Expressivity of Datalog Variants – Completing the Picture

Sebastian Rudolph, Michaël Thomazo

Expressivity of Datalog Variants – Completing the Picture

Sebastian Rudolph
TU Dresden
Germany
sebastian.rudolph@tu-dresden.de

Michaël Thomazo
Inria
France
michael.thomazo@inria.fr

Abstract

Computational and model-theoretic properties of logical languages constitute a central field of research in logic-based knowledge representation. Datalog is a very popular formalism, a de-facto standard for expressing and querying knowledge. Diverse results exist regarding the expressivity of Datalog and its extension by input negation (semipositive Datalog) and/or a linear order (order-invariant Datalog). When classifying the expressivity of logical formalisms by their model-theoretic properties, a very natural and prominent such property is preservation under homomorphisms. This paper solves the remaining open questions needed to arrive at a complete picture regarding the interrelationships between the class of homomorphism-closed queries and the query classes related to the four versions of Datalog. Most notably, we exhibit a query that is both homomorphism-closed and computable in polynomial time but cannot be expressed in order-invariant Datalog.

1 Introduction

Various logical languages have been defined to formalize and query knowledge. A central topic of logic-based knowledge representation (KR) is to analyze and compare these languages regarding expressivity and computational properties.

Among the most prominent KR formalisms is Datalog. Even in its plain form, Datalog serves as basis for KR languages (such as the popular OWL 2 RL profile of the Web Ontology Language) and also is seen as a common subsumer for a variety of very expressive query languages (cf. [Bourhis, Krötzsch, and Rudolph, 2014; 2015]). Moreover, it is quite often used as a target for knowledge compilation from much more expressive KR languages (for instance, a recent topic of research has been to show how tractable description logics or classes of existential rules can be reformulated by using Datalog for query answering purposes [Ortiz, Rudolph, and Simkus, 2010; Gottlob and Schwentick, 2012; Cuenca Grau et al., 2013; Kaminski, Nenov, and Cuenca Grau, 2014; Gottlob, Rudolph, and Simkus, 2014]).

Several moderate extensions of plain Datalog have been introduced, in order to enhance its expressivity. In this paper, we focus on two notorious extensions: the ability to use negation for the database predicates (also referred to as *input negation*, resulting in so-called *semipositive Datalog*) and the availability of a linear order on the domain elements (giving rise to *order-invariant Datalog*). This leads to four distinct versions of Datalog-based languages. A very natural question is to study and compare the relative and absolute expressivities of these query languages.

One of the seminal results in that respect is that a query can be computed in polynomial time (PTIME) **exactly if** it can be expressed using semipositive Datalog whenever a linear order on the domain individuals is present and can be accessed by the Datalog program [Abiteboul, Hull, and Vianu, 1994]. Such a clear-cut characterization in the spirit of descriptive complexity [Immerman, 1999] is not available for the other languages, but one can get further insights by restricting the focus to queries satisfying certain model-theoretic properties.

Clearly, by disabling input negation altogether, one loses the capability of detecting the absence of database information, which restricts the expressivity of the formalism to queries satisfying some monotonicity property. For semipositive Datalog without the additional assumption of a linear order, this property has been precisely characterized: removing negation makes one lose exactly those queries not closed under homomorphism. Put positively: any homomorphism-closed query expressible in semipositive Datalog can be expressed in plain Datalog [Feder and Vardi, 2003]. Intuitively, a query (language) is homomorphism-closed if every answer remains valid if more domain elements or relationships are added or if domain elements are identified with each other.

In view of these results, a plausible conjecture would be that **any** homomorphism-closed PTIME computable query can be expressed in plain Datalog. Unfortunately, this conjecture was refuted by Dawar and Kreuzer [Dawar and Kreutzer, 2008] exhibiting such a query but using a pumping argument to show that it cannot be expressed in Datalog. Thus required to revise the conjecture we might suppose that the presence of a linear order is the (only) missing ingredient to make sure that (at least) all homomorphism-closed PTIME-computable queries are captured. This assumption is corroborated by the fact that we can show that the query exhibited by Dawar and Kreuzer [Dawar and Kreutzer, 2008] indeed can be expressed by Datalog when such a linear order is present.

Despite these indications in favor of the conjecture, we

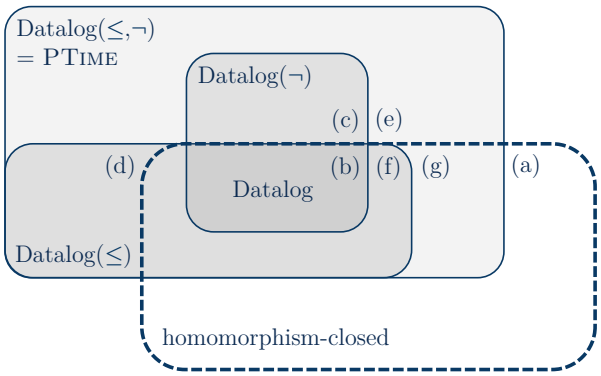

Figure 1: Relationships between the query classes. See Section 5 for details.

show in this paper that, perhaps surprisingly, there exists a PTIME-computable query that is closed under homomorphisms and that cannot be expressed by an order-invariant Datalog query. We first define this query, prove that it has the claimed properties, and then show that an order-invariant Datalog program encoding this query would allow us to build a polynomial family of monotone Boolean circuits that decide the existence of a perfect matching in a graph. This is known to be impossible thanks to a deep result by Razborov.

Combining all the known and newly established findings, we arrive at a complete map of the interrelationships of the five query classes defined by the four Datalog variants plus the class of homomorphism-closed queries. The result of this analysis is summarized by the diagram in Fig. 1.

We proceed as follows: we first recall basic definitions about queries, Datalog and the variants we study. Then, preparing the paper's main result, we introduce the notion of perfect matching in a graph as well as the result regarding the size of Boolean circuit deciding the existence of a perfect matching. Our main technical contribution follows: we introduce a polynomial query, closed under homomorphism, that is not equivalent to a Datalog query on ordered structures. We the summarize known and easy relationships between the four variants we consider. We finally describe some future work.

2 Preliminaries

We consider two countable disjoint sets V and Δ_u of *variables* and *universal domain elements*, respectively. Elements of $V \cup \Delta_u$ are also called *terms*. We consider two finite disjoint sets $\mathcal{P}_i$ and $\mathcal{P}_e$ of *intensional predicates* and *extensional predicates*. Each predicate is either intensional or extensional and possesses an *arity* $n \in \mathbb{N}$.

An *atom* is an expression a of the form $\mathtt{p}(x_1, \ldots, x_n)$ where $\mathtt{p}$ is a predicate of arity n and $x_1, \ldots, x_n$ are terms. The terms of a are denoted by $\mathsf{terms}(a)$. The terms of a set of atoms A are defined by $\bigcup_{a \in A} \mathsf{terms}(a)$.

For a set $\mathcal{P}$ of predicates, a *$\mathcal{P}$-database* (or just *database*, if $\mathcal{P}$ is clear from the context) over some finite domain $\Delta \subseteq \Delta_u$ is a finite set D of atoms with terms from Δ and predicates from $\mathcal{P}$.[1] For a fixed $\mathcal{P}$, given a database D with domain Δ and a database D' with domain Δ', a *homomorphism* from D to D' is a mapping π from Δ to Δ' such that if $\mathtt{p}(x_1, \ldots, x_n) \in D$, then $\mathtt{p}(\pi(x_1), \ldots, \pi(x_n)) \in D'$ for every $\mathtt{p} \in \mathcal{P}$. A *strong homomorphism*[2] from D to D' is a homomorphism π from D to D' such that for any $\mathtt{p}(y_1, \ldots, y_n) \in D'$, there exists $\mathtt{p}(x_1, \ldots, x_n) \in D$ such that $\pi(x_i) = y_i$ for all $1 \leq i \leq n$. An *isomorphism* from D to D' is a bijective homomorphism π from D to D' for which π^{-1} is also a homomorphism.

Given a set of extensional predicates $\mathcal{P}_e$, a *(Boolean) query* $\mathtt{q}$ is a set of $\mathcal{P}_e$-databases that is closed under isomorphisms.[3] For $D \in \mathtt{q}$ we say D *belongs to* $\mathtt{q}$ or $\mathtt{q}$ *matches* D. A query $\mathtt{q}$ is said to be *preserved under homomorphisms* or *homomorphism-closed* if for all $D \in \mathtt{q}$, the existence of a homomorphism from D to D' implies $D' \in \mathtt{q}$.

A *semipositive Datalog program* is a set of first-order logic formulae, also called *rule*, of the shape

$$\forall \mathbf{x}\, \forall \mathbf{y}\, B[\mathbf{x}, \mathbf{y}] \rightarrow \mathtt{p}(\mathbf{y}),$$

where $\mathbf{x}$ and $\mathbf{y}$ are sequences of variables from V, and B is a conjunction of

- extensional atoms of the form $\mathtt{r}(\mathbf{z})$ with $\mathtt{r} \in \mathcal{P}_e$ and $\mathbf{z} \subseteq \mathbf{x} \cup \mathbf{y}$,
- negated extensional atoms of the form $\neg\mathtt{r}(\mathbf{z})$ with $\mathtt{r} \in \mathcal{P}_e$ and $\mathbf{z} \subseteq \mathbf{x} \cup \mathbf{y}$, and
- intensional atoms of the form $\mathtt{s}(\mathbf{z})$ with $\mathtt{s} \in \mathcal{P}_i$ and $\mathbf{z} \subseteq \mathbf{x} \cup \mathbf{y}$,

and where $\mathtt{p}(\mathbf{y})$ is an atom of an intensional predicate whose variables belong to $\mathbf{y}$. Note that $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ are sequences of variables from V; in particular, the semipositive Datalog programs considered by us do not contain constants. For brevity, the leading universal quantifiers are usually omitted. A *semipositive Datalog query* is a semipositive Datalog program containing a special nullary predicate $\mathtt{goal}$. If $\mathbb{P}$ is a semipositive Datalog query, the $\mathcal{P}_e$-database D with domain Δ belongs to $\mathbb{P}$ if and only if $\mathsf{comp}(D) \cup \mathbb{P} \models \mathtt{goal}$ according to first-order logic semantics, where $\mathsf{comp}(D) := D \cup \{\neg\mathtt{p}(\mathbf{a}) \mid \mathtt{p}(\mathbf{a}) \notin D, \mathtt{p} \in \mathcal{P}_e, \mathbf{a} \in \Delta^n\}$.

Semipositive Datalog programs (queries) without negated extensional atoms are called *Datalog programs (queries)*.

Given a $\mathcal{P}_e$-database D over Δ and a linear order $\leq$ over Δ, we define the $\mathcal{P}_e \cup \{\mathtt{initial}, \mathtt{final}, \mathtt{succ}\}$-database $D^{\leq}$ as D extended by the atoms

- $\mathtt{initial}(a)$ for the $\leq$-minimal element a of Δ,
- $\mathtt{final}(b)$ for the $\leq$-maximal element b of Δ,

[1] Since Datalog originated from databases, we employ database terminology in this paper. In KR or logical terms, D could be also seen as a finite interpretation over $\mathcal{P}_e$.

[2] We consider the definition from Chang and Keisler [Chang and Keisler, 1989].

[3] This definition reflects the common understanding of a query that it "[...] should be independent of the representation of the data in a data base and should treat the elements of the data base as uninterpreted objects" [Chandra and Harel, 1980]. This understanding also justifies why we do not distinguish the domain elements into constants and labeled nulls, as it sometimes done in the literature, and why we do not allow for constants in our query languages.

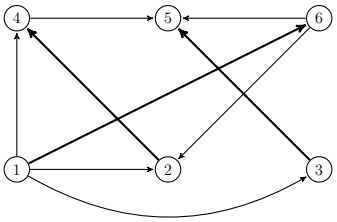

Figure 2: A graph G_1 and a perfect matching, in bold.

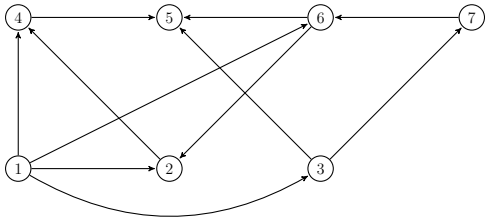

Figure 3: A graph G_2 without perfect matching.

- $\texttt{succ}(c, d)$ for any two $\leq$-consecutive elements c and d of Δ, that is, $c \leq d$ and for no $e \in \Delta \setminus \{c, d\}$, it holds that $c \leq e \leq d$.

An *order-invariant (semipositive) Datalog* query is a (semipositive) Datalog query $\mathbb{P}$ making use of `initial`, `final`, and `succ`) (besides predicates in $\mathcal{P}_e$ and $\mathcal{P}_i$) whose result (match or no match) is independent of the particular choice of the linear order $\leq$. We then let D belong to $\mathbb{P}$ iff $\texttt{comp}(D^{\leq}) \cup \mathbb{P} \models \texttt{goal}$ for some (or, equivalently, every) linear order $\leq$.

We now define the following classes of queries:

- Datalog$(\leq, \neg)$: the class of queries expressible by an order-invariant semipositive Datalog query,
- Datalog$(\leq)$: the class of queries expressible by an order-invariant Datalog query,
- Datalog$(\neg)$: the class of queries expressible by a semipositive Datalog query,
- Datalog: the class of queries expressible by a Datalog query, and
- HC: the class of homomorphism-closed queries.

3 Perfect Matchings and Razborov's Result

As usual, a *directed finite graph* (in the following just *graph*) is a pair $G = (V_G, E_G)$, where V_G is a finite set, called *vertices* and $E_G \subseteq V_G \times V_G$ is a binary relation on V_G, called *edges*. For any $(v, v') = e \in E_G$, the vertices v and v' are called the *ends* of e. Two edges are *adjacent* if they share an end. A *matching* for G is a set M of pairwise non-adjacent edges. A *perfect matching* for G is a matching M such that every vertex of V_G belongs to an edge of M. It is well-known (but not immediate to see) that the existence of a perfect matching in a graph can be checked in polynomial time [Edmonds, 1965].

We use two different encodings to represent graphs. Let $G = (V_G, E_G)$ be a graph, such that $V_G = \{1, \ldots, n\}$. The relational representation of G is a finite relational structure whose domain is V_G and whose unique relation is the binary relation `edge`. For each edge $(u, v) \in E_G$, the relation `edge` contains the pair (u, v). We may also represent the same graph by an n^2-tuple $(g_0, \ldots, g_{n^2-1}) \in \{0,1\}^{n^2}$ such that $g_{(i-1)*n+(j-1)} = 1$ if and only if $(i, j) \in E_G$.

A *k-ary Boolean function* (for $k \geq 1$) is a function $f : \{0,1\}^k \to \{0,1\}$. A *$k$-input Boolean circuit* $\mathcal{C}$ (for $k \geq 1$) is a directed acyclic graph with k distinguished nodes (labeled from 0 to $k-1$), called *sources (inputs)* which have no incoming edges, and with one distinguished node called *sink (output)* which has no outgoing edges. Every non-source node of $\mathcal{C}$ is called a *gate*; it is labeled with either one of $\wedge, \vee$, in which case is has two incoming edges[4], or with $\neg$, in which case there is one incoming edge. Given a vector $\mathbf{x} \in \{0,1\}^k$, the *value of source i* on $\mathbf{x}$ is the i^{th} bit of $\mathbf{x}$. The *value of a gate x* labeled by $\wedge$ (resp. $\vee$, resp. $\neg$) is the conjunction of the values of its incoming gates (resp. disjunction, resp. negation). The value of the sink on $\mathbf{x}$ is denoted by $\mathcal{C}(\mathbf{x})$. The number of nodes in $\mathcal{C}$ is its size, denoted by $|\mathcal{C}|$. $\mathcal{C}$ is said to compute a Boolean function f if for any tuple $\mathbf{x} \in \{0,1\}^k$, $\mathcal{C}(\mathbf{x}) = f(\mathbf{x})$. A Boolean circuit is *monotone* if no gate is labeled with $\neg$.

A family of circuits for the perfect matching problem is a sequence of circuits $(\mathcal{C}_i)$ where $\mathcal{C}_i$ has i^2 inputs, $0, \ldots, i^2-1$, such that the output of $\mathcal{C}_i$ is 1 if and only if its input is the representation of a graph with i vertices that contains a perfect matching, and 0 otherwise. Razborov showed the following.

Theorem 1 ([Razborov, 1985], Th. 3)**.** *There exists a constant $c > 0$ such that the size of the circuits in an arbitrary family of monotone circuits for the perfect matching function on bipartite graphs is greater than $n^{c \log n}$.*

We will use a weaker statement, following from this theorem: there can be no family of monotone circuits for the perfect matching function (for arbitrary graphs) of polynomial size. Such a family could be easily transformed into one violating Theorem 1 by removing nodes and edges.

4 A Homomorphism-Closed PTIME Query not Expressible in Datalog$(\leq)$

In this section, we present a homomorphism-closed, PTIME-computable query that is not expressible by a Datalog query on a linearly ordered database. The key idea is that the existence of a Datalog program (independent of the database) expressing this query implies the existence of a polynomial circuit for the perfect matching query, which would contradict Theorem 1. A similar argument has been used to show that some monotonic PTIME queries are not expressible by some variant of Datalog [Afrati, Cosmadakis, and Yannakakis, 1995].[5] The challenge here is to craft a suitable query which

[4] In fact, we will consider circuits with arbitrarily many ingoing edges for $\wedge$ and $\vee$ gates (straightforwardly generalizing the definition). Since we are only interested in size-polynomiality, this is not a problem as for every such circuit there is a circuit with gates $\wedge$ and $\vee$ having only two ingoing edges and of polynomial size with respect to the original circuit.

[5] More precisely, monotonic queries as defined in that work correspond to queries preserved under injective homomorphisms. This

checks for perfect matchings and is homomorphism-closed at the same time. Indeed, we cannot directly use "D encodes a graph containing a perfect matching" as our query, as this query would not be preserved under homomorphism, as can be seen in Fig. 3: there is a homomorphism from graph G_1 from Fig. 2 to G_2, but there is no perfect matching in G_2.

However, there is some reminiscence of this property: adding an edge to a graph containing a perfect matching results in a graph that also contains a perfect matching. The following proposition formalize this intuition.

Proposition 2. *Let G be a graph containing a perfect matching, H be a graph. If there exists a bijective homomorphism h from G to H, then H contains a perfect matching.*

Proof. Let M be a matching of G. Let us consider $h(M)$. Since no edges of M share an end and h is injective, no edges of $h(M)$ share an edge. Moreover, since every vertex of G appears once in M and h is surjective, every vertex of H appears once in $h(M)$. □

The idea to arrive at a homomorphism-closed query is to add additional information to the database about which vertices are considered when looking for a perfect matching.

Thus, beyond a binary predicate `edge` that describes the edges of the graph, we use the additional predicates `first` (unary), `last` (unary), and `next` (binary) which, intuitively, we will use to merge elements of the database attempting to obtain a linear order so we have a better control over the graph vertices when looking for a perfect matching. We call Σ the set of these four predicates.

The idea to construct a query that is homomorphism-closed but still can be used to solve the perfect matching problem is as follows: for the class of databases where `first`, `last`, and `next` happen to encode a linear order (let us call them *well-behaved databases*) on the database's domain, the query should match exactly if `edge` encodes a graph containing a perfect matching. Note that within the class of well-behaved databases, every homomorphism is necessarily bijective, thus Proposition 2 ensures homomorphism-closedness within this class. Then, in order to define query membership for all other databases and still ensure homomorphism-closedness we proceed as follows:

- the query does not match any database where there is no `next`-connected component containing both a `first`-individual and a `last`-individual (as it is clear that no database with `first`, `last`, and `next` forming a linear order – and hence no well-behaved database – can be homomorphically mapped into such a database, so it is safe to let these databases not be matched by the query)
- the query matches all databases which – even after removal of all elements not contained in any `first`- and `last`-containing `next`-connected component – cannot be homomorphically mapped into any structure where `first`, `last`, and `next` constitute a linear order (obviously, these databases can never send but at best receive homomorphisms from well-behaved databases, so it is safe to let these databases be matched by the query)
- any non-well-behaved database D not falling in any of the two previous categories can be turned into a well-behaved database D' in a deterministic fashion (by iteratively merging domain elements and finally removing unconnected domain elements) with the following two properties: *(i)* any well-behaved database receiving a homomorphism from D also receives a homomorphism from D'; *(ii)* any well-behaved database having a homomorphism into D also has a homomorphism into D'. Therefore, we need to have D satisfy the query if and only if D' does, i.e., iff `edge` in D' encodes a graph that contains a perfect matching.

In the following, we formally elaborate the above intuition.

Definition 3 (Enlisted element)**.** *Let D be a Σ-database over the domain Δ. An element $d \in \Delta$ is called* enlisted *if it is contained in a sequence $d_1, \ldots, d_n$ of domain elements with $\texttt{first}(d_1) \in D$ and $\texttt{last}(d_n) \in D$ and $\texttt{next}(d_i, d_{i+1}) \in D$ for every $i \in \{1, \ldots, n-1\}$.*

Fig. 4 illustrates the notion of enlisted elements.

Attempting to merge elements such that `first`, `last`, and `next` describe a linear order on enlisted elements, we next consider an equivalence relation on the database elements.

Definition 4 (Congruence)**.** *Given a Σ-database D over Δ, let $\cong$ be the smallest equivalence relation on the database elements satisfying the following:*

- *for all $d_1, d_2 \in \Delta$ with $\texttt{first}(d_1) \in D$ and $\texttt{first}(d_2) \in D$, $d_1 \cong d_2$ holds*
- *for all $d_1, d_2 \in \Delta$ with $\texttt{last}(d_1) \in D$ and $\texttt{last}(d_2) \in D$, $d_1 \cong d_2$ holds*
- *for all $d_1, d_2, d_1', d_2' \in \Delta$ with $d_1 \cong d_2$ and $\texttt{next}(d_1, d_1') \in D$ as well as $\texttt{next}(d_2, d_2') \in D$, $d_1' \cong d_2'$ holds*
- *for all $d_1, d_2, d_1', d_2' \in \Delta$ with $d_1' \cong d_2'$ and $\texttt{next}(d_1, d_1') \in D$ as well as $\texttt{next}(d_2, d_2') \in D$, $d_1 \cong d_2$ holds.*

The $\cong$-equivalence class of d is denoted by $[d]_\cong$.

In the next step, we define a new database by merging elements of the database which are $\cong$-equivalent.

Definition 5 (Compression of D)**.** *The* compression *of a Σ-database D is the Σ-database D' over $\Delta' := \{[d]_\cong \mid d \text{ enlisted}\}$ such that:*

- $\texttt{first}([d]_\cong)$ *iff there exists $d' \in \Delta$ such that $d' \cong d$ and $\texttt{first}(d')$*
- $\texttt{last}([d]_\cong)$ *iff there exists $d' \in \Delta$ such that $d' \cong d$ and $\texttt{last}(d')$*
- $\texttt{next}([d_1]_\cong, [d_2]_\cong)$ *iff there exists $d_1', d_2' \in \Delta$ such that $d_1' \cong d_1, d_2' \cong d_2$ and $\texttt{next}(d_1', d_2')$*
- $\texttt{edge}([d_1]_\cong, [d_2]_\cong)$ *iff there exists $d_1', d_2' \in \Delta$ such that $d_1' \cong d_1, d_2' \cong d_2$ and $\texttt{edge}(d_1', d_2')$.*

defines a larger class of queries than the homomorphism-closed queries considered here. In particular the monotonic queries described there are not homomorphism-closed.

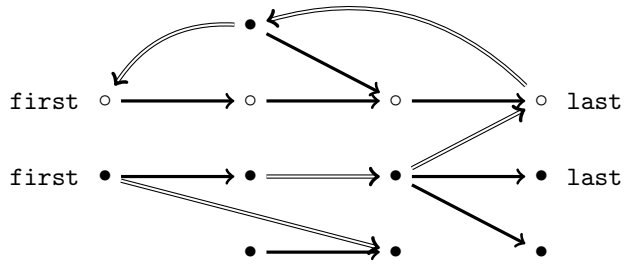

Figure 4: Example of a Σ-database. Simple edges represent `next`, double edges represent `edge`. Enlisted elements are drawn with empty circles.

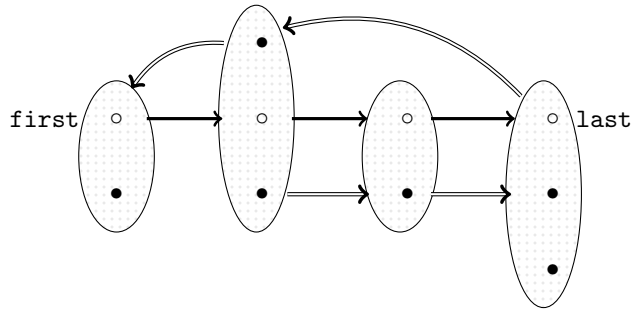

Figure 5: The compression of D from Fig. 4, ellipses representing equivalence classes.

Fig. 5 displays the database obtained by compressing the database from Fig. 4.

We now have the required tools to define our query. Intuitively, there are two reasons why the query might match a database after the compression operation: (1) contradictory encoding of the linear order or (2) correct encoding and existence of a perfect matching.

Definition 6 (The query $\mathtt{q}$)**.** *We define* $\mathtt{q}$ *such that* $D \in \mathtt{q}$ *if and only if one of the following holds:*

- *there are* $d, d' \in \Delta'$ *such that* $\{\mathtt{next}(d, d'), \mathtt{first}(d')\} \subseteq D'$ *or*
- *there are* $d, d' \in \Delta'$ *such that* $\{\mathtt{last}(d), \mathtt{next}(d, d')\} \subseteq D'$ *or*
- Δ *contains enlisted elements and the graph encoded by* D' *(via the* `edge` *relation) contains a perfect matching.*

We can now establish, that the defined query has the claimed properties. PTime computability follows from PTIME preprocessing and PTIME checking for a perfect matching.

Proposition 7. $\mathtt{q}$ *is computable in polynomial time.*

Preservation under homomorphisms is obtained by construction, as explained above.

Proposition 8. $\mathtt{q}$ *is preserved under homomorphisms.*

Finally, the general argument of $\mathtt{q}$ non-expressibility in order-invariant Datalog is an indirect one: Suppose there were an order-invariant Datalog program $\mathbb{P}$ computing $\mathtt{q}$. Given this program and a natural number n, we show how to construct a polynomial-size monotone Boolean circuit over n^2 input variables recognizing the existence of a perfect matching in a graph with n vertices. However, this contradicts Theorem 1, hence the initial assumption must be wrong.

Theorem 9. *The query* $\mathtt{q}$ *defined above is not expressible in order-invariant Datalog.*

5 Expressivities – Known and Easy Cases

In this section, we provide the justification of the complete picture about the relationships of the considered query classes depicted in Fig. 1, for which the query established in the preceding section constitutes the final building block. We first note that by syntactic inclusion of the query languages the following semantic inclusions hold: $\mathrm{Datalog} \subseteq \mathrm{Datalog}(\leq)$, $\mathrm{Datalog} \subseteq \mathrm{Datalog}(\neg)$, $\mathrm{Datalog}(\leq) \subseteq \mathrm{Datalog}(\leq, \neg)$, and $\mathrm{Datalog}(\neg) \subseteq \mathrm{Datalog}(\leq, \neg)$. Furthermore, it is well-known that Datalog queries are homomorphism-closed: $\mathrm{Datalog} \subseteq \mathrm{HC}$. The following theorem, which ensures $\mathrm{Datalog}(\neg) \cap \mathrm{HC} \subseteq \mathrm{Datalog}$ has been established by Feder and Vardi [Feder and Vardi, 2003].

Theorem 10. *Every* $\mathrm{Datalog}(\neg)$ *query that is preserved under homomorphisms is expressible in* $\mathrm{Datalog}$*.*

In order to further clarify the relationship between the classes of queries considered here, the following two propositions – which establish model-theoretic properties of $\mathrm{Datalog}(\leq)$ and $\mathrm{Datalog}(\neg)$ – come handy:

Proposition 11. *Every* $\mathrm{Datalog}(\leq)$ *query is preserved under bijective homomorphisms.*

Proposition 12. *Every* $\mathrm{Datalog}(\neg)$ *query is preserved under strong homomorphisms.*

We will use these propositions below to show non-expressibility of certain queries. Furthermore, they can be employed to obtain the following result:

Theorem 13. *Every query that is expressible both in* $\mathrm{Datalog}(\leq)$ *and* $\mathrm{Datalog}(\neg)$ *can be expressed in* $\mathrm{Datalog}$*.*

To ensure that no further inclusion relationships than the ones noted above (and their consequences) hold, we provide examples of queries for all the remaining admissible combinations of memberships and non-memberships. These combinations correspond to the regions in the Venn diagram depicted in Fig. 1, labeled by letters. If not specified otherwise, we assume just one binary database predicate `edge`, allowing us to interpret databases as directed graphs.

(a) Regarding homomorphism-closed queries not contained in any of the Datalog classes, consider the class of existential rules queries, which has been shown to precisely capture all homomorphism-closed recursively enumerable queries [Rudolph and Thomazo, 2015]. In particular this class contains all homomorphism-closed EXPTIME-hard queries, which cannot be expressed in $\mathrm{Datalog}(\leq, \neg)$, since PTIME $\subsetneq$ EXPTIME.

(b) A query that checks if a graph contains at least one edge is homomorphism-closed and expressible in all the Datalog variants.

$$
\begin{aligned}
\texttt{edge}(x,y) &\rightarrow \texttt{path}(x,y) && (1)\\
\texttt{edge}(x,y) \wedge \texttt{path}(y,z) &\rightarrow \texttt{path}(x,z) && (2)\\
\texttt{path}(x,x) &\rightarrow \texttt{goal} && (3)\\
&\rightarrow \texttt{eqdist}(x,x,y,y) && (4)\\
\texttt{eqdist}(x,x',y,y') \wedge \texttt{succ}(x',x'') \wedge \wedge \texttt{succ}(y',y'') &\rightarrow \texttt{eqdist}(x,x'',y,y'') && (5)\\
\texttt{initial}(x) &\rightarrow \texttt{sqr}(x,x) && (6)\\
\texttt{initial}(x) \wedge \texttt{succ}(x,y) &\rightarrow \texttt{sqr}(y,y) && (7)\\
\texttt{succ}(x,y) \wedge \texttt{succ}(y,z) \wedge \texttt{sqr}(x,x') \wedge \texttt{sqr}(y,y') & \\
\wedge\, \texttt{eqdist}(x',y',y',z') \wedge \texttt{succ}(z',z'') \wedge \texttt{succ}(z'',z''') &\rightarrow \texttt{sqr}(z,z''') && (8)\\
\texttt{initial}(x) \wedge \texttt{succ}(x,y) &\rightarrow \texttt{dualpower}(x,y) && (9)\\
\texttt{initial}(x) \wedge \texttt{dualpower}(y,z) \wedge \texttt{succ}(y,y') & \\
\wedge\, \texttt{eqdist}(x,z,z,w) &\rightarrow \texttt{dualpower}(y',w) && (10)\\
\texttt{square}(x,y) \wedge \texttt{dualpower}(y,z) \wedge \texttt{dualpower}(z,w) &\rightarrow \texttt{goodlength}(w) && (11)\\
\texttt{source}(x) \wedge \texttt{initial}(y) &\rightarrow \texttt{distance}(x,y) && (12)\\
\texttt{distance}(x,y) \wedge \texttt{edge}(x,x') \wedge \texttt{succ}(y,y') &\rightarrow \texttt{distance}(x',y') && (13)\\
\texttt{distance}(x,y) \wedge \texttt{final}(x) \wedge \texttt{goodlength}(y) &\rightarrow \texttt{goal} && (14)
\end{aligned}
$$

Figure 6: $\mathrm{Datalog}(\leq)$ version of the query described by Dawar and Kreutzer

(c) A query checking if a graph is not symmetric (*i.e.*, there exists an edge (a, b) but no edge (b, a)) is expressible in $\mathrm{Datalog}(\neg)$ (and hence in $\mathrm{Datalog}(\leq, \neg)$) via the one-rule program $\texttt{edge}(x,y) \wedge \neg\texttt{edge}(y,x) \rightarrow \texttt{goal}$, but not in $\mathrm{Datalog}(\leq)$ (nor Datalog) since it is not closed under bijective homomorphisms. Consequently it is also not homomorphism-closed.

(d) A query checking if a graph contains an even number of vertices is expressible in $\mathrm{Datalog}(\leq)$ (and hence in $\mathrm{Datalog}(\leq, \neg)$) as follows:

$$
\begin{aligned}
\texttt{initial}(x) &\rightarrow \texttt{odd}(x)\\
\texttt{odd}(x) \wedge \texttt{succ}(x,y) &\rightarrow \texttt{even}(y)\\
\texttt{even}(x) \wedge \texttt{succ}(x,y) &\rightarrow \texttt{odd}(y)\\
\texttt{final}(x) \wedge \texttt{even}(x) &\rightarrow \texttt{goal}
\end{aligned}
$$

However the query cannot be expressed in $\mathrm{Datalog}(\neg)$ (nor Datalog) since it is not closed under strong homomorphisms. It is thus also not homomorphism-closed.

(e) A query that checks for acyclicity of a graph is not homomorphism-closed, while it is in $\mathrm{Datalog}(\leq, \neg)$ (as it clearly is in PTIME) but in none of its subclasses (the query is neither closed under bijective homomorphisms nor under strong homomorphisms).

(f) For a query that is homomorphism-closed and expressible in $\mathrm{Datalog}(\leq)$ (and hence in $\mathrm{Datalog}(\leq, \neg)$) but not in Datalog (and therefore, via Theorem 10 also not in $\mathrm{Datalog}(\neg)$) we refer to Dawar and Kreutzer (2008), where the following query is defined: given unary predicates `source` and `target` and a binary predicate `edge`, the query checks if the relation `edge` contains a cycle or there is a natural number n such that there is an `edge`-path of length $2^{(2^{n^2})}$ from some $s \in \Delta$ with $\texttt{source}(s)$ to some $t \in \Delta$ with $\texttt{target}(t)$. It was shown that this query is homomorphism-closed and in PTIME, but not in Datalog (this was shown via some pumping argument). It remains to be shown that this query is in $\mathrm{Datalog}(\leq)$. To see this, consider the $\mathrm{Datalog}(\leq)$ query displayed in Fig. 6.

Rules (1)–(3) ensure that the query matches whenever there is an `edge`-cycle. Now assume w.l.o.g. $\Delta = \{0, \ldots, m\}$ such that the the element i is the $(i+1)$th element in the linear order encoded by $\leq$. Then, rules (4) and (5) ensure that $\texttt{equidist}(i,j,k,\ell)$ is entailed whenever $j - i = \ell - k \geq 0$. Consequently rules (6)-(8) ensure that $\texttt{sqr}(i,j)$ is entailed exactly if $j = i^2$, while rules (9) and (10) make sure that $\texttt{dualpower}(i,j)$ is entailed whenever $j = 2^i$. Then, Rule (11) delivers $\texttt{goodlength}(i)$ as a consequence whenever $i = 2^{(2^{n^2})}$ for some n. Rules (12) and (13) make sure that $\texttt{distance}(i,j)$ is a consequence, if the domain element i can be reached by an `edge`-path of length j from some individual a with $\texttt{source}(a) \in D$, consequently Rule (14) makes the query match if there is an `edge`-path of length $2^{(2^{n^2})}$ from some a with $\texttt{source}(a) \in D$ to some b with $\texttt{target}(b) \in D$.

(g) Finally, a query that is homomorphism-closed and expressible in $\mathrm{Datalog}(\leq, \neg)$ but in none of the others has been exposed in the preceding section.

6 Conclusion and Future Work

We have compared the expressive power of four variants of Datalog, where input negation and a linear order may or may not be used. For completing the picture, we had to show that there exists a PTIME homomorphism-closed query that is not expressible in order-invariant Datalog without input negation (while it is by a classical result when allowing for input negation). This is in strong contrast with the classical result by Feder and Vardi [Feder and Vardi, 2003] showing that in the absence of a linear order, input negation is dispensable for expressing homomorphism-closed queries. We are somewhat baffled by this result: in order to express queries which satisfy the strongest notion of monotonicity, one cannot dispense with negation, the epitome of non-monotonicity.

In future work, we plan to characterize further variants of Datalog by model-theoretic and computational properties.

7 Acknowledgement

M. Thomazo acknowledges support from the ANR project ContentCheck, ANR-15-CE23-0025-01.

References

[Abiteboul, Hull, and Vianu, 1994] Abiteboul, S.; Hull, R.; and Vianu, V. 1994. *Foundations of Databases*. Addison Wesley.

[Afrati, Cosmadakis, and Yannakakis, 1995] Afrati, F. N.; Cosmadakis, S. S.; and Yannakakis, M. 1995. On datalog vs. polynomial time. *J. Comput. Syst. Sci.* 51(2):177–196.

[Bourhis, Krötzsch, and Rudolph, 2014] Bourhis, P.; Krötzsch, M.; and Rudolph, S. 2014. How to best nest regular path queries. In Bienvenu, M.; Ortiz, M.; Rosati, R.; and Simkus, M., eds., *Informal Proc. 27th International Workshop on Description Logics (DL2014)*, volume 1193 of *CEUR Workshop Proceedings*, 404–415. CEUR-WS.org.

[Bourhis, Krötzsch, and Rudolph, 2015] Bourhis, P.; Krötzsch, M.; and Rudolph, S. 2015. Reasonable highly expressive query languages. In Yang, Q., and Wooldridge, M., eds., *Proc. 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, 2826–2832. AAAI Press.

[Chandra and Harel, 1980] Chandra, A. K., and Harel, D. 1980. Computable queries for relational data bases. *J. Comput. Syst. Sci.* 21(2):156–178.

[Chang and Keisler, 1989] Chang, C. C., and Keisler, H. J. 1989. *Model Theory*. North Holland, Amsterdam, third edition.

[Cuenca Grau et al., 2013] Cuenca Grau, B.; Motik, B.; Stoilos, G.; and Horrocks, I. 2013. Computing Datalog rewritings beyond Horn ontologies. In Rossi, F., ed., *Proc. 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*. IJCAI/AAAI.

[Dawar and Kreutzer, 2008] Dawar, A., and Kreutzer, S. 2008. On datalog vs. LFP. In Aceto, L.; Damgård, I.; Goldberg, L. A.; Halldúrsson, M. M.; Ingólfsdóttir, A.; and Walukiewicz, I., eds., *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*, 160–171.

[Edmonds, 1965] Edmonds, J. 1965. Paths, trees and flowers. *Canad. J. Math.* 17:449–467.

[Feder and Vardi, 2003] Feder, T., and Vardi, M. Y. 2003. Homomorphism closed vs. existential positive. In *Proc. 18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, 311–320.

[Gottlob and Schwentick, 2012] Gottlob, G., and Schwentick, T. 2012. Rewriting ontological queries into small nonrecursive Datalog programs. In Brewka, G.; Eiter, T.; and McIlraith, S. A., eds., *Proc. 13th International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*. AAAI Press.

[Gottlob, Rudolph, and Simkus, 2014] Gottlob, G.; Rudolph, S.; and Simkus, M. 2014. Expressiveness of guarded existential rule languages. In Hull, R., and Grohe, M., eds., *Proc. 33rd Symposium on Principles of Database Systems (PODS'14)*, 27–38.

[Immerman, 1999] Immerman, N. 1999. *Descriptive complexity*. Graduate texts in computer science. Springer.

[Kaminski, Nenov, and Cuenca Grau, 2014] Kaminski, M.; Nenov, Y.; and Cuenca Grau, B. 2014. Datalog rewritability of disjunctive Datalog programs and its applications to ontology reasoning. In Brodley, C. E., and Stone, P., eds., *Proc. 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, 1077–1083. AAAI Press.

[Ortiz, Rudolph, and Simkus, 2010] Ortiz, M.; Rudolph, S.; and Simkus, M. 2010. Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In Lin, F.; Sattler, U.; and Truszczynski, M., eds., *Proc. 12th International Conference on Principles of Knowledge Representation and Reasoning (KR'10)*. AAAI Press.

[Razborov, 1985] Razborov, A. 1985. Lower bounds for the monotone complexity of some boolean functions. *Dokl. Akad. Nauk SSSR* 281(4):798–801.

[Rudolph and Thomazo, 2015] Rudolph, S., and Thomazo, M. 2015. Characterization of the expressivity of existential rule queries. In Yang, Q., and Wooldridge, M., eds., *Proc. 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press.

8 Appendix

Proof of Proposition 7

Proof. Let us first notice that the compression D' of D can be computed in polynomial time. A naive algorithm is to merge, as long as possible, any pair of elements that fulfills one of the conditions of Definition 4. One can perform at most $|D|$ merges, and each step can be done in polynomial time. Elements that are not merged with an enlisted elements can then be removed to compute D'. Given D', checking the first two conditions of Definition 6 can be done in quadratic time. The fourth condition can also be checked in polynomial time [Edmonds, 1965]. □

Proof of Proposition 8

Proof. Let $D_1 \in \mathfrak{q}$ (with domain Δ_1), and let D_2 (with domain Δ_2) such that there is a homomorphism h from D_1 to D_2. Let us first notice that if d is enlisted, then $h(d)$ is enlisted as well. Indeed, if the sequence $d_1, \ldots, d_n$ witnesses that d is enlisted, then $h(d_1), \ldots, h(d_n)$ witnesses that $h(d)$ is enlisted. Moreover, if $d \cong d'$, then $h(d) \cong h(d')$. We thus define $h([d]_{\cong})$ as $[h(d)]_{\cong}$, which defines h on Δ_1'. Let us now consider the three possibilities that prove that $D_1 \in \mathfrak{q}$:

- If there exists $d, d' \in \Delta_1'$ such that $\{\mathtt{next}(d, d'), \mathtt{first}(d')\} \subseteq D_1'$, then $h(d), h(d')$ are such that $\{\mathtt{next}(h(d), h(d')), \mathtt{first}(h(d'))\} \subseteq D_2'$
- If there are $d, d' \in \Delta_1'$ such that $\{\mathtt{last}(d), \mathtt{next}(d, d')\} \subseteq D_1'$, then $h(d), h(d')$ are such that $\{\mathtt{last}(h(d)), \mathtt{next}(h(d), h(d'))\} \subseteq D_2'$

- Let us now assume that none of the first two cases hold. If Δ_1 contains enlisted elements, then Δ_2 as well. We distinguish two cases:
  - If there exist enlisted elements $d, d' \in \Delta_1$ such that $d \not\cong d'$ and $h(d) \cong h(d')$, then there exists a sequence $d_1, \ldots, d_k$ of elements of Δ'_2 such that for any i with $1 \leq i < k$, $\texttt{next}(d_i, d_{i+1}) \in D'_2$, with $h(d) = d_1 = d_k$. This in turn implies that there exist $e, e' \in \Delta'_2$ such that $\{\texttt{next}(e, e'), \texttt{first}(e')\} \subseteq D'_2$.
  - If for all elements $d, d' \in \Delta_1$ with $d \not\cong d'$ it holds that $h(d) \not\cong h(d')$, and that D_2 does not fulfill any of the first two cases, then there is a bijection between Δ'_1 and Δ'_2. h (lifted to sets) is thus a bijective homomorphism from D'_1 to D'_2, and by Proposition 2, D'_2 contains a perfect matching.

In the three cases, we proved that $D_2 \in \mathfrak{q}$, which proves that $\mathfrak{q}$ is closed under homomorphisms. □

Proof of Theorem 9

For technical purposes, we recall that the semantics of Datalog can be captured through *derivation trees* (also known as *proof trees*). A derivation tree of an atom a from a database D and a program $\mathbb{P}$ is a tree where:

1. each vertex of the tree is labeled by an atom
2. each leaf is labeled by an atom of D
3. the root is labeled by a
4. for each internal vertex, the exists an instantiation r_i of a rule of $\mathbb{P}$ such that the vertex is labeled by the head of r_i and its children are labeled by the body atoms of r_i.

An atom a is entailed by $\mathbb{P}$ and D if and only if there exists a derivation tree of a from D and $\mathbb{P}$. The reader is referred to [Abiteboul, Hull, and Vianu, 1994] for more details.

In the following we provide the stepwise construction of the circuit and prove the required properties of the applied transformations.

We start by fixing the database domain to $\{1, \ldots, n\}$ which allows us to ground $\mathbb{P}$, resulting in a program $\mathbb{P}_{\text{ground}}$ only containing ground atoms[6](*i.e.*, atoms having only domain elements as arguments) which is equivalent to $\mathbb{P}$.

Definition 14 (From $\mathbb{P}$ to $\mathbb{P}_{\text{ground}}$)**.** *$\mathbb{P}_{\text{ground}}$ is built from $\mathbb{P}$ and constants from $\{1, \ldots, n\}$ as follows: for every rule r of $\mathbb{P}$, we add to $\mathbb{P}_{\text{ground}}$ all the rules that can be obtained by grounding r,* i.e.*, by applying a mapping from the set of variables of r to $\{1, \ldots, n\}$.*

Proposition 15. *$\mathbb{P}$ and $\mathbb{P}_{\text{ground}}$ are equivalent on any database whose domain is $\{1, \ldots, n\}$. Moreover, $\mathbb{P}_{\text{ground}}$ is of size polynomial with respect to n.*

Proof. Let k be the maximum number of variables appearing in a single rule of $\mathbb{P}$. Each rule can thus be grounded in at most n^k different rules, which is polynomial in $\mathbb{P}$. As any rule of $\mathbb{P}_{\text{ground}}$ is an instantiation of a rule of $\mathbb{P}$, it is clear that for any D, $D, \mathbb{P}_{\text{ground}} \models p(i_1, \ldots, i_k)$ implies that $D, \mathbb{P} \models p(i_1, \ldots, i_k)$. For the converse, let π be a homomorphism from the body of a rule r to a database whose domain is $\{1, \ldots, n\}$. Its image is isomorphic to the body of a grounded rule, which concludes the proof. □

Next we prune $\mathbb{P}_{\text{ground}}$ by removing rules which we know can never "fire" under the assumption that both predicate sets `first`, `last`, `next` and `initial`, `final`, `succ` follow the "natural" linear order on $\{1, \ldots, n\}$.

Definition 16 (From $\mathbb{P}_{\text{ground}}$ to $\mathbb{P}_{\text{cleared}}$)**.** *We build $\mathbb{P}_{\text{cleared}}$ from $\mathbb{P}_{\text{ground}}$ by removing from $\mathbb{P}_{\text{ground}}$ all rules whose body contains one of the following atoms:*

- $\texttt{next}(i, j)$ *or* $\texttt{succ}(i, j)$ *with* $j \neq i + 1$,
- $\texttt{first}(i)$ *or* $\texttt{initial}(i)$ *with* $i \neq 1$,
- $\texttt{last}(i)$ *or* $\texttt{final}(i)$ *with* $i \neq n$.

Proposition 17. *$\mathbb{P}_{\text{cleared}}$ and $\mathbb{P}_{\text{ground}}$ are equivalent on any database whose domain is $\{1, \ldots, n\}$ and for which* `first`, `next`, `last` *describe the linear order $1, \ldots, n$. Moreover, the size of $\mathbb{P}_{\text{cleared}}$ is smaller than the size of $\mathbb{P}_{\text{ground}}$.*

Proof. $\mathbb{P}_{\text{cleared}}$ being a subset of $\mathbb{P}_{\text{ground}}$, it is clear that it is smaller, and that any consequence of D and $\mathbb{P}_{\text{cleared}}$ is a consequence of D and $\mathbb{P}_{\text{ground}}$, for any database D. Let us show that the converse hold as well for the above specified databases. In particular, let r be a rule such that $\texttt{next}(i, j)$ belongs to the body of r, with $j \neq i + 1$. By assumption on D, `next` describes the linear order $1, \ldots, n$. Thus, $\texttt{next}(i, j)$ does not belong to D. Moreover, since `next` is an extensional predicate, no atom of this predicate may be generated through a rule application. This means that r can not be applied, and we can thus remove it from $\mathbb{P}_{\text{ground}}$ without changing the semantics. The same reasoning applies for rules containing $\texttt{first}(i)$ with $i \neq 1$ or $\texttt{last}(i)$ with $i \neq n$.

Let us now consider the case of rules containing $\texttt{succ}(i, j)$ with $j \neq i + 1$. By assumption, the query computed by $\mathbb{P}_{\text{ground}}$ is independent on the linear order encoded by `succ`, `initial`, `final`. We can freely choose which order to use, and consider the same linear order $1, \ldots, n$. The same reasoning as above implies that we can remove any rule whose body contain an atom of the prescribed form. □

Next, we remove atoms from rule bodies that we know are always satisfied under the assumptions made above.

Definition 18 (From $\mathbb{P}_{\text{cleared}}$ to $\mathbb{P}_{\text{reduct}}$)**.** *We build $\mathbb{P}_{\text{reduct}}$ from $\mathbb{P}_{\text{cleared}}$ by removing all atoms of the predicates* `first`, `last`, `next`, `initial`, `final`, `succ` *from the rule bodies of $\mathbb{P}_{\text{cleared}}$.*

Proposition 19. *$\mathbb{P}_{\text{reduct}}$ and $\mathbb{P}_{\text{cleared}}$ are equivalent on any database whose domain is $\{1, \ldots, n\}$ and for which* `first`, `next`, `last` *describe the linear order $1, \ldots, n$. Moreover, the size of $\mathbb{P}_{\text{reduct}}$ is smaller than the size of $\mathbb{P}_{\text{cleared}}$.*

Proof. As before, it is clear that $\mathbb{P}_{\text{reduct}}$ is smaller than $\mathbb{P}_{\text{cleared}}$. It is also clear that any atom entailed by D and $\mathbb{P}_{\text{reduct}}$ is also entailed by $\mathbb{P}_{\text{cleared}}$.

[6]Note that "partially grounded programs" do not fall into the original query category, but their semantics is straightforward and we only need them as a "vehicle" for our proofs.

Let r be a rule of $\mathbb{P}_{\text{cleared}}$ containing an atom of predicate $\texttt{first}$. By construction of $\mathbb{P}_{\text{cleared}}$, this atom must be $\texttt{first}(1)$. By assumption, $\texttt{first}(1) \in D$. Thus, this atom can be removed from r. The same reasoning holds for atoms of predicate $\texttt{last}, \texttt{next}$. For atoms of predicate $\texttt{initial}, \texttt{final}, \texttt{succ}$, we again use the fact that the computed query is independent on the chosen linear order, and we choose $1, \ldots, n$, then apply the same reasoning. □

$\mathbb{P}_{\text{reduct}}$ is then a Datalog program whose extensional atoms are restricted to the binary predicate $\texttt{edge}$. The next step is an equivalent transformation which ensures that no cyclic relationships between ground atoms can hold.

Definition 20 (From $\mathbb{P}_{\text{reduct}}$ to $\mathbb{P}_{\text{acyclic}}$)**.** *We create $\mathbb{P}_{\text{acyclic}}$ from $\mathbb{P}_{\text{reduct}}$ as follows. Let gr be the number of ground atoms in $\mathbb{P}_{\text{reduct}}$.*

- *for every ground atom $\texttt{p}(i_1, \ldots, i_k)$ of $\mathbb{P}_{\text{reduct}}$, we introduce a rule $\texttt{p}(i_1, \ldots, i_k) \rightarrow \texttt{p}_0(i_1, \ldots, i_k)$*
- *for every rule r of $\mathbb{P}_{\text{reduct}}$ and every natural number ℓ such that $0 \leq \ell < gr$, let $\mathbb{P}_{\text{acyclic}}$ contain a rule obtained from r by replacing every intensional body atom $\texttt{p}(i_1, \ldots, i_k)$ by $\texttt{p}_\ell(i_1, \ldots, i_k)$ and the head atom $\texttt{p}'(j_1, \ldots, j_{k'})$ is replaced by $\texttt{p}'_{\ell+1}(j_1, \ldots, j_{k'})$*
- *for every ground atom $\texttt{p}(i_1, \ldots, i_k)$ of $\mathbb{P}_{\text{reduct}}$ and every natural number ℓ such that $0 \leq \ell < gr$ let $\mathbb{P}_{\text{acyclic}}$ contain the rule $\texttt{p}_\ell(i_1, \ldots, i_k) \rightarrow \texttt{p}_{\ell+1}(i_1, \ldots, i_k)$.*
- *we introduce the rule $\texttt{goal}_{gr} \rightarrow \texttt{goal}$*

Proposition 21. *$\mathbb{P}_{\text{reduct}}$ and $\mathbb{P}_{\text{acyclic}}$ are equivalent on any database. Moreover, the size of $\mathbb{P}_{\text{acyclic}}$ is polynomial in the size of $\mathbb{P}_{\text{reduct}}$.*

Proof. $\Leftarrow$ Let D be a database such that $D, \mathbb{P}_{\text{acyclic}} \models$ **goal**. We show by induction on j that if $\texttt{p}_j(i_1, \ldots, i_k)$ is derived from D and $\mathbb{P}_{\text{acyclic}}$, then $\texttt{p}(i_1, \ldots, i_k)$ is derived from D and $\mathbb{P}_{\text{reduct}}$.

- For $j = 0$, there is only one way to derive $p_0(i_1, \ldots, i_k)$, which is to use a rule of the shape $p(i_1, \ldots, i_k) \rightarrow p_0(i_1, \ldots, i_k)$. Since no rule of $\mathbb{P}_{\text{acyclic}}$ can derive an atom of predicate p, this implies that $p(i_1, \ldots, i_k)$ belongs to D, and thus $D, \mathbb{P}_{\text{reduct}} \models p(i_1, \ldots, i_k)$.
- Let us assume the result to be true for any $j' \leq j$, and let us show it for $j+1$. An atom of the shape $p_{j+1}(i_1, \ldots, i_k)$ can be derived in two ways. The first case is by a rule of the shape $p_j(i_1, \ldots, i_k) \rightarrow p_{j+1}(i_1, \ldots, i_k)$. By induction assumption, $p(i_1, \ldots, i_k)$ is entailed by D and $\mathbb{P}_{\text{reduct}}$, which shows the claim. The other possibility is by an application of a rule of the shape $\bigwedge p^k_j(i^k_1, \ldots, i^k_{k_k}) \rightarrow p_{j+1}(i_1, \ldots, i_k)$. This rule belongs to $\mathbb{P}_{\text{acyclic}}$ because $\bigwedge p^k(i^k_1, \ldots, i^k_{k_k}) \rightarrow p(i_1, \ldots, i_k)$ belongs to $\mathbb{P}_{\text{reduct}}$. Since the body of the first rule is entailed by D and $\mathbb{P}_{\text{acyclic}}$, and by induction hypothesis, the body of the second rule is entailed by D and $\mathbb{P}_{\text{reduct}}$, which implies that $p(i_1, \ldots, i_k)$ is entailed by D and $\mathbb{P}_{\text{reduct}}$, which conclude the proof.

$\Rightarrow$ We show by induction on the depth of the derivation tree that for any atom $\texttt{p}(i_1, \ldots, i_k)$ entailed by D and $\mathbb{P}_{\text{reduct}}$, $\texttt{p}_j(i_1, \ldots, i_k)$ is entailed by D and $\mathbb{P}_{\text{acyclic}}$. This implies in particular that $\texttt{p}_{gr}(i_1, \ldots, i_k)$ is entailed as well.

- If the depth is 0, then $p(i_1, \ldots, i_k) \in D$, and thus $p_0(i_1, \ldots, i_k)$ is entailed by D and $\mathbb{P}_{\text{acyclic}}$.
- Otherwise, let us notice that for any atom entailed by D and $\mathbb{P}_{\text{reduct}}$, there is a derivation tree of depth at most gr. Indeed, there is a derivation tree for any atom entailed. If there is path of length greater than gr from the root to a leave of the tree, there exists an atom that appears twice in this path. The tree rooted in the highest occurrence of this atom can be replaced by the tree rooted in the lowest occurrence, whose depth is strictly smaller. Let thus be an atom derived by a tree of depth j. The last rule application, of say rule r, use atoms that can be derived from $\mathbb{P}_{\text{reduct}}$ through derivation whose trees are of depth at most $j-1$. By induction assumption, similar atoms (indiced by $j-1$) can be derived from $\mathbb{P}_{\text{acyclic}}$. By applying r whose body atoms have been indiced by $j-1$, one derives $p_j(i_1, \ldots, i_k)$, which shows the claim.

□

The last step consists in transforming $\mathbb{P}_{\text{acyclic}}$ in a Boolean circuit.

Definition 22 (From $\mathbb{P}_{\text{acyclic}}$ to $\mathbf{C}_{P_{\text{acyclic}}}$)**.** *From $\mathbb{P}_{\text{acyclic}}$, we build a Boolean circuit as follows:*

- *for every ground atom $\texttt{p}(i_1, \ldots, i_k)$, $\mathbf{C}_{\mathbb{P}_{\text{acyclic}}}$ contains a node, denoted by $[\texttt{p}(i_1, \ldots, i_k)]$*
- *for every rule r of $\mathbb{P}_{\text{acyclic}}$, $\mathbf{C}_{\mathbb{P}_{\text{acyclic}}}$ contains a node, denoted by $[r]$.*
- *the nodes of the shape $[\texttt{edge}(i, j)]$ are the sources*
- *for every intensional ground atom $\texttt{p}(i_1, \ldots, i_k)$, the corresponding node $[\texttt{p}(i_1, \ldots, i_k)]$ is an OR-gate having as parents all nodes of the shape $[r]$ where r is a rule of $\mathbb{P}_{\text{acyclic}}$ having $\texttt{p}(i_1, \ldots, i_k)$ as head.*
- *for every rule r, the corresponding node is an AND-gate having as inputs all the nodes $[\texttt{p}(i_1, \ldots, i_k)]$ where $\texttt{p}(i_1, \ldots, i_k)$ belongs to the body of r.*
- *the sink of $\mathbf{C}_{\mathbb{P}_{\text{acyclic}}}$ is $[\texttt{goal}_{gr}]$.*

The size of $\mathbf{C}_{\mathbb{P}_{\text{acyclic}}}$ is polynomial in n: there are at most $pred \cdot n^{maxarity}$ ground atoms in $\mathbb{P}_{\text{reduct}}$, where $pred$ is the number of predicates used and $maxarity$ is the maximal predicate arity. Moreover, $\mathbb{P}_{\text{reduct}}$ contains at most $rul \cdot n^{maxvar}$ rules where rul is the number of rules in $\mathbb{P}$ and $maxvar$ is the maximal number of distinct variables per rule. Consequently, the number of rules in $\mathbb{P}_{\text{acyclic}}$ (and hence also the number of rule bodies) is upper bounded by

$$(pred \cdot n^{maxarity}) \cdot (rul \cdot n^{maxvar} + 1),$$

whereas the number of ground predicates in $\mathbb{P}_{\text{acyclic}}$ is at most $(pred \cdot n^{maxarity})^2$. Consequently we have found a polynom in n as upper bound for the number of nodes in $\mathbf{C}_{\mathbb{P}_{\text{acyclic}}}$.

Finally we verify that the constructed $\mathbf{C}_{\mathbb{P}_{\text{acyclic}}}$ is indeed a Boolean circuit computing the perfect matching function for graphs with n vertices, which concludes our overall argument.

Proposition 23. *Let $\mathcal{G} = (g_1, \ldots, g_{n^2})$ be a tuple representing an oriented graph. $\mathbf{C}_{\mathbb{P}_{\text{acyclic}}}$ outputs 1 on $\mathcal{G}$ if and only if G represents a graph containing a perfect matching.*

Proof. Let $\mathcal{G} = (g_1, \ldots, g_{n^2})$ be a tuple representing an oriented graph. Let D be the relational representation of the graph represented by $\mathcal{G}$. We prove that the value of the node $[p(i_1, \ldots, i_k)]$ on the input $\mathcal{G}$ is 1 if and only if $D, \mathbb{P}_{\text{acyclic}} \models p(i_1, \ldots, i_k)$.

- $\Rightarrow$ We prove this by induction on the depth of $[p(i_1, \ldots, i_k)]$, where the depth of a gate is defined as the length of the longest path from a source to this node.
  - If $[p(i_1, \ldots, i_k)]$ is a source, then $p(i_1, \ldots, i_k)$ is of the shape $\texttt{edge}(i_1, i_2)$. Since edge is an extensional predicate, $D, \mathbb{P}_{\text{acyclic}} \models \texttt{edge}(i_1, i_2)$ if and only if $D \models \texttt{edge}(i_1, i_2)$. Because D and $\mathcal{G}$ represent the same graph, $[\texttt{edge}(i_1, i_2)]$ equals to 1 implies that $D \models \texttt{edge}(i_1, i_2)$ holds, which concludes the base case
  - Let us assume that the result holds for all gates of type $[p(i_1, \ldots, i_k)]$ up to depth j. We prove the result for nodes of depth $j + 1$. A gate of label $[p(i_1, \ldots, i_k)]$ is an OR-gate, whose input gates are of the form $[r]$, where the head of all such rule r is $[p(i_1, \ldots, i_k)]$. There is thus an input gate $[r^*]$ whose value is 1. The gate $[r^*]$ is an AND-gate, whose input gates are of depth strictly smaller than $j + 1$. The body of the rule r^* is thus entailed, by induction assumption, by D and $\mathbb{P}_{\text{acyclic}}$. Thus its head as well, which concludes the proof.
- $\Leftarrow$ We prove the result by induction on the depth of shortest derivation tree showing that $D, \mathbb{P}_{\text{acyclic}} \models p(i_1, \ldots, i_k)$.
  - If the derivation tree is of depth 0, then $p(i_1, \ldots, i_k)$ belongs to D, and by definition of the representations of a graph, $[p(i_1, \ldots, i_k)]$ is a source of $\mathbf{C}_{\mathbb{P}_{\text{acyclic}}}$.
  - If the result is true for any atom derived by a derivation tree of depth j, let us show that it holds as well for atoms derived by trees of depth $j + 1$. Let r^* be the last rule applied. r^* has as head $[p(i_1, \ldots, i_k)]$. By induction assumption, for each atom of its body, its associated gate has 1 as value on $\mathcal{G}$. The AND-gate associated with r^* has thus 1 as value as well. The OR-gate $[p(i_1, \ldots, i_k)]$, which has $[r^*]$ has input gate, has thus 1 as value as well on $\mathcal{G}$.

□

Proof of Proposition 11

Proof. Assume a $\text{Datalog}(\leq)$ query $\mathbb{P}$ matches some database D and let π be a bijective homomorphism into a database D'. W.l.o.g, we can assume that the domains of D and D' coincide, and therefore $D \subseteq D'$. Then, $\mathbb{P}$ matches D means that $D^{\leq}, \mathbb{P} \models \texttt{goal}$ for some $\leq$, which can be witnessed by a proof tree. However, (due to $D \subseteq D'$) the very same proof tree also witnesses $D'^{\leq} \cup \mathbb{P} \models \texttt{goal}$ from which follows that $\mathbb{P}$ matches D'. □

Proof of Proposition 12

Proof. Assume a $\text{Datalog}(\neg)$ query $\mathbb{P}$ matches some database D over the domain Δ and let π be a strong homomorphism into a database D' over the domain Δ'. Note that due to π being a strong homomorphism, the restriction of D' to $\pi(\Delta)$ is identical to $\pi(D)$ an therefore the restriction of $\texttt{comp}(D')$ to $\pi(\Delta)$ is identical to $\texttt{comp}(\pi(D))$. Consequently, the proof tree witnessing $\texttt{comp}(\pi(D)) \cup \mathbb{P} \models \texttt{goal}$ can be turned into a proof tree witnessing $\texttt{comp}(\pi(D')) \cup \mathbb{P} \models \texttt{goal}$ by replacing all domain elements $d \in \Delta$ by $\pi(d)$. □

Proof of Theorem 13

Proof. We will show that such a query must be closed under homomorphisms. Via expressibility in $\text{Datalog}(\neg)$ and Theorem 10, expressibility in Datalog is then an immediate consequence.

Let $Q \in \text{Datalog}(\leq) \cap \text{Datalog}(\neg)$ and let $D \in Q$. Let $\pi : \Delta \to \Delta'$ be a homomorphism between D and some database D'. Let now $\tilde{D}$ be the database over Δ' with $\tilde{D} = \{\texttt{p}(\pi(d_1), \ldots, \pi(d_n)) \mid \texttt{p}(d_1, \ldots, d_n) \in D\}$. Then π is a strong homomorphism from D to $\tilde{D}$, thus $\tilde{D} \in Q$ by Proposition 12. On the other hand, the identity function over Δ' is a bijective homomorphism from $\tilde{D}$ to D', therefore $D' \in Q$ by Proposition 11. □