

Tuple-Generating Dependencies Capture Complex Values

Extended Abstract

Maximilian Marx^{1,*}, Markus Krötzsch^{2,*}

¹Knowledge-Based Systems Group, TU Dresden, Dresden, Germany

Abstract

We review a recently introduced extension of Datalog that allows complex values constructed by nesting elements of the input database in sets and tuples. We study its complexity and show that it can be translated into existential rules for which the standard chase terminates on every input database. We identify a tractable fragment, for which membership is undecidable and propose decidable sufficient conditions for membership.

Keywords

tuple-generating dependencies, standard chase, universal termination, high data complexity

1. Introduction

We recently introduced Datalog^{CV} [1], an extension of Datalog with *complex values*, which are constructed by building sets and tuples from constants and other complex values. It formalises the positive fragment of a Datalog variant described by Abiteboul et al. [2], generalises Datalog(S) [3], and is highly expressive. Existential rules, another extension of Datalog, have similarly high expressivity [4]. Entailment over such rules is undecidable. Decidable fragments exist, e.g., rule sets with *terminating chase*, but chase termination is again undecidable. As Datalog^{CV} admits a translation into such rule sets, we propose Datalog^{CV} as a “frontend” language for obtaining expressive rule sets with terminating chases.

2. Preliminaries

We consider fixed, pairwise disjoint, and countably infinite sets \mathbf{C} (*constants*), \mathbf{P} (*predicate names*), \mathbf{V} (*variables*), and \mathbf{N} (*labelled nulls*). Each predicate name $p \in \mathbf{P}$ has *arity* $\text{ar}(p) \in \mathbb{N}_{\geq 0}$. An *atom* is of the form $p(t_1, \dots, t_{\text{ar}(p)})$, where $p \in \mathbf{P}$ and $t_1, \dots, t_{\text{ar}(p)} \in \mathbf{C} \cup \mathbf{V} \cup \mathbf{N}$ are *terms*. An atom is *ground* if it contains neither variables nor labelled nulls. An *existential*


Datalog 2.0 2022: 4th International Workshop on the Resurgence of Datalog in Academia and Industry, September 05, 2022, Genova - Nervi, Italy


*Corresponding author.

✉ maximilian.marx@tu-dresden.de (M. Marx); markus.kroetzsch@tu-dresden.de (M. Krötzsch)

🌐 <https://kbs.inf.tu-dresden.de/max> (M. Marx); <https://kbs.inf.tu-dresden.de/mak> (M. Krötzsch)

🆔 0000-0003-1479-0341 (M. Marx); 0000-0002-9172-2601 (M. Krötzsch)

 © 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

rule (also called Tuple-Generating Dependency) ρ is a formula of first-order logic of the form $\forall \mathbf{x}, \mathbf{y}. \varphi[\mathbf{x}, \mathbf{y}] \rightarrow \exists \mathbf{z}. \psi[\mathbf{y}, \mathbf{z}]$, where (i) \mathbf{x} , \mathbf{y} , and \mathbf{z} are mutually disjoint lists of variables, (ii) φ and ψ are conjunctions of null-free atoms, (iii) φ contains only variables from $\mathbf{x} \cup \mathbf{y}$, and (iv) ψ contains only variables from $\mathbf{y} \cup \mathbf{z}$. If \mathbf{z} is empty, then ρ is a *Datalog rule*. We call φ the *body* and ψ the *head* of ρ .

A *boolean conjunctive query* (BCQ) q is a first-order sentence of the form $\exists \mathbf{x}. \varphi[\mathbf{x}]$ with φ a conjunction of atoms. A *rule set* Σ is a finite set of existential rules. Σ is a *Datalog program* if each rule $\rho \in \Sigma$ is a Datalog rule. An *EDB schema* is a finite set $\mathbf{P}^{\text{EDB}} \subsetneq \mathbf{P}$ of predicate names; Σ is *compatible* with \mathbf{P}^{EDB} if no predicate name $p \in \mathbf{P}^{\text{EDB}}$ occurs in the head of a rule in Σ . A *database* D over \mathbf{P}^{EDB} is a finite set of ground atoms using only predicate names from \mathbf{P}^{EDB} . We always assume that D is over an EDB schema compatible with Σ . A *database instance* is a set of variable-free atoms. We consider the usual first-order semantics: a database instance \mathcal{I} is a *model* of D and Σ if $D \subseteq \mathcal{I}$ and $\mathcal{I} \models \Sigma$ as a first-order theory. Σ and D entail BCQ q (written $\Sigma, D \models q$) if $\mathcal{I} \models q$ for every model \mathcal{I} of Σ and D . A model \mathcal{I} of Σ and D is *universal* if $\mathcal{I} \models q$ iff $\Sigma, D \models q$ for all BCQ q . The *standard chase* is a well-known algorithm for computing (finite) universal models; in general, it need not terminate. It terminates *universally* for a rule set if it terminates on any database over a compatible EDB schema.

3. Datalog^{CV}

Before defining Datalog^{CV} formally, we can already gain some intuition for complex values and the additional expressive power they offer by looking at the following example.

Example 1 Consider a database encoding a directed graph using facts $\text{edge}(s, t)$ to denote edges from vertex s to vertex t . In our proposed formalism Datalog^{CV}, we can query for paths (represented as sets of edges, i.e., as sets of pairs of vertices) from x to y as follows:

$$\text{edge}(x, y) \rightarrow \text{path}(x, y, \{\langle x, y \rangle\}) \quad (1)$$

$$\text{path}(x, y, P) \wedge \text{edge}(y, z) \rightarrow \text{path}(x, z, P \cup \{\langle y, z \rangle\}) \quad (2)$$

Intuitively, rule (1) states that whenever there is an edge from x to y , there is also a path going along that edge. Rule (2) extends a path from x to y along an edge from y to z .

Consider a database with the following facts:

$$\begin{array}{lll} \text{edge}(a, b) & \text{edge}(b, c) & \text{edge}(a, c) \\ \text{edge}(a, d) & \text{edge}(d, c) & \text{edge}(d, e) \end{array}$$

Then we can derive the following three paths from a to c :

$$\text{path}(a, c, \{\langle a, c \rangle\}) \quad \text{path}(a, c, \{\langle a, b \rangle, \langle b, c \rangle\}) \quad \text{path}(a, c, \{\langle a, d \rangle, \langle d, c \rangle\})$$

3.1. Syntax of Datalog^{CV}

The set \mathcal{S} of *sorts* is defined inductively; it contains (i) the *domain sort* Δ , (ii) for all $\tau \in \mathcal{S}$, the *set sort* $\{\tau\}$, and (iii) for all $\ell \geq 2$ and $\tau_1, \dots, \tau_\ell \in \mathcal{S}$, the *tuple sort* $\langle \tau_1, \tau_2, \dots, \tau_\ell \rangle$. Every variable

$v \in \mathbf{V}$ has a sort $\text{sort}(v) = \tau$ such that the set $\mathbf{V}_\tau = \{v' \in \mathbf{V} \mid \text{sort}(v') = \tau\}$ is countably infinite. The sets \mathbf{T}_τ of *terms of sort* τ are defined as follows:

$$\mathbf{T}_\Delta ::= c \mid v \quad c \in \mathbf{C}, v \in \mathbf{V}_\Delta \quad (3)$$

$$\mathbf{T}_{\langle \tau_1, \dots, \tau_\ell \rangle} ::= \langle t_1, \dots, t_\ell \rangle \mid v \quad t_i \in \mathbf{T}_{\tau_i} \text{ for } 1 \leq i \leq \ell, v \in \mathbf{V}_{\langle \tau_1, \dots, \tau_\ell \rangle} \quad (4)$$

$$\mathbf{T}_{\{\tau\}} ::= \{t_1, \dots, t_n\} \quad t_i \in \mathbf{T}_\tau \text{ for } 1 \leq i \leq n \quad (5)$$

$$\mid (t \cap t') \mid (t \cup t') \mid v \quad t, t' \in \mathbf{T}_{\{\tau\}}, v \in \mathbf{V}_{\{\tau\}}$$

Terms are constants (of the domain sort), tuples (over the individual component sorts), set literals (over an element sort), intersections or unions of set terms (of a set sort), or variables (of any sort). We also write $\{\}$ as \emptyset , and we may emphasise the sort for overloaded symbols with a subscript, as in $\emptyset_{\{\tau\}}$. *Basic* terms do not contain \cup or \cap ; *ground* terms are variable-free.

A predicate $p \in \mathbf{P}$ has associated sort $\text{sort}(p)$. A *schema* $\mathbf{S} = \langle \mathbf{P}^{\text{EDB}}, \mathbf{P}^{\text{IDB}} \rangle$ is a partition of \mathbf{P} into *EDB predicates* \mathbf{P}^{EDB} and *IDB predicates* \mathbf{P}^{IDB} . An *atom* for predicate $p \in \mathbf{P}$ is an expression $p(t)$ where $t \in \mathbf{T}_{\text{sort}(p)}$. For tuple-sorted predicates, we write the usual $p(t_1, \dots, t_\ell)$ instead of $p(\langle t_1, \dots, t_\ell \rangle)$; for set terms, we may omit the outermost parentheses, writing, e.g., $t_1 \cup t_2$ for $(t_1 \cup t_2)$. A *fact* is an atom $p(t)$, where t contains only basic ground terms. A *rule* is a sentence of the form $\forall \mathbf{x}. \varphi[\mathbf{x}] \rightarrow \psi[\mathbf{x}]$, where *body* φ (possibly empty) and *head* ψ (non-empty) are conjunctions of atoms. Universal quantifiers are usually omitted. A *Datalog^{CV} program* for schema \mathbf{S} is a finite set of rules, where no rule head contains an EDB predicate. A *database* \mathbb{D} for schema \mathbf{S} is a finite set of facts using only EDB predicates. A *Datalog^{CV} BCQ* is a sentence of the form $\exists \mathbf{x}. \varphi[\mathbf{x}]$ with φ a conjunction of atoms containing only basic terms.

Some useful predicates and operations can be defined in *Datalog^{CV}*.

Example 2 The following rules express the powerset function – we interpret $\text{PS}_\sigma(S, P)$ as “ S (of sort $\sigma = \{\tau\}$) has powerset P (of sort $\{\sigma\} = \{\{\tau\}\})$ – and predicates $\subseteq_\sigma, \in_\sigma, \notin_\sigma, \neq_\tau$, and \subset_σ for a set sort σ and an arbitrary sort τ , which we write infix $t_1 \diamond t_2$ instead of $\diamond(t_1, t_2)$ for better readability.

$$\rightarrow \text{PSU}_\tau(x, \emptyset, \emptyset) \quad (6)$$

$$\text{PSU}_\tau(x, P, Q) \rightarrow \text{PSU}_\tau(x, P \cup \{S\}, Q \cup \{S \cup \{x\}\}) \quad (7)$$

$$\rightarrow \text{PS}_\sigma(\emptyset, \{\emptyset\}) \quad (8)$$

$$\text{PS}_\sigma(S, P) \wedge \text{PSU}_\tau(x, P, Q) \rightarrow \text{PS}_\sigma(S \cup \{x\}, P \cup Q) \quad (9)$$

$$\rightarrow S \subseteq_\sigma S \cup T \quad (10)$$

$$S \cup \{x\} \subseteq_\sigma S \rightarrow x \in_\sigma S \quad (11)$$

$$S \cap \{x\} \subseteq_\sigma \emptyset \rightarrow x \notin_\sigma S \quad (12)$$

$$\{x\} \cap \{y\} \subseteq_{\{\tau\}} \emptyset \rightarrow x \neq_\tau y \quad (13)$$

$$S \subseteq_\sigma T \wedge x \in_\sigma T \wedge x \notin_\sigma S \rightarrow S \subset_\sigma T \quad (14)$$

Rule (8) states that the powerset of \emptyset is $\{\emptyset\}$ and rule (9) states that, given the powerset P of S , the powerset of $S \cup \{x\}$ is obtained from P by adding $Q \cup \{x\}$ for every $Q \in P$, which is computed by rules (6) and (7). Intuitively, the auxiliary atom $\text{PSU}_\tau(t, P, Q)$ expresses that

$Q = \{S \cup \{t\} \mid S \in P\}$. On a database containing only the constant c , the following facts are entailed:

$$\begin{array}{lll} \text{PSU}_\tau(c, \emptyset, \emptyset) & \text{PSU}_\tau(c, \{\emptyset\}, \{\{c\}\}) & \text{PSU}_\tau(c, \{\{\emptyset\}\}, \{\emptyset_c\}) \\ \text{PS}_\tau(\emptyset, \{\emptyset\}) & \text{PS}_\tau(\{c\}, \{\emptyset, \{c\}\}) & \end{array}$$

Note that rules like (6) or (10) are “unsafe”, i.e., they use variables in the head that do not occur in the body. As the scope of variables is always restricted to a finite active domain, such rules can be made safe by axiomatising an active domain predicate.

We assume that these shortcuts are always available in $\text{Datalog}^{\text{CV}}$ programs, and that databases are *flat*, i.e., they contain only facts of the domain sort Δ or of some tuple sort $\langle \Delta, \dots, \Delta \rangle$ over domain elements; all other facts can be constructed using appropriate rules.

3.2. Semantics of $\text{Datalog}^{\text{CV}}$

Similar to Datalog, we base our semantics on an *Herbrand interpretation* with the set of constants syntactically occurring in a given program and database as domain, where we interpret constants as themselves (*unique name assumption*). For sorts and ground terms, we define the interpretation using a function $\text{eval}(\cdot)$. For sorts, let $\text{eval}(\Delta) := \mathbf{C}$, $\text{eval}(\langle \tau_1, \dots, \tau_\ell \rangle) := \text{eval}(\tau_1) \times \dots \times \text{eval}(\tau_\ell)$, and $\text{eval}(\{\tau\}) := \mathcal{P}(\text{eval}(\tau)) = \{T \mid T \subseteq \text{eval}(\tau)\}$. For ground terms t , we define $\text{eval}(t)$ as follows:

$$\text{eval}(c) := c \qquad c \in \mathbf{C} \qquad (15)$$

$$\text{eval}(\langle s_1, \dots, s_\ell \rangle) := \langle \text{eval}(s_1), \dots, \text{eval}(s_\ell) \rangle \qquad (16)$$

$$\text{eval}(\{t_1, \dots, t_n\}) := \{\text{eval}(t_1), \dots, \text{eval}(t_n)\} \qquad (17)$$

$$\text{eval}(t_1 \cap t_2) := \text{eval}(t_1) \cap \text{eval}(t_2) \qquad (18)$$

$$\text{eval}(t_1 \cup t_2) := \text{eval}(t_1) \cup \text{eval}(t_2) \qquad (19)$$

The *domain* $\text{dom}(\mathbb{P}, \mathbb{D})$ of a $\text{Datalog}^{\text{CV}}$ program \mathbb{P} and database \mathbb{D} is the set of all constants occurring in \mathbb{P} or \mathbb{D} . An *interpretation* for \mathbb{P} and \mathbb{D} maps every predicate p to a set $p^{\mathcal{I}} \subseteq \text{eval}(\text{sort}(p))$ containing only constants from $\text{dom}(\mathbb{P}, \mathbb{D})$. \mathcal{I} *satisfies* a ground atom $p(t)$, written $\mathcal{I} \models p(t)$, if $\text{eval}(t) \in p^{\mathcal{I}}$; it satisfies a conjunction φ of such atoms, written $\mathcal{I} \models \varphi$, if $\mathcal{I} \models \alpha$ for all $\alpha \in \varphi$; and it *satisfies* a variable-free rule $\varphi \rightarrow \psi$, written $\mathcal{I} \models \varphi \rightarrow \psi$, if $\mathcal{I} \not\models \varphi$ or $\mathcal{I} \models \psi$. \mathcal{I} *satisfies* \mathbb{D} if $\mathcal{I} \models \alpha$ for all $\alpha \in \mathbb{D}$. If \mathcal{I} satisfies X , we also call \mathcal{I} a *model* of X .

As usual, we obtain a notion of entailment from this model theory: a ground fact α is entailed by \mathbb{P} and \mathbb{D} , written $\mathbb{P}, \mathbb{D} \models \alpha$, if $\mathcal{I} \models \alpha$ for all models \mathcal{I} of \mathbb{P} and \mathbb{D} . We can reduce BCQ entailment to ground fact entailment by using BCQs as bodies of a rule with some ground head atom not derived by any other rule.

As for Datalog, entailment can be decided by considering a single *least model*, which can also be computed by a chase-like process.

3.3. Complexity of Datalog^{CV}

The complexity of reasoning for program \mathbb{P} depends on the nesting of sorts occurring in \mathbb{P} . The *set-height* of a sort is the maximal nesting level of sets within, e.g., $\text{s-height}(\Delta) = 0$ and $\text{s-height}(\{\{0\}\}) = 2$.

Theorem 1 *Let \mathbb{P} be a Datalog^{CV} program and \mathbb{D} be a database for schema \mathbf{S} , and let α be a ground fact. Deciding $\mathbb{P}, \mathbb{D} \models \alpha$ is $k\text{EXPTIME}$ -complete for data complexity and $(k + 2)\text{EXPTIME}$ -complete for combined complexity, where k is the largest set-height of a sort in \mathbf{S} .*

For hardness, we extend a linear order encoded in \mathbb{D} to k levels of nested powersets, obtaining a k -exponentially long Turing tape. For the additional exponent (over Datalog) in combined complexity, we first obtain a doubly-exponential chain by ordering nested tuples, which forms the new basis for the powerset construction.

3.4. Translating Datalog^{CV} into Existential Rules

Datalog^{CV} programs, databases, and BCQs admit a translation into sets of existential rules such that query entailment over the translation has the same answers and the same complexity. This requires additional rules and also additional facts for every fact $\alpha \in \mathbb{D}$:

Theorem 2 *Let \mathbb{P} be a Datalog^{CV} program and \mathbb{D} be a database for \mathbb{P} . Then there is a set $\text{tr}(\mathbb{P})$ of existential rules and a set of facts $\text{tr}(\mathbb{D})$ such that: (i) for every Datalog^{CV} BCQ q , there is $\text{tr}(q)$ with $\mathbb{P}, \mathbb{D} \models q$ iff $\text{tr}(\mathbb{P}), \text{tr}(\mathbb{D}) \models \text{tr}(q)$; (ii) $\text{tr}(\mathbb{P})$, $\text{tr}(\mathbb{D})$, and $\text{tr}(q)$ can be computed in logarithmic space; and (iii) every standard chase sequence for $\text{tr}(\mathbb{P})$ over $\text{tr}(\mathbb{D})$ terminates in a number of steps that is k -exponential in the size of \mathbb{D} and $(k + 2)$ -exponential in the size of \mathbb{P} .*

4. Tractable reasoning with Datalog^{CV}

We now identify a fragment of Datalog^{CV} for which reasoning is still data-tractable. As 0EXPTIME is PTIME , one such fragment is set-free Datalog^{CV}. By flattening nested tuples, set-free Datalog^{CV} programs can be translated into Datalog programs with exponentially larger arities. We can do this even in the presence of sets, as long as we can bound the cardinality of all sets that might be derived.

A program \mathbb{P} has *k-bounded cardinality* if, for any database \mathbb{D} for \mathbb{P} , all ground facts α with $\mathbb{P}, \mathbb{D} \models \alpha$ are of the form $\{t_1, \dots, t_n\}$ with $n \leq k$. It has *bounded cardinality* if it has k -bounded cardinality for some k . A k -bounded cardinality Datalog^{CV} program can be transformed into a set-free Datalog^{CV} program by replacing set sorts $\tau = \{\sigma\}$ with k -tuples $\langle \sigma, \dots, \sigma \rangle$ and filling up empty positions with placeholders \square_τ .

Theorem 3 *Entailment for bounded cardinality Datalog^{CV} programs is PTIME -complete for data complexity and 2EXPTIME -complete for combined complexity.*

While bounded cardinality is undecidable, we can check for k -bounded cardinality by considering a variant of the *critical instance* with $(k + 1)$ elements.

Theorem 4 *It is undecidable whether a program \mathbb{P} has bounded cardinality; given $k \geq 0$, deciding if \mathbb{P} has k -bounded cardinality is 2^{EXPTIME} -complete.*

Weak set acyclicity tracks the propagation of sets along positions of the program and ensures the absence of cycles involving set unions. The *cardinality constraints problem* is a system of inequalities estimating the minimal cardinality of sets. Both approaches lead to sufficient conditions for bounded cardinality:

Theorem 5 *Let \mathbb{P} be a $\text{Datalog}^{\text{CV}}$ program. Deciding if \mathbb{P} is weakly set-acyclic is NL-complete; checking if the cardinality constraints problem for \mathbb{P} has an optimal value is in PTIME. Both conditions imply bounded cardinality.*

5. Discussion

Many syntactic criteria guarantee universal termination of the chase, but all fragments identified so far exhibit PTIME data complexity. Yet rule sets with universally terminating chases can express arbitrarily complex queries [5, 4]. This makes $\text{Datalog}^{\text{CV}}$ the first *structured* approach for obtaining such rule sets, and it puts complex values within reach of reasoners such as VLog [6].

Acknowledgments

The authors thank Stephan Mennicke for feedback on an earlier draft of this paper. This work is partly supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in project number 389792660 (TRR 248, Center for Perspicuous Systems), by the Bundesministerium für Bildung und Forschung (BMBF, Federal Ministry of Education and Research) in the Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI), and by BMBF and DAAD (German Academic Exchange Service) in project 57616814 (SECAI, School of Embedded and Composite AI).

References

- [1] M. Marx, M. Krötzsch, Tuple-generating dependencies capture complex values, in: D. Olteanu, N. Vortmeier (Eds.), Proc. 25th Int. Conf. on Database Theory (ICDT'22), volume 220 of *LIPICs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 13:1–13:20.
- [2] S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison Wesley, 1994.
- [3] D. Carral, I. Dragoste, M. Krötzsch, C. Lewe, Chasing sets: How to use existential rules for expressive reasoning, in: S. Kraus (Ed.), Proc. 28th Int. Joint Conf. on Artificial Intelligence (IJCAI'19), ijcai.org, 2019, pp. 1624–1631.
- [4] C. Bourgaux, D. Carral, M. Krötzsch, S. Rudolph, M. Thomazo, Capturing homomorphism-closed decidable queries with existential rules, in: M. Bienvenu, G. Lakemeyer, E. Erdem (Eds.), Proc. 18th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'21), 2021, pp. 141–150.

- [5] M. Krötzsch, M. Marx, S. Rudolph, The power of the terminating chase, in: Proc. 22nd Int. Conf. on Database Theory (ICDT'19), volume 127 of *LIPICs*, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2019, pp. 3:1–3:17.
- [6] D. Carral, I. Dragoste, L. González, C. J. H. Jacobs, M. Krötzsch, J. Urbani, Vlog: A rule engine for knowledge graphs, in: C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. F. Cruz, A. Hogan, J. Song, M. Lefrançois, F. Gandon (Eds.), Proc. 18th Int. Semantic Web Conf. (ISWC'19), Part II, volume 11779 of *LNCS*, Springer, 2019, pp. 19–35.