}$ = class of all generic queries f for which QOT is in coNP $\Leftrightarrow$ deciding $R(\vec{c}) \notin f(\mathbf{I})$ is in NP (f is fixed)

The world of NP and coNP



(Assuming P $\neq \mathrm{NP}$ and $\mathrm{NP} \neq$ coNP)

Examples

- The transitive closure query f_{TC} is in $\mathbf{QP}$
- The Even-Query f_{Even} is in $\mathbf{QP}$
- The Graph 3-Uncolorability query f_{3uncol} is in $\mathbf{QcoNP}$

Complexity of Query Evaluation

Measures of query evaluation complexity ($\mathcal{L}$ is fixed):

Data Complexity: For a fixed query expression $E \in \mathcal{L}$, decide for a given $\mathbf{I} \in inst(\mathbf{EDB})$ and fact A whether $A \in \mu(E)(\mathbf{I})$ (i.e., QOT for fixed $f = \mu(E)$)

Expression Complexity: For fixed $\mathbf{I} \in inst(\mathbf{EDB})$, decide for given E from $\mathcal{L}$ and A whether $A \in \mu(E)(\mathbf{I})$ (i.e., QOT for fixed $\mathbf{I}$ and varying $f = \mu(E)$).

Combined Complexity: For given $E \in \mathcal{L}$, $\mathbf{I} \in inst(\mathbf{EDB})$ and A, decide whether $A \in \mu(E)(\mathbf{I})$ (i.e., QOT without further constraints)

- Typically, combined and expression complexity are similar.
- Most relevant: data complexity

Main issues

- Is data complexity of $\mathcal{L}$ polynomial / (presumably) not polynomial ?

- Is the language $\mathcal{L}$ balanced ?

  That is, if computationally “hard” queries for a complexity class C are in $\mathcal{L}$, are all queries with the complexity C in $\mathcal{L}$?

- If $\mathcal{L}$ is a “hard” query language, are there fragments of $\mathcal{L}$ in which queries are “easy” ?

  Desired: Queries from an “easy” fragment should be efficiently recognizable.

Some Important Complexity Classes

$$
\begin{aligned}
\mathrm{P} &= \textstyle\bigcup_{d>0} \mathrm{TIME}(n^d), \\
\mathrm{NP} &= \textstyle\bigcup_{d>0} \mathrm{NTIME}(n^d), \\
\mathrm{EXPTIME} &= \textstyle\bigcup_{d>0} \mathrm{TIME}(2^{n^d}), \\
\mathrm{NEXPTIME} &= \textstyle\bigcup_{d>0} \mathrm{NTIME}(2^{n^d}), \\
\mathrm{PSPACE} &= \textstyle\bigcup_{d>0} \mathrm{SPACE}(n^d), \\
\mathrm{EXPSPACE} &= \textstyle\bigcup_{d>0} \mathrm{SPACE}(2^{n^d}), \\
(= \mathrm{LOG})\ \mathrm{L} &= \mathrm{SPACE}(\log n), \\
(= \mathrm{NLOG})\ \mathrm{NL} &= \mathrm{NSPACE}(\log n).
\end{aligned}
$$

where

$$
\begin{aligned}
\mathrm{(N)TIME}(f(n)) &= \{L \mid L \text{ is decided by some (non-)DTM in time } O(f(n))\}, \\
\mathrm{(N)SPACE}(f(n)) &= \{L \mid L \text{ is decided by some DTM within space } O(f(n))\},
\end{aligned}
$$

Properties & Relationships

- Each deterministic class is closed under complementation.
- Each deterministic class is included in its nondeterministic counterpart.
- $\mathrm{P} \subseteq_{=?} \mathrm{NP} \subseteq_{=?} \mathrm{PSPACE}$
- $\mathrm{PSPACE} = \mathrm{NPSPACE}$
- $\mathrm{LOG} \subseteq_{=?} \mathrm{NL} \subseteq_{=?} \mathrm{P} \subseteq_{=?} \mathrm{PSPACE}$
- $\mathrm{NL} \subset \mathrm{PSPACE} \subseteq_{=?} \mathrm{EXPTIME}$
- $\mathrm{P} \subset \mathrm{EXPTIME}$
- $\mathrm{NP} \subset \mathrm{NEXPTIME}$

Completeness

- Recall: A decision problems Π is complete for complexity class C if (1)Π belongs to C, and (2) each problem Π' is reducible to Π.
- Usual notion of reducibility: polynomial-time transformation, inside P logspace transformations.

Defn. A query language $\mathcal{L}$ has data (resp., expression) complexity in class C, if if every query $\mu(E)$, $E \in Exp$, has data (resp., expression) complexity in C.

Defn. A query language $\mathcal{L} = \langle Exp, \mu \rangle$ is data- (resp. expression-) complete with respect to complexity class C, if

1. $\mathcal{L}$ has data (resp., expression) complexity in C, and
2. QOT for $\{\mu(E) \mid E \in Exp\}$ is complete for C under data (resp., expression) complexity.

Complexity of Generic Queries

- Often, one considers also only generic queries.
- For generic queries, the following notion is used (Abiteboul et al., 1995)

Defn. A query language $\mathcal{L}$ is in $\mathbf{QC}$, if

1. each query generic query $f \in \{\mu(E) \mid E \in Exp\}$ is in $\mathbf{QC}$, and
2. is complete w.r.t. $\mathbf{QC}$, if in addition for some such f QOT is complete for C.

Query Complexity of Relational Calculus

Theorem. Relational Calculus under active domain semantics

1. has data complexity in L.
2. is expression-complete w.r.t. PSPACE.
3. is for generic queries in **QL**.

Intuition:

- Evaluating a variable-free formula recursively is easy (scan input tape for atoms $R(\vec{c})$)
- We can evaluate quantifiers $\exists x$, $\forall x$ by looping through all values for x in the input.
- A pointer to positions of the input tape is sufficient to catch all values for x
- For fixed query, we have fixed recursion depth, and a fixed number of pointers, each of which requires $O(\log |\mathbf{I}|)$ space. $|\mathbf{I}| \ldots$ size of input $\mathbf{I} \in inst(\mathbf{EDB})$

PSPACE-Hardness

- The hardness part of expression-completeness of Relational Calculus for PSPACE can be shown by a reduction from Quantified Boolean Formulas:

  Given a formula

  $$Q_1 X_1 Q_2 X_2 \cdots Q_n X_n E$$

  where $Q_i \in \{\exists, \forall\}$ and E is a Boolean formula on variables $X_1, \ldots, X_n$, decide whether the formula evaluates to true (where variables range over $\{$0,1$\}$).

- Relational Calculus is *not* data-complete for **QL** (under non-trivial notion of reduction).

- Relational Calculus queries have data-complexity in AC_0, which means that they are evaluable by polynomial-size Boolean circuits of constant depth with $\vee$, $\wedge$, and $\neg$ gates of unbounded fan-in.

- Under parallel computation, Rel. Calculus queries are evaluable in constant time.

Fixpoint Queries and Partial Fixpoint Queries

Theorem. The Fixpoint Queries are

- data-complete w.r.t. P
- expression-complete w.r.t. $\mathrm{EXPTIME}$

Theorem. The Partial Fixpoint Queries are

- data-complete w.r.t. PSPACE
- expression-complete w.r.t. $\mathrm{EXPSPACE}$

Similar results for While^{+} and While queries.

Intuition:

Evaluation of query Q

Consider n-ary relation R, m constants: m^n tuples for R

- In computation of $\mu^+_R(\phi(R))$, R can take on at most $adom(Q, \mathbf{I})^n + 1$ different values, hence at most $adom(Q, \mathbf{I})^n + 1$ iterations.
- In computation of $\mu_R(\phi(R))$, R can take on $\leq 2^{adom(Q,\mathbf{I})^n}$ different values $\Rightarrow$ need to consider $\leq 2^{adom(Q,\mathbf{I})^n}$ iterations (otherwise Q is undefined)
- Data-complexity (n fixed): R occupies polynomial space
  - For Fixpoint Queries, $adom(Q, \mathbf{I})^n + 1$ is polynomial
  - For Partial Fixpoint Queries, counter for $2^{adom(Q,\mathbf{I})^n}$ uses $\log(2^{adom(Q,\mathbf{I})^n}) = adom(Q, \mathbf{I})^n$, i.e., polynomial space
- Expression-complexity: Exponential blow-up, because of dynamic arities / number of variables ($adom(Q, \mathbf{I})^n + 1$ is exponential).

Query Complexity of Datalog

Theorem. (Plain) datalog is

1. data-complete w.r.t. P.
2. expression-complete w.r.t. $\mathrm{EXPTIME}$.

Proof:

- Membership part: $\mathbf{T}_P^\omega(\mathbf{I})$ is reached in a polynomial resp. exponential number of steps.
- Hardness part: Show this e.g. by a generic encoding of Turing machines to datalog.

Turing Machines

- Informally: a Turing machine (TM) is a device able to read from and write on a semi-infinite tape, whose contents may be locally accessed and changed in a computation.
- Formally: A Turing machine is a quadruple

$$(S, \Sigma, \delta, s_0),$$

  where

  - S is a finite set of states,
  - Σ is a finite alphabet of symbols, containing a special symbol $\sqcup$ called the blank.
  - δ is a transition function, and
  - $s_0 \in S$ is the initial state.

Turing Machines /2: Transition Function

- The transition function δ is a map

$$\delta: \; S \times \Sigma \;\rightarrow\; (S \cup \{\texttt{halt}, \texttt{yes}, \texttt{no}\}) \times \Sigma \times \{\texttt{-1, 0, +1}\},$$

  where

  - $\texttt{halt}$, $\texttt{yes}$, and $\texttt{no}$ denote three additional states not occurring in S, and
  - $\texttt{-1, 0, +1}$ denote motion directions.

- Assumption: The machine is well-behaved and never moves off the tape, i.e., $d \neq -1$ whenever the cursor is on the leftmost cell; this can be ensured by proper design of δ.

Turing Machines /3: Tape & Input

- The tape of T is divided into cells containing symbols of Σ.
- There is a cursor that may move along the tape.
- At the start, T is in the initial state s_0, and the cursor points to the leftmost cell of the tape.
- An input string I is written on the tape as follows: the first $|I|$ cells $c_0, \ldots, c_{|I|-1}$ of the tape, where $|I|$ denotes the length of I, contains the symbols of I, and all other cells contain $\sqcup$.

Example: String "$ABCA$": $c_0 = A$,, $c_1 = B$, $c_2 = C$, $c_3 = C$,
$c_4 = A$, $c_5 = \sqcup$, $c_6 = \sqcup$, $\ldots$

Turing Machines /4: Computation

- Successive steps of computation are made according to δ. Assume that
  - T is in a state $s \in S$,
  - the cursor points to the symbol $\sigma \in \Sigma$ on the tape.

  Let

$$\delta(s, \sigma) = (s', \sigma', d).$$

  Then
  - T changes its current state to s',
  - overwrites σ' on σ, and
  - moves the cursor to the $\begin{cases} \text{previous cell,} & \text{if } d = -1, \\ \text{next cell,} & \text{if } d = +1, \\ \text{samce cell,} & \text{if } d = 0. \end{cases}$

Turing Machines /5: Halting

- When any of the states `halt`, `yes` or `no` is reached, T halts.
- T accepts the input I, if T halts in `yes`.
- T rejects the input I if T halts in `no`.
- If `halt` is reached, the output of T on I is computed.
- This output, denoted by $T(I)$, is defined as the string contained in the initial segment of the tape which ends before the first blank.

Simulating a TM by a logic program

- **Goal**: Given a TM T, describe a datalog program $P(T, I, N)$ which simulates the computation of T on some input I *for at most* N *steps*

- First Step (conceptually easier): Write a *propositional (variable-free)* program $P(T, I, N)$ for such simulation (no edb needed).

- Use a special atom *accept* such that $P(T, I, N)$ computes *accept* true iff T accepts I in at most N steps.

- Modify the program $P(T, I, N)$ to obtain a datalog program with relations and edb.

Expressing the Transition Function by Rules

- The transition function δ can be represented by a table
- The rows are tuples $t = \langle s, \sigma, s', \sigma', d \rangle$, expressing an if-then-rule:

  if at some time instant τ T is in state s, the cursor points to cell number π, and this cell contains symbol σ
  then at instant $\tau + 1$ the T is in state s', cell number π contains symbol σ', and the cursor points to cell number $\pi + d$.

- Using this table, we describe the complete evolution of T on input string I from its initial configuration at time instant 0 to the configuration at instant N by a propositional logic program $P(T, I, N)$.

Groups of Propositional Atoms

symbol$_\sigma[\tau, \pi]$ for $0 \leq \tau \leq N$, $0 \leq \pi \leq N$ and $\sigma \in \Sigma$. Intuitive meaning: at instant τ of the computation, cell number π contains symbol σ.

cursor$[\tau, \pi]$ for $0 \leq \tau \leq N$ and $0 \leq \pi \leq N$. Intuitive meaning: at instant τ the cursor points to cell number π.

state$_s[\tau]$ for $0 \leq \tau \leq N$ and $s \in S$. Intuitive meaning: at instant τ, T is in state s.

accept Intuitive meaning: T has reached state $\texttt{yes}$.

Initial Configuration

- Denote by I_k the k-th symbol of the string $I = I_0 \cdots I_{|I|-1}$.
- The initial configuration of T on input I is reflected by the following initialization facts in $P(T, I, N)$:

$$
\begin{array}{rll}
\textit{symbol}_\sigma[0,\pi] & \leftarrow & \text{for } 0 \leq \pi < |I| \text{, where } I_\pi = \sigma \\
\textit{symbol}_\sqcup[0,\pi] & \leftarrow & \text{for } |I| \leq \pi \leq N \\
\textit{cursor}[0,0] & \leftarrow & \\
\textit{state}_{s_0}[0] & \leftarrow &
\end{array}
$$

Transition Rules

- Each entry $\langle s, \sigma, s', \sigma', d \rangle$ of δ is translated into the following transition rules ($0 \leq \tau < N$, $0 \leq \pi < N$, and $0 \leq \pi + d$):

$$
\begin{aligned}
\textit{symbol}_{\sigma'}[\tau+1,\pi] &\leftarrow \textit{state}_s[\tau], \textit{symbol}_\sigma[\tau,\pi], \textit{cursor}[\tau,\pi] \\
\textit{cursor}[\tau+1,\pi+d] &\leftarrow \textit{state}_s[\tau], \textit{symbol}_\sigma[\tau,\pi], \textit{cursor}[\tau,\pi] \\
\textit{state}_{s'}[\tau+1] &\leftarrow \textit{state}_s[\tau], \textit{symbol}_\sigma[\tau,\pi], \textit{cursor}[\tau,\pi]
\end{aligned}
$$

- Further *inertia rules* carry over values of tape cells which are not changed during the transition ($0 \leq \tau < N, 0 \leq \pi < \pi' \leq N$):

$$
\begin{aligned}
\textit{symbol}_\sigma[\tau+1,\pi] &\leftarrow \textit{symbol}_\sigma[\tau,\pi], \textit{cursor}[\tau,\pi'] \\
\textit{symbol}_\sigma[\tau+1,\pi'] &\leftarrow \textit{symbol}_\sigma[\tau,\pi'], \textit{cursor}[\tau,\pi]
\end{aligned}
$$

Accept Rules

- The accept rules derive the atom *accept*, whenever an accepting configuration is reached:

$$\textit{accept} \quad \leftarrow \quad \textit{state}_{\mathtt{yes}}[\tau] \qquad \text{for } 0 \leq \tau \leq N.$$

Simulation Result

Proposition. The least model of $P(T, I, N)$ contains *accept* if and only if T accepts the input string I within N steps.

Observations:

- $\mathbf{T}_P^0 = \emptyset$
- $\mathbf{T}_P^1$ contains the initial configuration of T at time instant 0.
- By construction, the least fixpoint $\mathbf{T}_P^\omega$ of P is reached at T_P^{N+2}
- the ground atoms added to $\mathbf{T}_P^\tau$, $2 \leq \tau \leq N + 1$, describe the configuration of T on the input I at the time instant $\tau - 1$.

Modification to Datalog Program with Variables

The above propositional program can be lifted to programs with variables to simulate computation of T on designated inputs I.

Main ideas:

- Use relations *symbol*$_\sigma(\vec{x}, \vec{y})$, *cursor*$(\vec{x}, \vec{y})$ and *state*$_s(\vec{x})$ instead of the propositional atoms *symbol*$_\sigma[X, Y]$, *cursor*$[X, Y]$ and *state*$_s[X]$ respectively.
- The time points τ and tape positions π from 0 to $N-1$ are represented by tuples $t_\tau = \langle c_1, ..., c_l\rangle$ of the same arity.
- Use a *successor relation* on tuples $\langle c_1, ..., c_l\rangle$ to encode $\pi + 1$ and $\tau + 1$.

Modification for Data Complexity

$N = n^k$, k constant ($n = |I|$), is polynomial.

- Use k-ary tuples and an active domain U of size n, stored in the edb.
- The functions $\tau+1$ and $\pi+d$ are realized by means of the successor *Succ*^{k} and first (last) element *First*^{k} (*Last*^{k}) w.r.t. a linear order $\leq^k$ on U^k (built from relations in edb, or fully built-in).
- Store the string $I = enc(\mathbf{I})$ encoding a (selected) input database $\mathbf{I}$ of the query, and padding ␣'s on the initial tape of T in the edb using relations *input*$_\sigma(\cdot)$ of arity k.

  Informally, *input*$_\sigma(\pi)$ means that cell c_π contains initially symbol σ.
- Copy *input*$_\sigma(\pi)$ to *symbol*$_\sigma(0, \pi)$

Modification for Expression Complexity

$N = 2^m$, where $m = n^k$.

- Use m-ary tuples over a fixed domain $U = \{0, 1\}$, and an empty database.
- The functions $\tau{+}1$ and $\pi{+}d$ are realized by means of the successor *Succ*^{m} w.r.t. a linear order $\leq^m$ on U^m, built entirely in P.

Defining *Succ*^{m} and $\leq^m$ over U

Inductive definition of *Succ*^{m} on U

- Suppose *Succ*$^i(\vec{x}, \vec{y})$, *First*$^i(\vec{x})$, and *Last*$^i(\vec{x})$ tell the successor, the first, and the last element from a linear order $\leq^i$ on U^i, where $\vec{x}$ and $\vec{y}$ have arity i.
- Use rules

$$\begin{array}{rcl} \mathit{Succ}^{i+1}(z, \vec{x}, z, \vec{y}) & \leftarrow & \mathit{Succ}^{i}(\vec{x}, \vec{y}) \\ \mathit{Succ}^{i+1}(z, \vec{x}, z', \vec{y}) & \leftarrow & \mathit{Succ}^{1}(z, z'), \mathit{Last}^{i}(\vec{x}), \mathit{First}^{i}(\vec{y}) \\ \mathit{First}^{i+1}(z, \vec{x}) & \leftarrow & \mathit{First}^{1}(z), \mathit{First}^{i}(\vec{x}) \\ \mathit{Last}^{i+1}(z, \vec{x}) & \leftarrow & \mathit{Last}^{1}(z), \mathit{Last}^{i}(\vec{x}) \end{array}$$

- For $i = 1$ *Succ*$^1(x, y)$, *First*$^1(x)$, and *Last*$^1(x)$ on $U^1 = U$ must be provided.

- The order $\leq^m$ is then easily defined by rules

$$\begin{array}{rcl} \leq^m(\vec{x},\vec{x}) & \leftarrow & dom(x_1),\ldots,dom(x_m) \\ \leq^m(\vec{x},\vec{y}) & \leftarrow & \textit{Succ}^m(\vec{x},\vec{z}),\ \leq^m(\vec{z},\vec{y}) \\ dom(x) & \leftarrow & \textit{First}^1(x) \\ dom(y) & \leftarrow & \textit{Succ}^1(x,y) \end{array}$$

where $\vec{x} = x_1,\ldots,x_m$, using dom for the active domain.

Modification for expression-complexity Hardness

$N = 2^m$, where $m = n^k$, and $U = \{0, 1\}$, use ordering $0 \leq^1 1$

Modify the program $P(T, I, N)$ as follows:

- Provide facts *Succ*$^1(0, 1)$, *First*$^1(0)$, and *Last*$^1(1)$ in P.
- Initialization facts:
  - Translate *symbol*$_\sigma[0, \pi]$ into rules

$$\textit{symbol}_\sigma(\vec{x}, \vec{t}) \leftarrow \textit{First}^m(\vec{x}),$$

    where $\vec{t}$ represents the position π,
  - translate similarly the facts *cursor*$[0, 0]$ and *state*$_{s_0}[0]$.
  - Translate *symbol*$_\textvisiblespace[0, \pi]$, where $|I| \leq \pi \leq N$, to the rule

$$\textit{symbol}_\textvisiblespace(\vec{x}, \vec{y}) \quad \leftarrow \quad \textit{First}^m(\vec{x}),\ \leq^m(\vec{t}, \vec{y})$$

    where $\vec{t}$ represents the number $|I|$.

- transition and inertia rules: For realizing $\tau + 1$ and $\pi + d$, use in the body atoms $\mathit{Succ}^m(\vec{x}, \vec{x}')$.

  Example:

  $$\mathit{symbol}_{\sigma'}[\tau + 1, \pi] \quad \leftarrow \quad \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau, \pi], \mathit{cursor}[\tau, \pi]$$

  is translated into

  $$\mathit{symbol}_{\sigma'}(\vec{x}', \vec{y}) \quad \leftarrow \quad \mathit{state}_s(\vec{x}), \mathit{symbol}_\sigma(\vec{x}, \vec{y}), \mathit{cursor}(\vec{x}, \vec{y}), \mathit{Succ}^m(\vec{x}, \vec{x}').$$

- accept rules: translation is straightforward.

Concluding $\mathrm{EXPTIME}$ Hardness

Let $P'(T, I, N)$ denote the datalog program with empty edb described for T, I, and $N = 2^m$, $m = n^k$ (where $n = |I|$)

- $P'(T, I, N)$ is constructible from T and I in polynomial time (in fact, in logarithmic space).
- $P'(T, I, N)$ has *accept* in its least model $\Leftrightarrow$ T accepts input I in at most N steps.

Consequence:

Theorem. Datalog has $\mathrm{EXPTIME}$-hard expression complexity.

Notice:

- The program $P'(T, I, N)$ uses constants (0,1)
- They can be easily eliminated from the program (move e.g. *Succ*^{1}, *First*^{1}, and *Last*^{1} to edb).

Bibliography

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001. Available at `http://www.kr.tuwien.ac.at/staff/eiter/et-archive/`.

[3] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems – The Complete Book*. Prentice Hall, 2002.