Complexity and Expressive Power of Datalog

Datalog Programs

- A Datalog Program P consists of a finite set of rules of form

$$A_0 \leftarrow A_1, \ldots, A_m \qquad (m \geq 0),$$

where each A_i is a positive atom of the form $r(t_1, \ldots, t_k)$ where each t_i is a variable or a constant.

- Two important settings

1. Datalog programs are “stand alone”. Program may contain variables and constants.
2. Datalog programs operate over factual databases. The database contains *ground facts*, no constants occur within the program. Distinction between EDB and IDB Predicates.

Example of stand-alone Datalog

- Datalog program:

$$
\begin{aligned}
\textit{parent}(X,Y) &\ \text{:-}\ \textit{father}(X,Y) \\
\textit{parent}(X,Y) &\ \text{:-}\ \textit{mother}(X,Y) \\
\textit{ancestor}(X,Y) &\ \text{:-}\ \textit{parent}(X,Y) \\
\textit{ancestor}(X,Y) &\ \text{:-}\ \textit{parent}(X,Z), \textit{ancestor}(Z,Y) \\
\textit{person}(X) &\ \text{:-} \\
\textit{father}(\textit{john},\textit{mary}) &\ \text{:-} \\
\textit{father}(\textit{joe},\textit{kurt}) &\ \text{:-} \\
\textit{mother}(\textit{mary},\textit{joe}) &\ \text{:-} \\
\textit{mother}(\textit{tina},\textit{kurt}) &\ \text{:-}
\end{aligned}
$$

Datalog as a Query Language

- Datalog is used as a database query language
- In this context, a datalog program is evaluated over a *database*, which is a set facts.
- Programs are composed of a "derived" part P (defined predicates) and an "input part" D_{in} (database facts): $P \cup D_{in}$

Example:

$$\left.\begin{array}{rl} \textit{parent}(X,Y) & \textbf{:-}\ \textit{father}(X,Y) \\ \textit{parent}(X,Y) & \textbf{:-}\ \textit{mother}(X,Y) \\ \textit{ancestor}(X,Y) & \textbf{:-}\ \textit{parent}(X,Y) \\ \textit{ancestor}(X,Y) & \textbf{:-}\ \textit{parent}(X,Z), \textit{ancestor}(Z,Y) \\ \textit{person}(X) & \textbf{:-} \end{array}\right\} \text{defined part } P$$

$$\left.\begin{array}{rl} \textit{father}(\textit{john},\textit{mary})\ \textbf{:-} & \textit{father}(\textit{joe},\textit{kurt})\ :- \\ \textit{mother}(\textit{mary},\textit{joe})\ \textbf{:-} & \textit{mother}(\textit{tina},\textit{kurt})\ :- \end{array}\right\} \text{database part } D_{in}$$

Refined Notions of Datalog Complexity

- The ***data complexity*** is the complexity of checking whether $D_{in} \cup P \models A$ when datalog programs P are *fixed*, while input databases D_{in} and ground atoms A are an *input*.

- The ***program complexity*** (also called *expression complexity*) is the complexity of checking whether $D_{in} \cup P \models A$ when input databases D_{in} are *fixed*, while datalog programs P and ground atoms A are an *input*.

- The ***combined complexity*** is the complexity of checking whether $D_{in} \cup P \models A$ when input databases D_{in}, datalog programs P, and ground atoms A are an *input*.

Semantics of Datalog as a Query Language

The semantics of a datalog program P is defined by reduction to the propositional case (by “Grounding”)

- Let P be a datalog program operating on a database D.
- Let U_D be the universe of D (usually the active universe, i.e., the set of all domain elements present in D).
- The *grounding* of a rule r, denoted *ground*(r, D), is the set of all rules obtained from r by all possible uniform substitutions of elements of U_D for the variables in r.

Semantics of Datalog

- For any datalog program P and database D,

$$\textit{ground}(P,D) = \bigcup_{r \in P} \textit{ground}(r,D).$$

- If S is a set of atoms then $IDB_P(S)$ denotes those facys of S whose predicate symbol is an IDB predicate symbol of P.

- The semantics of P is given by

$$\mathcal{M}_P : D \rightarrow IDB_P(T^{\infty}_{ground(P,D)\cup D}).$$

Examples /2

Program P:

$$\textit{parent}(X, Y) \text{ :- } \textit{father}(X, Y)$$
$$\textit{parent}(X, Y) \text{ :- } \textit{mother}(X, Y)$$
$$\textit{ancestor}(X, Y) \text{ :- } \textit{parent}(X, Y)$$
$$\textit{ancestor}(X, Y) \text{ :- } \textit{parent}(X, Z), \textit{ancestor}(Z, Y)$$
$$\textit{person}(X) \text{ :- }$$
$$\textit{father}(\textit{john}, \textit{mary}) \text{ :- } \quad \textit{father}(\textit{joe}, \textit{kurt}) \longleftarrow$$
$$\textit{mother}(\textit{mary}, \textit{joe}) \text{ :- } \quad \textit{mother}(\textit{tina}, \textit{kurt}) \longleftarrow$$

ground(P):

$$\textit{parent}(\textit{john}, \textit{john}) \text{ :- } \textit{father}(\textit{john}, \textit{john})$$
$$\textit{parent}(\textit{john}, \textit{mary}) \text{ :- } \textit{father}(\textit{john}, \textit{mary})$$
$$\ldots$$
$$\textit{parent}(\textit{john}, \textit{john}) \text{ :- } \textit{mother}(\textit{john}, \textit{john})$$
$$\textit{parent}(\textit{john}, \textit{mary}) \text{ :- } \textit{mother}(\textit{john}, \textit{mary})$$
$$\ldots$$
$$\textit{ancestor}(\textit{john}, \textit{john}) \text{ :- } \textit{parent}(\textit{john}, \textit{john})$$
$$\ldots$$
$$\textit{father}(\textit{john}, \textit{mary}) \text{ :- } \quad \textit{father}(\textit{joe}, \textit{kurt})$$
$$\textit{mother}(\textit{mary}, \textit{joe}) \text{ :- } \quad \textit{mother}(\textit{tina}, \textit{kurt})$$

- Herbrand Universe: *john*,*mary*, *joe*,*kurt*, *tina*
- Herbrand Base: *person*(*john*) *person*(*mary*), . . . , *parent*(*john*,*john*), *parent*(*john*,*mary*), . . .
- $LM(P)$ = { *father*(*john*,*mary*), *father*(*joe*,*kurt*), *mother*(*mary*,*joe*), *mother*(*tina*,*kurt*), *parent*(*john*,*mary*), . . . , *ancestor*(*john*,*mary*), . . . , *person*(*john*), . . . *person*(*tina*),. . . }

Complexity of Datalog Programs

- For Datalog programs, both "$A \in lm(P)$" is decidable, similarly "$A \in lm(P \cup D)$" in case P operates on a database D.
- **Reason:** *Ground*(P) is finite (as U_P, B_P are finite)

  Effective reduction to Propositional Logic Programming is possible:
  - Generate *Ground*(P)
  - Decide whether $A \in lm($*Ground*$(P))$
- **Questions:**
  - What is the complexity of this algorithm? (Key: How expensive is computing *Ground*(P)?)
  - Is this the best algorithm to decide $A \in lm(P)$?

Complexity of Grounding Strategy

- Given P, D, the number of rules in *ground*(P, D) is bounded by

$$|P| * \#consts(D)^{vmax}$$

  - $vmax$ (≥ 1) is the maximum number of different variables in any rule $r \in P$
  - $\#consts(P) = |U_D|$ is the number of constants in D (ass.: $|U_D| > 0$).

- *ground*(P, D) can be naively generated in time

$$O(|P| * \#consts(D)^{vmax}) = O(2^{\log |P| + vmax * \log \#consts(D)}) = O(2^{p(\|P \cup D\|)}),$$

where $p(\dots)$ is some polynomial and $\|P \cup \|$ is the size of $P \cup D$.

- Therefore, $A \in lm(P \cup D)$ is decidable in *exponential time*.

- **Observation:** *ground*$(P \cup D)$ can be *exponential* in the size of P.

- **Question:** Is $A \in lm(P)$ feasible in polynomial space ?

EXPTIME-Completeness of Datalog Case

Theorem. Given a positive Datalog program P and a ground atom A, deciding whether $A \in lm(P)$ is **EXPTIME**-complete.

Proof Sketch.

- Membership: By reduction to propositional case (grounding)
- Hardness:
  - Adapt the propositional program $P(T, I, N)$ deciding acceptance of input I for T within N steps, where $N = 2^m$, $m = n^k$ ($n = |I|$) to a datalog program $P_{dat}(T, I, N)$
  - Note: We can't simply generate $P(T, I, N)$, since this program is exponentially large (and thus the reduction would not be polynomial!)

EXPTIME-Hardness of Datalog Programs

Main ideas for lifting $P(T, I, N)$ to $P_{dat}(T, I, N)$:

- Use predicates *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 encoded in binary, i.e. by m-ary tuples $t_\tau = \langle c_1, ..., c_m \rangle$, $c_i \in \{0, 1\}$, $i = 1, \ldots, m$, such that $0 = \langle 0, ..., 0 \rangle$, $1 = \langle 0, ..., 1 \rangle$, $\ldots$, $N - 1 = \langle 1, ..., 1 \rangle$
- 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 in P.

Modification for Datalog-Complexity Hardness

Modify the program $P(T, I, N)$ as follows ($N = 2^m$, where $m = n^k$):

- 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*$_\sqcup[0, \pi]$, where $|I| \leq \pi \leq N$, to the rule

$$\textit{symbol}_\sqcup(\vec{x}, \vec{y}) \texttt{ :- } \textit{first}^m(\vec{x}),\ \leq^m(\vec{t}, \vec{y})$$

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

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

  Example:

  $$\textit{symbol}_{\sigma'}[\tau + 1, \pi] \texttt{ :- } \textit{state}_s[\tau], \textit{symbol}_{\sigma}[\tau, \pi], \textit{cursor}[\tau, \pi]$$

  is translated into

  $$\textit{symbol}_{\sigma'}(\vec{x}', \vec{y}) \texttt{ :- } \textit{state}_s(\vec{x}), \textit{symbol}_{\sigma}(\vec{x}, \vec{y}), \textit{cursor}(\vec{x}, \vec{y}), \textit{succ}^m(\vec{x}, \vec{x}').$$

- accept rules: translation is straightforward.

Defining $succ^m$ and $\leq^m$

- Add facts $\mathit{succ}^1(0,1)$, $\mathit{first}^1(0)$, and $\mathit{last}^1(1)$.
- Inductively define succ^{i+1}:

$$\begin{aligned} \mathit{succ}^{i+1}(z,\vec{x},z,\vec{y}) &\text{ :- } \mathit{succ}^i(\vec{x},\vec{y}) \\ \mathit{succ}^{i+1}(z,\vec{x},z',\vec{y}) &\text{ :- } \mathit{succ}^1(z,z'), \mathit{last}^i(\vec{x}), \mathit{first}^i(\vec{y}) \\ \mathit{first}^{i+1}(z,\vec{x}) &\text{ :- } \mathit{first}^1(z), \mathit{first}^i(\vec{x}) \\ \mathit{last}^{i+1}(z,\vec{x}) &\text{ :- } \mathit{last}^1(z), \mathit{last}^i(\vec{x}) \end{aligned}$$

(where $\vec{x} = x_1, \ldots, x_i$, $\vec{y} = y_1, \ldots, y_i$, and $\vec{z} = z_1, \ldots, z_i$.)

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

$$\begin{aligned} \leq^m(\vec{x},\vec{x}) &\text{ :- } \\ \leq^m(\vec{x},\vec{y}) &\text{ :- } \mathit{succ}^m(\vec{x},\vec{z}),\ \leq^m(\vec{z},\vec{y}) \end{aligned}$$

($\vec{x} = x_1, \ldots, x_m$, $\vec{y} = y_1, \ldots, y_m$, and $\vec{z} = z_1, \ldots, z_m$.)

Concluding EXPTIME Hardness of Datalog

Let $P_{dat}(T, I, N)$ denote the datalog program with empty edb described for T, I, and $N = 2^m$, $m = n^k$ (where $n = |I|$)

- $P_{dat}(T, I, N)$ is constructible from T and I in polynomial time (in fact, careful analysis shows feasibility in logarithmic space).

- $P_{dat}(T, I, N)$ has *accept* in its least model $\Leftrightarrow$ T accepts input I within N steps.

- Thus, the decision problem for any language in **EXPTIME** is reducible to deciding $P \models A$ for datalog program P and fact A.

- Consequently, deciding $P \models A$ for a given datalog program P and fact A is **EXPTIME**-hard.

Program and Combined Complexity

- Clearly, combined complexity matches the problem $P \models A$ we considered so far $\Rightarrow$ Datalog is **EXPTIME**-complete w.r.t. combined complexity.
- As for program complexity, **EXPTIME** is an upper bound
- From the **EXPTIME**-hardness proof of $P \models A$, we can conclude that Datalog is **EXPTIME**-hard w.r.t. program complexity (take empty D_{in}).
- This can be sharpened to instances where program P contains no constants (take D_{in} to be *succ*$^1(0, 1)$, *first*$^1(0)$, and *last*$^1(1)$.)

Data Complexity

- For fixed P, the grounding *ground*$(D_{in} \cup P)$ has size *polynomial* in the size of $D_{in} \cup P$ ($|P| * \#consts(P)^{vmax}$) = $O(\|P\|^k)$ for some constant k).
- Moreover, *ground*$(D_{in} \cup P)$ can be easily generated in polynomial time
- Therefore, $LM(D_{in} \cup P)$ is computable in polynomial time, and Datalog has polynomial-time data complexity.
- Furthermore, $P \models A$ is **P**-hard w.r.t. data complexity. This can be shown by proving that a fixed datalog program is able to act as a meta-interpreter for propositional logic programming.

A Datalog Meta-Interpreter for Propositional LP

Note: It is sufficient to interpret propositional logic programs whose clauses have at most 3 atoms in the rule bodies. In fact, we have shown that atom-inference from such programs is P-hard.

Encode a propositional LP as follows by a unary relation T_0 and a 4-ary relation R.

Encoding of facts: The fact "$p \longleftarrow$" is encoded by the tuple $T(p)$.

Encoding of rules: A rule "$p \leftarrow q_1, q_2, q_3$" is encoded by the tuple $R(p, q_1, q_2, q_3)$. In case a rule has less than 3 atoms in its body, a body-atom can be repeated to get a tuple of length 4.

This encoding of a propositional logic program P, which is obviously feasible in logspace, is denoted by $D(P)$.

The meta-interpreter M:

$$T(X_0) \texttt{ :- } R(X_0, X_1, X_2, X_3), T(X_1), T(X_2), T(X_3)$$

$$T(X) \texttt{ :- } T_0(X)$$

We have $P \models A$ iff $M \cup D(P) \models T(A)$.

Therefore the data complexity od datalog is PTIME-complete.

Semipositive Datalog (Datalog$^{\perp}$)

So far, only positive atoms were allowed in rule bodies.

We are going to define a slight extension.

Semipositive datalog programs: EDB-atoms in rule bodies may occur both in positive and negated form. IDB-atoms cannot be negated.

Semantics: Obvious. Let P be a semipositive program and D a database. Add the complement relation $\overline{r}$ for each relation r to the database, yielding D^+. Replace each atom $\neg r(\mathbf{x})$ in a rule body by $\overline{r}(\mathbf{x})$, yielding P^+. Then:

$$P(D) := P^+(D^+).$$

We denote semipositive datalog by datalog$^{\perp}$.

Expressive Power of Semipositive Datalog

A *successor ordering* of a structure consists of a successor relation $Succ$ on its universe and special relations Min and Max with the obvious meanings.

THEOREM: On structures provided with a successor ordering, datalog$^{\perp}$ = PTIME.

PROOF SKETCH:

We outline this for ordered graphs $G = (V, Succ, Min, Max, E)$.

We have to show that each PTIME property over such databases can be encoded by a semipositive datalog program.

Let us assume some property π is computable in time n^k, where $n = |V|$. There must exist a Turing machine T that does this job on a suitable binary encoding of G. Our intention is to simulate (the behaviour of) T by a datalog$^{\perp}$ program.

Ideas:

1.) We use vectors $\vec{x} = (x_1, \ldots, x_k)$ to encode time instants and workhead position (cell numbers). Here the arguments range over all domain elements from V, and hence we can encode exactly $|V|^k = n^k$ elements (or numbers) with each such vector.

2.) We define a vectorized successor relation $succ^k(\vec{x}, \vec{y})$ on vectors of length k in a similar way as we did it before for binary vectors. (Iteratively, by defining $succ^i$ for $i = 0 \ldots k$, and based on the Min, Max, and $Succ$ predicates).

3.) We put the graph G on the (datalog-simulated) input tape of the datalog-simulated Turing machine T that runs in time n^k by using the following binary encoding $\vec{e}$ of E. E is encxoded as a bit vector $\vec{e}$ of size n^2 such that $\vec{e}[i * n + j]$ is 1 iff $(i, j) \in E$ and 0 otherwise.

This vector $\vec{e}$ is “put on the input tape” by the following 2 rules:

symbol$_1(0^k, 0^{k-2}, X, Y)$ **:-** $E(X, Y)$
symbol$_0(0^k, 0^{k-2}, X, Y)$ **:-** $\neg E(X, Y)$

4.) We simulate T on this input in the usual way. Note that the resulting program is semipositive.

QED

Bibliography

[1] 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/`.

[2] T. Eiter and G. Gottlob. Expressiveness of Stable Model Semantics for Disjunctive Logic Programs with Functions. *Journal of Logic Programming*, 33(2):167–178, 1997.

[3] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, September 1997.

[4] G. Gottlob, N. Leone, and H. Veith. Succinctness as a Source of Expression Complexity. *Annals of Pure and Applied Logic*, 97(1–3):231–260, 1999.

[5] P. Kolaitis and C. H. Papadimitriou. Why Not Negation By Fixpoint ? *Journal of Computer and System Sciences*, 43:125–144, 1991.

[6] V. W. Marek and J. B. Remmel. On the expressibility of stable logic programming. *Journal of the Theory and Practice of Logic Programming*, 3:551–567, Nov. 2003.

[7] J. Minker and D. Seipel. Disjunctive logic programming: A survey and assessment. In A. Kakas and F. Sadri, editors, *Computational Logic: From Logic Programming into the Future*, number 2407 in LNCS/LNAI, pages 472–511. Springer Verlag, 2002. Festschrift in honour of Bob Kowalski.

[8] J. Schlipf. The Expressive Powers of Logic Programming Semantics. *Journal of Computer and System Sciences*, 51(1):64–86, 1995. Abstract in Proc. PODS 90, pp. 196–204.

[9] J. Schlipf. Complexity and Undecidability Results in Logic Programming. *Annals of Mathematics and Artificial Intelligence*, 15(3/4):257–288, 1995.