On Datalog, LFP and MSO

Anuj Dawar

University of Cambridge

joint work with Stephan Kreutzer (Oxford)

LIAFA, 20 March 2009

Datalog

Datalog is a language originally introduced for defining *deductive databses*. It is also alternatively viewed as a *query language* for relational databases.

It is the language of function-free Horn clauses or *equivalently*, the closure of existential positive first-order logic under recursive definitions.

A database is a *finite relational structure* in some vocabulary ore relations σ. The relations in σ are the *extensional database predicates* (EDBs).

In addition, a number of *intensional database predicates* (IDBs) are defined by means of a Datalog *program*.

Datalog Programs

A Datalog program is a series of rules of the form

$$H(\mathbf{x}) \leftarrow B_1(\mathbf{x}_1), \ldots, B_m(\mathbf{x}_m)$$

where H is an IDB predicate, each B_i is an EDB or an IDB predicate (possibly H itself) and $\mathbf{x}$ and $\mathbf{x}_i$ are tuples of variables and constants.

The semantics defines the interpretation of an IDB predicate H on a given structure $\mathbb{A}$ to be the set of tuples that can be placed in H by a finite sequence of application of the rules.

To define a fixed *query*, we specify one IDB predicate to be the *goal predicate*

Example

Given one binary EDB predicate E, the following program defines its transitive closure.

$$\begin{aligned} T(x,y) &\leftarrow E(x,y) \\ T(x,y) &\leftarrow E(x,z), T(z,y) \end{aligned}$$

To formulate the query that there is an E-path from s to t, we could add the 0-ary goal predicate G and the rule.

$$G \leftarrow T(s,t)$$

LFP

Datalog can be seen as allowing recursive definitions by means of formulas involving

- *conjunction* (in the form of the comma);
- *disjunction* (in the form of multiple rules); and
- *existential quantification* (in the form of projection of variables)

If we allow other operations of first-order logic, including *universal quantification* and *negation* (but only on EDB predicates and equalities), we get LFP.

By disallowing negation on IDBs, we ensure that the recursion is still well-defined.

Example

On a relational structure with unary relations E (for existential nodes) and A (universal nodes) and a binary relation S, the following program defines the pairs of elements (x, y) with an alternating S-path.

$$
\begin{aligned}
\mathsf{AP}(x,y) &\leftarrow x = y \\
\mathsf{AP}(x,y) &\leftarrow E(x) \wedge \exists z(S(x,z) \wedge \mathsf{AP}(z,y)) \\
\mathsf{AP}(x,y) &\leftarrow A(x) \wedge \forall z(S(x,z) \rightarrow \mathsf{AP}(z,y))
\end{aligned}
$$

LFP and PTIME

An important reason for the interest in LFP is its connection to *polynomial-time computation*.

Every query definable in LFP is computable in P.

If we consider structures with a *linear order* as a built-in relation, then every query computable in P is definable in LFP.

Homomorphisms

Given two structures $\mathbb{A}$ and $\mathbb{B}$ over a vocabulary σ, a homomorphism h is a map from $\mathbb{A}$ to $\mathbb{B}$ such that

- $h(c^{\mathbb{A}}) = c^{\mathbb{B}}$ for each constant symbol c in σ;
- $R^{\mathbb{A}}(\mathbf{a}) \Rightarrow R^{\mathbb{B}}(h(\mathbf{a}))$ for each relation symbol R in σ and each tuple of elements $\mathbf{a}$.

Every existential-positive first-order formula φ is preserved under homomorphisms, i.e.

$$\mathbb{A} \models \varphi[\mathbf{a}] \quad \Rightarrow \quad \mathbb{B} \models \varphi[h(\mathbf{a})]$$

And the same easily follows for all Datalog queries.

Preservation Theorems

A classical *preservation theorem* of model theory tells us that every first-order formula that is preserved under homomorphisms is equivalent to an existential-positive formula.

A result by **(Rossman 2005)** states that this remains true, even when restricted to finite structures.

Atserias (MathCSP 2006) asked whether a similar result could be shown for fixed-point logics:

> Is every query that is definable in LFP and closed under homomorphisms definable in Datalog?

Failure of Preservation

The answer is *no* **(D.-Kreutzer, ICALP 08)**.

> If we admit *infinite structures*, we can construct an LFP formula, preserved under homomorphisms, which we can show is not Datalog definable by a method of *diagonalisation*.

Establshing the result when only *finite* structures are permitted is more of a challenge.

Finite Structures

For any set $S \subseteq \mathbb{N}$, we define the class of *directed source-target graphs* (i.e. structures over (E, s, t))

C_S: the class of graphs (G, s, t) such that either G contains a cycle, or there is a path from s to t of length p for some $p \in S$.

We prove:

- C_S is closed under homomorphisms.
- If S is decidable in P (with numbers represented in unary), then C_S is definable in LFP.
- C_S is not definable in Datalog when

$$S = \{n \mid n = 2^{2^{m^2}} \text{ for some } m \in \mathbb{N}\} .$$

Closure under Homomorphisms

C_S: the class of graphs (G, s, t) such that either G contains a cycle, or there is a path from s to t of length p for some $p \in S$.

If h is a homomorphism from (G, s, t) to (H, s, t) then

- If G contains a cycle, the image of this cycle under h is a cycle in H.
- If G contains a simple path of length p from s to t, the image of this path under h either contains a cycle, or is a simple path of length p.

Definability in LFP

There is an LFP formula that defines graphs that contain a cycle.

For an *acyclic* graph G, let n^G denote the length of the longest simple path in G.

We can show that there are LFP formulas that interpret, in any acyclic G a linear order of length n^G along with the relation

(u, v, p) : there is a path of length p from u to v.

By the fact that LFP capture P on ordered structures, we get:

If S is decidable in P (with numbers represented in unary), then C_S is definable in LFP.

A Pumping Lemma for Datalog

To show: C_S is not definable in Datalog when

$$S = \{n \mid n = 2^{2^{m^2}} \text{ for some } m \in \mathbb{N}\} .$$

We prove the following:

> If there is a Datalog program which accepts (G, s, t) if, and only if, G contains a path from s to t of length p for some $p \in S$, then there is a constant c and an increasing sequence $(a_i)_{i \in \mathbb{N}}$ of numbers such that:
>
> 1. $a_{i+1} < a_i^c$ for all i; and
> 2. $S \cap [a_i, a_{i+1}] \neq \emptyset$ for all i.

In contrast, note that we can define a Datalog program that accepts a simple path (P, s, t) iff its length is in S.

Unfoldings of Datalog Programs

We illustrate *unfoldings* of a Datalog program with an example.

Among the (infinitely many) unfoldings of

$$\begin{aligned} T(x,y) &\leftarrow E(x,y) \\ T(x,y) &\leftarrow E(x,z), T(z,y) \\ G &\leftarrow T(s,t) \end{aligned}$$

are:

$E(s,t)$

$E(s,z_1), E(z_1,t)$

$E(s,z_1), E(z_1,z_2), E(z_2,t)$

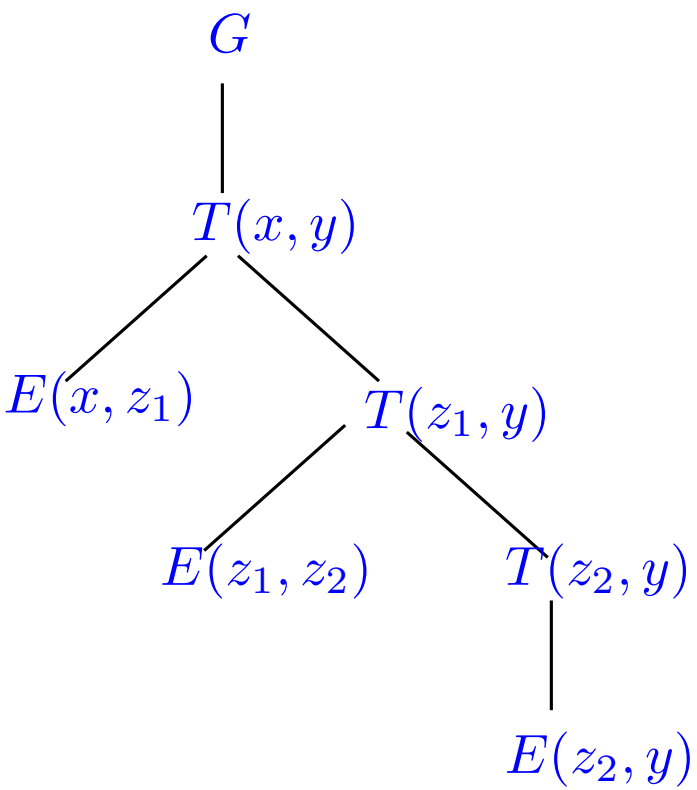
More formally, an unfolding is obtained from the *goal* by repeatedly replacing an occurrence of an IDB predicate by the right-hand side of a rule defining it (while renaming bound variables) until only EDB predicates remain.

Facts about Unfoldings

- Each unfolding of a program π can be itself seen as a relational structure $\mathbb{A}$ on the variables. We call such structures *expansions* of π.
- A structure $\mathbb{B}$ is accepted by π if, and only if, there is some expansion $\mathbb{A}$ of π with a homomorphism $h : \mathbb{A} \longrightarrow \mathbb{B}$.
- There is a k depending only on π such that all expansions of π have *treewidth* $\leq k$. Indeed, the number of distinct variables in π provides an upper bound, with the unfolding yielding a decomposition.

Decorated Expansions

We can represent an unfolding of π as a labelled tree.



In the labels of the nodes, it suffices to record the rule of the program used, and the equalities between variables in the node and those in its children.

As there are at most k variables per rule, there are only finitely many labels (depending on π).

MSO definability

For any Datalog program π we can write an MSO formula that defines those *labelled trees* that are *decorated expansions* of π.

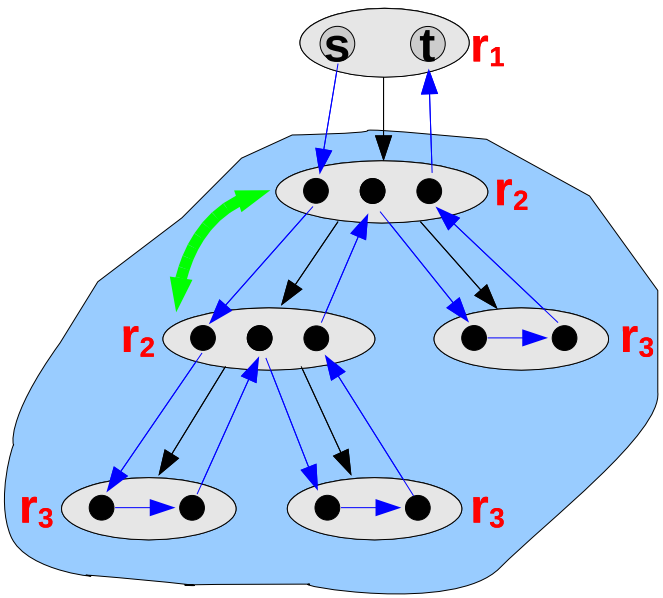
We can also say in MSO that the underlying *expansion* is an acyclic graph.

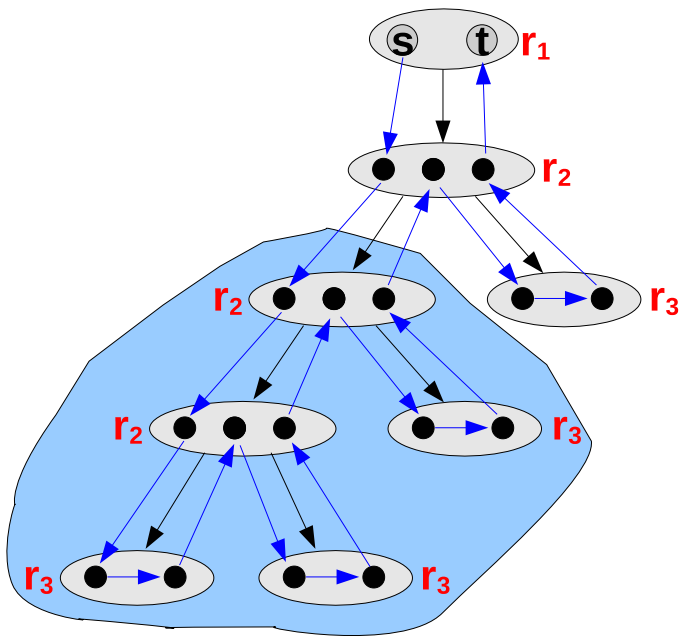
Thus, there is a *finite tree automaton* that accepts the decorated expansions of π with acyclic underlying expansion, and we will use this for our pumping argument.

Simple Paths

Suppose the program π accepts a *simple path* of length p.

If the *expansion accepting the path* is also a simple path, and p is large enough, we can pump, obtaining another expansion that is also a simple path but whose length l is strictly between p and $p + c$.

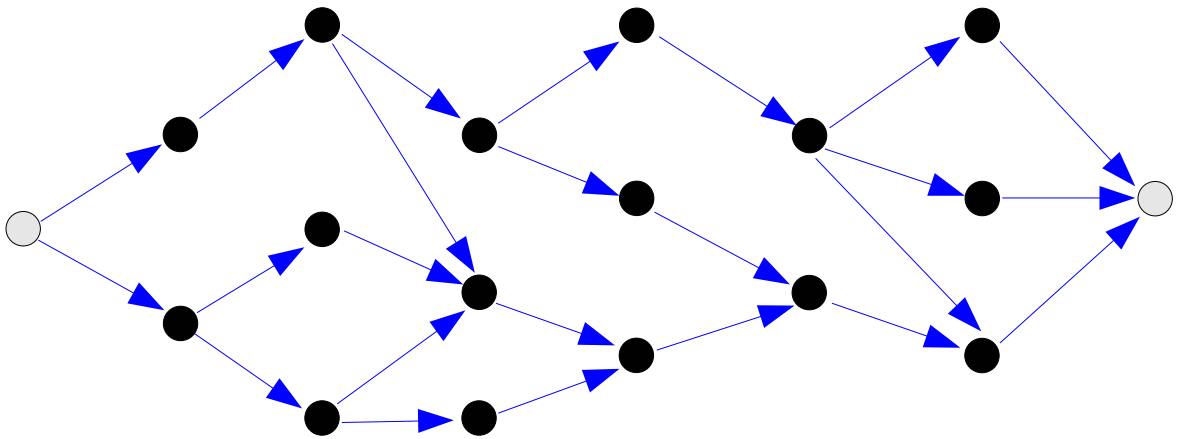




(Afrati, Cosmadakis, Yannakakis)

Acyclic Graph Expansions

In general, the expansion witnessing that π accepts a simple path will not itself be a simple path. It may be a some other graph that maps homomorphically to the simple path.



Pumping the decorated expansion corresponding to such a graph is not guaranteed to increase the length of all paths from s to t.

Indeed, it may even introduce shorter paths.

Pumping at Multiple Points

We show that in any decorated expansion witnessing that π accepts a simple path of length p (for large enough values of p), we can find a set of pairs of points $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ such that:

- for each i, the subtree rooted at x_i has the same type as the one at y_i;
- every long enough path has an edge (in a precise sense) in "the portion of the decorated expansion between" x_i and y_i.

We define a way of *pumping* the tree at all pairs (x_i, y_i) at the same time, and show that if this is done n times then:

- any path from s to t has length at least n;
- there is a c (depending only on the branching degree and the width of the decomposition), so that the total size of the expansion has size at most n^c.

Pumping Result

If there is a Datalog program which accepts (G, s, t) if, and only if, G contains a path from s to t of length p for some $p \in S$, then there is a constant c and an increasing sequence $(a_i)_{i \in \mathbb{N}}$ of numbers such that:

1. $a_{i+1} < a_i^c$ for all i; and
2. $S \cap [a_i, a_{i+1}] \neq \emptyset$ for all i.

C_S is not definable in Datalog when

$$S = \{n \mid n = 2^{2^{m^2}} \text{ for some } m \in \mathbb{N}\} .$$

Question: Can we replace a_i^c in the statement by $c \cdot a_i$ or even $c + a_i$?

This would allow us to define C_S with $n = 2^{m^2}$ or $n = m^2$.

On Datalog and MSO

For a formula φ of MSO, let $\mathcal{M}^k_\varphi$ denote the collection of structures $\mathbb{A}$ with $\mathrm{tw}(\mathbb{A}) \leq k$ and $\mathbb{A} \models \varphi$.

Write $\mathcal{H}^k_\varphi$ for the closure of $\mathcal{M}^k_\varphi$ under homomorphisms, i.e. the class of structures $\mathbb{B}$ with a homomorphism $h : \mathbb{A} \to \mathbb{B}$ for some $\mathbb{A} \in \mathcal{M}^k_\varphi$.

Fact: $\mathcal{H}^k_\varphi$ is Datalog-definable.

> This can be shown using Courcelle's construction to obtain from φ an automaton that works on tree-decompositions of structures of treewidth at most k and accepts those that are in $\mathcal{M}^k_\varphi$. The transitions of the automaton can then be defined by Datalog rules, yielding a program that defines $\mathcal{H}^k_\varphi$.

Could it be that *every* Datalog-definable class is of the form $\mathcal{H}^k_\varphi$ for some k and some MSO φ?

Defining Expansions

In our pumping argument, we used the fact that the class of decorated expansions of a Datalog program π is MSO definable.

Could it be that the class of *expansions* of π is itself MSO definable?

This would allow for an elegant result tying the expressive power of Datalog to that of MSO on classes of bounded treewidth:

> A class of structures $\mathcal{C}$ is Datalog definable if, and only if, the homomorphism closure of some MSO-definable class of bounded treewidth.