5. Complexity of Query Evaluation

Beyond Traktenbrot’s Theorem

- By Traktenbrot’s Theorem, it is undecidable to check whether a given first-order query $Q$ produces some output over some database.
- What happens if $D$ is actually given as input?
The following are natural (decision) problems in this context:

**QUERY-OUTPUT-TUPLE (QOT)**

**INSTANCE:** A database $D$, a query $Q$, a tuple $\vec{c}$ of values.
**QUESTION:** Does $\vec{c} \in Q(D)$ hold?

**BOOLEAN-QUERY-EVALUATION (BQE)**

**INSTANCE:** A database $D$, a Boolean query $Q$.
**QUESTION:** Does $Q$ evaluate to $\text{true}$ in $D$?

**NOTE:** we often view Boolean domain calculus queries $\{\langle \rangle | \phi \}$ simply as closed formulae $\phi$.

**QOT vs. BQE vs. QNE**

We concentrate here mainly on the complexity of BQE.

Not a limitation: in our setting QOT and QNE are essentially the same problems as BQE:

**From QOT to BQE**

Assume a database $D$, a domain calculus query $Q = \{\vec{x} | \phi(\vec{x})\}$, and a value tuple $\vec{c} = (c_1, \ldots, c_n)$. Then $\vec{c} \in Q(D)$ iff $Q'$ evaluates to $\text{true}$ in $D$, where

- $\vec{x} = (x_1, \ldots, x_n)$, and
- $Q' = \exists \vec{x}. (\phi(\vec{x}) \land x_1 = c_1 \ldots \land x_n = c_n)$

**From BQE to QNE and QOT $\Rightarrow$ trivial.**

We concentrate here mainly on the complexity of BQE.

Not a limitation: in our setting QOT and QNE are essentially the same problems as BQE:

**From QOT to BQE**

Assume a database $D$, a domain calculus query $Q = \{\vec{x} | \phi(\vec{x})\}$, and a value tuple $\vec{c} = (c_1, \ldots, c_n)$. Then $\vec{c} \in Q(D)$ iff $Q'$ evaluates to $\text{true}$ in $D$, where

- $\vec{x} = (x_1, \ldots, x_n)$, and
- $Q' = \exists \vec{x}. (\phi(\vec{x}) \land x_1 = c_1 \ldots \land x_n = c_n)$

**From QNE to BQE**

Assume a database $D$, a domain calculus query $Q = \{\vec{x} | \phi(\vec{x})\}$. Then $Q(D) \neq \emptyset$ iff $\exists \vec{x}. \phi(\vec{x})$ evaluates to $\text{true}$ in $D$.

From BQE to QNE and QOT $\Rightarrow$ trivial.
Complexity Measures for BQE

Combined complexity

The complexity of BQE without any assumptions about the input query Q and database D is called the combined complexity of BQE.

Further measures are obtained by restricting the input:

Data and query complexity

Data complexity of BQE refers to the following decision problem:
Let Q be some fixed Boolean query.
INSTANCE: An input database D.
QUESTION: Does Q evaluate to true in D?

Query complexity of BQE refers to the following decision problem:
Let D be some fixed input database.
INSTANCE: A Boolean query Q.
QUESTION: Does Q evaluate to true in D?

Relevant Complexity Classes

We recall the inclusions between some fundamental complexity classes:

\[ L \subseteq P \subseteq NP \subseteq PSPACE \subseteq \text{EXPTIME} \]

- L is the class of all problems solvable in logarithmic space,
- P \( \subseteq \) in polynomial time,
- NP \( \subseteq \) in nondeterministic polynomial time,
- PSPACE \( \subseteq \) in polynomial space,
- EXPTIME \( \subseteq \) in exponential time.

Complexity of First-order Queries

Theorem (A)

The query complexity and the combined complexity of domain calculus queries is PSPACE-complete (even if we disallow negation and equality atoms). The data complexity is in L (actually, even in a much lower class).
Complexity of First-order Queries

**Theorem (A)**

The query complexity and the combined complexity of domain calculus queries is PSPACE-complete (even if we disallow negation and equality atoms). The data complexity is in L (actually, even in a much lower class).

To prove the theorem, we proceed in steps as follows:

1. We provide an algorithm for query evaluation:
   - it shows PSPACE membership for combined complexity (and thus for query complexity as well), and
   - L membership w.r.t. data complexity,
2. We show PSPACE-hardness of query complexity (clearly, the lower bound applies to combined complexity as well).

---

**An algorithm for evaluating FO queries**

- We consider an arbitrary FO formula \( \psi \) and a database \( D \).
- W.l.o.g., the formula is of the form
  \[
  \psi = \exists x_1 \forall y_1 \ldots \exists x_n \forall y_n \varphi(x_1, y_1, \ldots, x_n, y_n).
  \]
- Let the active domain \( \text{dom} \) of \( D \) be \( \text{dom} = \{a_1, \ldots, a_m\} \).
- For the evaluation of the formula, we design two procedures \( \text{evaluate}_\exists \) and \( \text{evaluate}_\forall \), which call each other recursively.
- The algorithm uses global variables \( n \) and \( X = \{x_1, y_1, \ldots, x_n, y_n\} \).

```plaintext
GLOBAL x_1, y_1, \ldots, x_n, y_n

Boolean evaluate_\exists (Integer i)
    for x_i from a_1 to a_m do
        if evaluate_\forall (i) = true then return true
    endfor
    return false.

Boolean evaluate_\forall (Integer i)
    for y_i from a_1 to a_m do
        if i = n then
            if \( \varphi \) evaluates to false under the current values of \( x_1, y_1, \ldots, x_n, y_n \) then return false
        else
            if evaluate_\exists (i + 1) = false then return false
        endif
    endfor
    return true.
```

By construction: \( \psi \) is true in \( D \) iff \( \text{evaluate}_\exists (1) = \text{true} \).
Let us analyse the space usage of our algorithm. We have to store:

1. The input database $D$ and the formula $\psi$:
   - do not contribute to the space requirements.
2. The global variables $X = \{x_1, y_1, \ldots, x_n, y_n\}$.
   - Each variable requires $O(\log m)$ bits of space. Thus $X$ needs $O(n \log m)$ bits. Note that $X$ requires logarithmic space if $\psi$ is fixed.
3. A call stack $S = (S_1, \ldots, S_k)$, where $k \leq 2n$ and each $S_j$ stores a state in which a subroutine is called. Clearly, for both subroutines a state $S_j$ only needs to store the value of $i$ and the return position in the subroutine.
   - Storing a value $i \in \{1, \ldots, 2n\}$ requires logarithmic space in the size of $\psi$ (i.e. $O(\log n)$), but only constant space if $\psi$ is fixed. (The return position requires constant space in both cases.)
   - Hence $S$ needs $O(n \log n)$ bits of storage, which is constant if $\psi$ is fixed.
4. Space for evaluating $\varphi$ in an assignment
   - requires a transversal of the parse tree of $\psi$: space $O(\log ||\psi||)$ suffices.

Overall we need $O(n \log m + n \log n + \log ||\psi||)$ bits of storage.

$O(n \log m + n \log n + \log ||\psi||)$ means that we only need polynomial space in the combined size of $D$ and $\psi$.

**Proposition**

$\text{BQE} \in \text{PSPACE}$ w.r.t. combined complexity.
This also implies $\text{BQE} \in \text{PSPACE}$ w.r.t. query complexity.

For fixed $\psi$, the space required is $O(\log m)$, i.e. logarithmic in the data.

**Proposition**

$\text{BQE} \in \text{L}$ w.r.t. data complexity.

**Remark.** A detailed proof is given in the Komplexitätstheorie lecture.

The PSPACE lower bound

To prove the PSPACE-hardness result, we first recall quantified Boolean formulae:

**QSAT (QBF)**

**INSTANCE:** An expression $\exists x_1 \forall x_2 \exists x_3 \cdots Q x_n \phi$, where $Q$ is either $\forall$ or $\exists$ and $\phi$ is a Boolean formula in CNF with variables from $\{x_1, x_2, x_3, \ldots, x_n\}$.

**QUESTION:** Is there a truth value for the variable $x_1$ such that for both truth values of $x_2$ there is a truth value for $x_3$ and so on up to $x_n$, such that $\phi$ is satisfied by the overall truth assignment?

**Theorem**

**QSAT** is PSPACE-complete.
Proof of the PSPACE-Hardness of BQE

The PSPACE-hardness result for Theorem (A) can be shown by a reduction from the QSAT-problem. Let $\psi$ be an arbitrary QBF with
$$\psi = \exists x_1 \forall x_2 \ldots Q x_n \alpha(x_1, \ldots, x_n)$$
where $Q$ is either $\forall$ or $\exists$ and $\alpha$ is a quantifier-free Boolean formula with variables in $\{x_1, \ldots, x_n\}$.

We first define the (fixed) input database $D$ over the predicate symbols $\mathcal{L} = \{\text{istrue, isequal, not, or, and}\}$ with the obvious meaning:
$$D = \{\text{istrue}(1), \text{isequal}(0, 0), \text{isequal}(1, 1), \text{not}(1, 0), \text{not}(0, 1),$$
$$\text{or}(1, 1), \text{or}(1, 0, 1), \text{or}(0, 1, 1), \text{or}(0, 0, 0),$$
$$\text{and}(1, 1), \text{and}(1, 0), \text{and}(0, 1), \text{and}(0, 0)\}$$

Proof of the PSPACE-Hardness (continued)

The first-order query $\phi$ is then defined as follows:
$$\phi \equiv \exists x \exists z_1 \forall z_2 \ldots Q z_n \alpha(z, x) \land \text{istrue}(x)$$
where $Q$ is either $\forall$ or $\exists$ (as in the formula $\psi$).

We claim that this problem reduction is correct, i.e.: The QBF $\psi = \exists x_1 \forall x_2 \ldots Q x_n \alpha(x_1, \ldots, x_n)$ is $\text{true} \iff$ the first-order query $\phi \equiv \exists x \exists z_1 \forall z_2 \ldots Q z_n \alpha(z, x) \land \text{istrue}(x)$ evaluates to $\text{true}$ over the database $D$.

The proof is straightforward. It suffices to show by induction on the structure of $\alpha$ that the formulae $T_\beta(z_1, \ldots, z_n, x)$ indeed have the intended meaning.

Complexity of Conjunctive Queries

Recall that conjunctive queries (CQs) are a special case of first-order queries whose only connective is $\land$ and whose only quantifier is $\exists$ (i.e., $\forall$, $\neg$ and $\forall$ are excluded).

E.g.: $Q = \{\langle x \rangle \mid \exists y, z. R(x, y) \land R(y, z) \land P(z, x)\}$

Proof of the PSPACE-Hardness (continued)

For each sub-formula $\beta$ of $\alpha$, we define a quantifier-free, first-order formula $T_\beta(z_1, \ldots, z_n, x)$ with the following intended meaning:

The QBF $\psi$ is then defined as follows:

Case $\beta = z_i$ (with $1 \leq i \leq n$): $T_\beta(z, x) \equiv \text{isequal}(z_i, x)$

Case $\neg \beta'$:
$$T_\beta(z, x) \equiv \exists t_1 T_{\beta'}(z, t_1) \land \text{not}(t_1, x)$$

Case $\beta_1 \land \beta_2$:
$$T_\beta(z, x) \equiv \exists t_1, t_2 \ T_{\beta_1}(z, t_1) \land T_{\beta_2}(z, t_2) \land \text{and}(t_1, t_2, x)$$

Case $\beta_1 \lor \beta_2$:
$$T_\beta(z, x) \equiv \exists t_1, t_2 \ T_{\beta_1}(z, t_1) \land T_{\beta_2}(z, t_2) \lor \text{or}(t_1, t_2, x)$$
Complexity of Conjunctive Queries

Recall that conjunctive queries (CQs) are a special case of first-order queries whose only connective is $\land$ and whose only quantifier is $\exists$ (i.e., $\lor$, $\forall$ and $\forall$ are excluded).

E.g.: $Q = \{ \{ x \} \mid \exists y, z. R(x, y) \land R(y, z) \land P(z, x) \}$

**Theorem (B)**

The query complexity and the combined complexity of BQE for conjunctive queries is NP-complete.

**Proof**

NP-Membership (of the combined complexity). For each variable $u$ of the query, we guess a domain element to which $u$ is instantiated. Then we check whether all the resulting ground atoms in the query body exist in $D$. This check is obviously feasible in polynomial time.

Now let an arbitrary instance of the 3-SAT problem be given through the 3-CNF formula $\Phi = \bigwedge_{i=1}^{n} l_1 \lor l_2 \lor l_3$ over the propositional variables $x_1, \ldots, x_k$. Then we define a conjunctive query $Q$ as follows:

$$(\exists x_1, \ldots, x_k) c(l_1^1, l_2^1, l_3^1) \land \ldots \land c(l_1^k, l_2^k, l_3^k) \land \forall(x_1, \bar{x_1}) \land \cdots \land \forall(x_k, \bar{x_k})$$

where $l_i^l = x$ if $l = x$, and $l_i^l = \bar{x}$ if $l = \neg x$. Moreover, $x_1, \ldots, x_k$ are fresh first-order variables. By slight abuse of notation, we thus use $x_i$ to denote either a propositional atom (in $\Phi$) or a first-order variable (in $Q$).

It is straightforward to verify that the 3-CNF formula $\Phi$ is satisfiable $\iff$ $Q$ evaluates to true in $D$.

Complexity of Datalog

**Theorem (C)**

Query evaluation (i.e., the QOT problem) of Datalog has the following complexity:

- P-complete w.r.t. data complexity, and
- EXPTIME-complete w.r.t combined and query complexity.

To prove the theorem, we first concentrate on ground Datalog programs:

- A program is ground if it has no variables.
- Such programs are also known as propositional logic programs.
- Note that a ground atom $R(tim, bob)$ can be seen as a propositional variable $R_{tim,bob}$.

Ground Datalog

**Theorem**

Query evaluation for ground Datalog programs is P-complete w.r.t. combined complexity.

**Proof (continued)**

Hardness (of the query complexity). We reduce the NP-complete 3-SAT problem to our problem. For this purpose, we consider the following input database (over a ternary relation symbol $c$ and a binary relation symbol $v$) as fixed:

$$D = \{ c(1, 1, 1), c(1, 1, 0), c(1, 0, 1), c(1, 0, 0), c(0, 1, 1), c(0, 0, 1), v(1, 0), v(0, 1) \}$$

Now let an arbitrary instance of the 3-SAT problem be given through the 3-CNF formula $\Phi = \bigwedge_{i=1}^{n} l_1 \lor l_2 \lor l_3$ over the propositional variables $x_1, \ldots, x_k$.

Recall that the semantics of a given program $P$ on database $DB$ can be computed in polynomial time.

The number of iterations (i.e. applications of $T$) is bounded by the number of rules plus 1. Each iteration step is clearly feasible in polynomial time.

This least fixpoint $P\omega(DB)$ can be defined as

$$P\omega(DB) = \bigcup_{i \in \mathbb{N}} T^i(P(DB))$$

where $P(DB)$ is the least fixed-point of the immediate consequence operator $P$.

Recall that the semantics of a given program $P$ on database $DB$ can be computed in polynomial time.
Ground Datalog

**Theorem**

Query evaluation for ground Datalog programs is P-complete w.r.t. combined complexity.

**Proof: (Membership)**

- Recall that the semantics of a given program $P$ can be defined as the least fixed-point of the immediate consequence operator $T_P$.
- This least fixpoint $T_P^\omega(DB)$ can be computed in polynomial time even if the “naive” evaluation algorithm is applied.
- The number of iterations (i.e. applications of $T_P$) is bounded by the number of rules plus 1.
- Each iteration step is clearly feasible in polynomial time.

---

P-hardness of Ground Datalog: the Atoms

The propositional atoms in $P(M, N)$.

- **symbol$_\alpha[^{\tau}, \pi]$** for $0 \leq \tau \leq N$, $0 \leq \pi \leq N$ and $\alpha \in \Sigma$. Intuitive meaning: at instant $\tau$ of the computation, cell number $\pi$ contains symbol $\alpha$.
- **cursor[^{\tau}, \pi]$** for $0 \leq \tau \leq N$ and $0 \leq \pi \leq N$. Intuitive meaning: at instant $\tau$, the cursor points to cell number $\pi$.
- **state$_q[^{\tau}]$** for $0 \leq \tau \leq N$ and $q \in K$. Intuitive meaning: at instant $\tau$, the machine $M$ is in state $q$.
- **accept** Intuitive meaning: $M$ has reached state $q_{yes}$.

---

P-hardness of Ground Datalog: the Database

The construction of the database $DB(I, N)$:

- symbol$_0[0, 0]$.
- symbol$_\sigma[0, \pi]$, for $0 < \pi \leq |I|$, where $I_\pi = \sigma$
- symbol$_{|I|, \pi}$, for $|I| \leq \pi \leq N$
- cursor[0, 0].
- state$\delta_{start}$[0].
P-hardness of Ground Datalog: the Rules

- transition rules: for each entry \((q_1, \sigma_1, q_2, \sigma_2, d)\), \(0 \leq \tau < N\), \(0 \leq \pi < N\), and \(0 \leq \pi + d\).

\[
\text{symbol}_{q_1}[\tau + 1, \pi] \leftarrow \text{state}_{q_1}[\tau], \text{symbol}_{q_1}[\tau, \pi], \text{cursor}[\tau, \pi]
\]

\[
\text{cursor}[\tau + 1, \pi + d] \leftarrow \text{state}_{q_1}[\tau], \text{symbol}_{q_1}[\tau, \pi], \text{cursor}[\tau, \pi]
\]

\[
\text{state}_{q_1}[\tau + 1] \leftarrow \text{state}_{q_1}[\tau], \text{symbol}_{q_1}[\tau, \pi], \text{cursor}[\tau, \pi]
\]

- inertia rules: where \(0 \leq \tau < N\), \(0 \leq \pi < \pi' \leq N\)

\[
\text{symbol}_{q_1}[\tau + 1, \pi] \leftarrow \text{symbol}_{q_1}[\tau, \pi], \text{cursor}[\tau, \pi']
\]

\[
\text{symbol}_{q_1}[\tau + 1, \pi'] \leftarrow \text{symbol}_{q_1}[\tau, \pi'], \text{cursor}[\tau, \pi]
\]

- accept rule: for \(0 \leq \tau \leq N\)

\[
\text{accept} \leftarrow \text{state}_{q_1}[\tau]
\]

Data Complexity of Datalog

**Proposition**

*Query evaluation in Datalog is \(P\)-complete w.r.t. data complexity.*

**Proof:** (Membership)

Effective reduction to reasoning over ground Datalog programs is possible. Given a program \(P\), a database \(DB\), and atom \(A\):

- Generate \(P' = \text{ground}(P, DB)\), i.e. the set all ground instances of rules in \(P\):

\[
\text{ground}(P, DB) = \bigcup_{r \in P} \text{Ground}(r; P, DB)
\]

NB: more details on \(\text{Ground}(r; P, DB)\) in Lecture 2.

- Decide whether \(A \in T_{P'}(DB)\).
Grounding Complexity

Given a program $P$ and a database $DB$, the number of rules in $\text{ground}(P, DB)$ is bounded by

$$|P| \times \#\text{consts}(P, DB)^{v_{\text{max}}}$$

- $v_{\text{max}}$ is the maximum number of different variables in any rule $r \in P$
- $\#\text{consts}(P, DB)$ is the number of constants occurring in $P$ and $DB$.
- $\text{ground}(P, DB)$ is polynomial in the size of $DB$.
- Hence, the complexity of propositional logic programming is an upper bound for the data complexity.
- Note that $\text{ground}(P, DB)$ can be exponential in the size of $P$.

Data Complexity of Datalog: P-hardness

**Proof: Hardness**

The P-hardness can be shown by writing a simple Datalog meta-interpreter for ground programs with at most 3 atoms per rule:

- Represent rules $A_0 \leftarrow A_1, \ldots, A_i$ of such a program $P$, where $0 \leq i \leq 2$, using database facts $(A_0, \ldots, A_i)$ in an $(i+1)$-ary relation $R_i$ on the propositional atoms.
- Then, the program $P$ which is stored this way in a database $DB_{MI}(P)$ can be evaluated by a fixed Datalog program $P_{MI}$ which contains for each relation $R_i$, $0 \leq i \leq 2$, a rule

$$T(X_0) \leftarrow T(X_1), \ldots, T(X_i), R_i(X_0, \ldots, X_i).$$

- $T(x)$ intuitively means that atom $x$ is true. Then, $A \in T^P_{\text{DB}}$ iff $T(A) \in T^P_{DB_{MI}}(DB_{MI}(P))$
- P-hardness of the data complexity of Datalog is then immediately obtained.

Combined and Query Complexity of Datalog

**Proposition**

*Datalog is EXPTIME-complete w.r.t. query and combined complexity.*

**Proof**

*(Membership)* Grounding $P$ using $DB$ leads to a propositional program $\text{ground}(P, DB)$ whose size is exponential in the size of $P$ and $DB$.

Hence, the query and the combined complexity is in EXPTIME.
Combined and Query Complexity of Datalog

Proposition

**Datalog is EXPTIME-complete w.r.t. query and combined complexity.**

Proof

(Membership) Grounding 

\[ P \] using 

\[ DB \]

leads to a propositional program 

\[ ground(P, DB) \]

whose size is exponential in the size of \( P \) and \( DB \).

Hence, the query and the combined complexity is in EXPTIME.

(Hardness) We show hardness for query complexity only. Goal: adapt our previous encoding of \( TM \) and input \( I \) to obtain a program 

\[ P_{dat}(M, I, N) \]

and a fixed database \( DB_{dat} \) to decide acceptance of \( M \) on \( I \) within \( N = 2^m \) steps, where \( m = n^k(n = |I|) \) is a polynomial.

Note: We are not allowed to generate an exponentially large program by using exponentially many propositional atoms (the reduction would not be polynomial!).

Query Complexity of Datalog: EXPTIME-hardness

The predicates \( Succ^m, First^m, \) and \( Last^m \) are provided.

- The database facts \( symbol_0[0, \pi] \) are readily translated into the Datalog rules

\[
symbol_0(X, t) \leftarrow First^m(X),
\]

where \( t \) represents the position \( \pi \).

- Similarly for the facts \( cursor[0, 0] \) and \( state_0[0] \).

- Database facts \( symbol[I, 0, \pi] \), where \( |I| \leq \pi \leq N \), are translated to the rule

\[
symbol(X, Y) \leftarrow First^m(X), \leq^m(t, Y)
\]

where \( t \) represents the number \( |I| + 1 \).

Query Complexity of Datalog: EXPTIME-hardness

Ideas for lifting \( P(M, N) \) and \( DB(I, N) \) to \( P_{dat}(M, I, N) \) and \( DB_{dat} \):

- Use the predicates \( symbol_0(X, Y) \), \( cursor(X, Y) \) and \( state_0(X) \) instead of the propositional letters \( symbol[X, Y], cursor[X, Y] \) and \( state_0[X] \) respectively.

- W.l.o.g., let \( N \) be of the form \( N = 2^m - 1 \) for some integer \( m \geq 1 \).

The time points \( \tau \) and tape positions \( \pi \) from 0 to \( N \) are encoded in binary, i.e. by \( m \)-ary tuples \( t_\tau = (c_1, \ldots, c_m), c_i \in \{0, 1\}, i = 1, \ldots, m \), such that \( 0 = (0, \ldots, 0), 1 = (0, \ldots, 1), N = (1, \ldots, 1) \).

- The functions \( \tau + 1 \) and \( \pi + d \) are realized by means of the successor \( Succ^m \) from a linear order \( \leq^m \) on \( \{0, 1\}^m \).
Defining $Succ^m(X, X')$ and $\leq^m$

- The ground facts $Succ^1(0, 1)$, $First^1(0)$, and $Last^1(1)$ are provided in $DB_{dat}$.
- For an inductive definition, suppose $Succ^i(X, Y)$, $First^i(X)$, and $Last^i(X)$ tell the successor, the first, and the last element from a linear order $\leq^i$ on $\{0, 1\}^i$, where $X$ and $Y$ have arity $i$.

Then, use rules

\[
\begin{align*}
Succ^{i+1}(Z, X, Z, Y) & \leftarrow Succ^i(X, Y) \\
Succ^{i+1}(Z, X, Z', Y) & \leftarrow Succ^i(Z, Z'), Last^i(X), First^i(Y) \\
First^{i+1}(Z, X) & \leftarrow First^i(Z), First^i(X) \\
Last^{i+1}(Z, X) & \leftarrow Last^i(Z), Last^i(X)
\end{align*}
\]

Defining $Succ^m(X, X')$ and $\leq^m$

- The ground facts $Succ^1(0, 1)$, $First^1(0)$, and $Last^1(1)$ are provided in $DB_{dat}$.
- For an inductive definition, suppose $Succ^i(X, Y)$, $First^i(X)$, and $Last^i(X)$ tell the successor, the first, and the last element from a linear order $\leq^i$ on $\{0, 1\}^i$, where $X$ and $Y$ have arity $i$.

Alternatively, use rules

\[
\begin{align*}
Succ^{i+1}(0, X, 0, Y) & \leftarrow Succ^i(X, Y) \\
Succ^{i+1}(1, X, 1, Y) & \leftarrow Succ^i(X, Y) \\
Succ^{i+1}(0, X, 1, Y) & \leftarrow Last^i(X), First^i(Y) \\
First^{i+1}(0, X) & \leftarrow First^i(X) \\
Last^{i+1}(1, X) & \leftarrow Last^i(X)
\end{align*}
\]

The order $\leq^m$ is easily defined from $Succ^m$ by two clauses

\[
\begin{align*}
\leq^m(X, X) & \leftarrow \\
\leq^m(X, Y) & \leftarrow Succ^m(X, Z), \leq^m(Z, Y)
\end{align*}
\]

Combined and Query Complexity of Datalog: Conclusion

- Let $L$ be an arbitrary language in EXPTIME, i.e., there exists a Turing machine $M$ deciding $L$ in exponential time. Then there is a constant $k$ such that $M$ accepts/rejects every input $l$ within $2^{\ell^k}$ steps.
- The program $P_{dat}(M, I, |I|^k)$ is constructible from $M$ and $I$ in polynomial time (in fact, careful analysis shows feasibility in logarithmic space).
- $accept$ is in the answer of $P_{dat}(M, I, |I|^k)$ evaluated over $DB_{dat}$ $\iff$ $M$ accepts input $I$ within $N$ steps.
- Thus the EXPTIME-hardness follows.
Complexity of Datalog with Stratified Negation

Theorem

Reasoning in stratified ground Datalog programs with negation is P-complete. Stratified Datalog with negation is

- P-complete w.r.t. data complexity, and
- EXPTIME-complete w.r.t. combined and query complexity.

- A ground stratified program $P$ can be partitioned into disjoint sets $S_1, \ldots, S_n$ s.t. the semantics of $P$ is computed by successively computing in polynomial time the fixed-points of the immediate consequence operators $T_{S_1}, \ldots, T_{S_n}$.
- As with plain Datalog, for programs with variables, the grounding step causes an exponential blow-up.

Learning Objectives

- The BQE, QOT and QNE problems
- The notions of combined, data and query complexity
- The complexity of first-order queries
- The complexity of conjunctive queries
- The complexity of plain and stratified Datalog