Tuple-Generating Dependencies Capture Complex Values

Maximilian Marx, Markus Krötzsch

Knowledge-based systems group

TU Dresden

ICDT 2022

Full paper: `https://iccl.inf.tu-dresden.de/web/DatalogCV/en`

Datalog$^{\text{CV}}$: Datalog with Complex Values

Overview:

- Datalog$^{\text{CV}}$ extends Datalog with sorts for **tuples** and **sets**
- these sorts can be nested: e.g., $\{\langle\{0, 1\}, 1\rangle\}$
- sets support unions $S \cup T$ and intersections $S \cap T$
- like Datalog, entailment $\mathbb{P}, \mathbb{D} \models \alpha$ can be decided via **least model** of $\mathbb{P}$ and $\mathbb{D}$
- databases are **flat**, i.e., don't contain sets or tuples

Datalog$^{\mathrm{CV}}$: Datalog with Complex Values

Overview:

- Datalog$^{\mathsf{CV}}$ extends Datalog with sorts for **tuples** and **sets**
- these sorts can be nested: e.g., $\{\langle\{0,1\},1\rangle\}$
- sets support unions $S \cup T$ and intersections $S \cap T$
- like Datalog, entailment $\mathbb{P}, \mathbb{D} \models \alpha$ can be decided via **least model** of $\mathbb{P}$ and $\mathbb{D}$
- databases are **flat**, i.e., don't contain sets or tuples

Features:

- the usual set operations $x \in S, x \notin S, S \subseteq T, S \subset T,$ and $\boldsymbol{\mathcal{P}}(S)$ are definable
- inequality is axiomatisable for any sort

Datalog$^{\mathrm{CV}}$: Datalog with Complex Values

Overview:

- Datalog$^{\text{CV}}$ extends Datalog with sorts for **tuples** and **sets**
- these sorts can be nested: e.g., $\{\langle\{0,1\},1\rangle\}$
- sets support unions $S \cup T$ and intersections $S \cap T$
- like Datalog, entailment $\mathbb{P}, \mathbb{D} \models \alpha$ can be decided via **least model** of $\mathbb{P}$ and $\mathbb{D}$
- databases are **flat**, i.e., don't contain sets or tuples

Features:

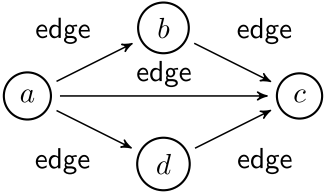
- the usual set operations $x \in S, x \notin S, S \subseteq T, S \subset T,$ and $\boldsymbol{\mathcal{P}}(S)$ are definable
- inequality is axiomatisable for any sort

Translation:

- Datalog$^{\text{CV}}$ programs admit a translation into sets of tuple-generating dependencies
- translated programs are all-instances all-strategies terminating under the standard chase
- use nulls to reify sets and tuples, represent sets as unions of singleton sets

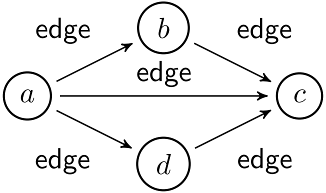
Datalog$^{\text{CV}}$ by Example

- Consider a database encoding a graph as edge(x, y):



Datalog$^{\mathrm{CV}}$ by Example

- Consider a database encoding a graph as edge(x, y):

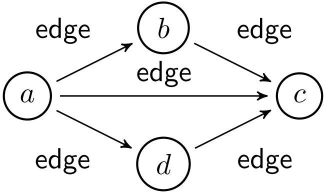


- Query for all paths from x to y:

$$\mathsf{edge}(x, y) \to \mathsf{path}(x, y, \{\langle x, y\rangle\})$$
$$\mathsf{path}(x, y, P) \wedge \mathsf{edge}(y, z) \to \mathsf{path}(x, z, P \cup \{\langle y, z\rangle\})$$

Datalog$^{\text{CV}}$ by Example

- Consider a database encoding a graph as edge(x, y):



- Query for all paths from x to y:

$$\text{edge}(x, y) \to \text{path}(x, y, \{\langle x, y\rangle\})$$
$$\text{path}(x, y, P) \land \text{edge}(y, z) \to \text{path}(x, z, P \cup \{\langle y, z\rangle\})$$

- Paths:

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

Complexity of Datalog$^{\mathrm{CV}}$

Schema Heights:

set-height maximal nesting depth of set sorts

tuple-height maximal nesting depth of tuple sorts

Theorem

Fact entailment for Datalog^{CV} *with set-height k is*

- kExpTime-*complete for data complexity,*
- $(k+2)$ExpTime-*complete for combined complexity, and*

Complexity of Datalog$^{\mathrm{CV}}$

Schema Heights:

set-height maximal nesting depth of set sorts

tuple-height maximal nesting depth of tuple sorts

Theorem

Fact entailment for Datalog^{CV} *with set-height k is*

- kEXPTIME-*complete for data complexity,*
- $(k+2)$EXPTIME-*complete for combined complexity, and*
- $(k+1)$EXPTIME-*complete for combined complexity if the tuple-height is bounded.*

Complexity of Datalog$^{\mathrm{CV}}$

Schema Heights:

set-height maximal nesting depth of set sorts

tuple-height maximal nesting depth of tuple sorts

Theorem

Fact entailment for Datalog^{CV} *with set-height k is*

- kExpTime*-complete for data complexity,*
- $(k+2)$ExpTime*-complete for combined complexity, and*
- $(k+1)$ExpTime*-complete for combined complexity if the tuple-height is bounded.*

- Lower bound by extending linear orders to powersets
- Upper bound from translation into TGDs

A Tractable Fragment

- Is there a non-trivial fragment with PTime data complexity?
- Observation: if all sets are bounded, we can replace sets by tuples
- $\rightsquigarrow$ bounded cardinality Datalog$^{\mathsf{CV}}$ is essentially high-arity Datalog

A Tractable Fragment

- Is there a non-trivial fragment with PTime data complexity?
- Observation: if all sets are bounded, we can replace sets by tuples
- $\rightsquigarrow$ bounded cardinality Datalog$^{\mathsf{CV}}$ is essentially high-arity Datalog

Definition

- ground fact α has k-bounded cardinality if all set terms occurring in α are of the form $\{t_1, \ldots, t_n\}$ with $n \leq k$
- program $\mathbb{P}$ has k-bounded cardinality if, for all databases $\mathbb{D}$, all ground facts α with $\mathbb{P}, \mathbb{D} \models \alpha$ have k-bounded cardinality
- $\mathbb{P}$ has bounded cardinality if it has k-bounded cardinality for some k

A Tractable Fragment

- Is there a non-trivial fragment with PTime data complexity?
- Observation: if all sets are bounded, we can replace sets by tuples
- $\rightsquigarrow$ bounded cardinality Datalog$^{\mathsf{CV}}$ is essentially high-arity Datalog

Definition

- ground fact α has k-bounded cardinality if all set terms occurring in α are of the form $\{t_1, \ldots, t_n\}$ with $n \leq k$
- program $\mathbb{P}$ has k-bounded cardinality if, for all databases $\mathbb{D}$, all ground facts α with $\mathbb{P}, \mathbb{D} \models \alpha$ have k-bounded cardinality
- $\mathbb{P}$ has bounded cardinality if it has k-bounded cardinality for some k

Example:

$$\mathsf{e}(x) \to \mathsf{s}(\{x\})$$
$$\mathsf{s}(X) \wedge \mathsf{s}(Y) \to \mathsf{p}(X \cup Y)$$

has 2-bounded cardinality.

A Tractable Fragment

- Is there a non-trivial fragment with PTime data complexity?
- Observation: if all sets are bounded, we can replace sets by tuples
- $\rightsquigarrow$ bounded cardinality Datalog$^{\mathsf{CV}}$ is essentially high-arity Datalog

Definition

- ground fact α has k-bounded cardinality if all set terms occurring in α are of the form $\{t_1, \ldots, t_n\}$ with $n \leq k$
- program $\mathbb{P}$ has k-bounded cardinality if, for all databases $\mathbb{D}$, all ground facts α with $\mathbb{P}, \mathbb{D} \models \alpha$ have k-bounded cardinality
- $\mathbb{P}$ has bounded cardinality if it has k-bounded cardinality for some k

Example:

$$\mathsf{e}(x) \to \mathsf{s}(\{x\})$$
$$\mathsf{s}(X) \wedge \mathsf{s}(Y) \to \mathsf{s}(X \cup Y)$$

does not have bounded cardinality.

Bounded Cardinality: Bad News & Good News

Bad News:

Theorem

It is undecidable whether a Datalog^{CV} *program has bounded cardinality.*

Bounded Cardinality: Bad News & Good News

Bad News:

Theorem

It is undecidable whether a Datalog^{CV} *program has bounded cardinality.*

Good News:

Theorem

For fixed k, deciding if a Datalog^{CV} *program $\mathbb{P}$ has k-bounded cardinality is* 2ExpTime*-complete (*ExpTime*-complete if set-height and tuple-height are bounded).*

Theorem

Bounded-cardinality Datalog^{CV} *is* PTime*-complete for data complexity and* 2ExpTime*-complete for combined complexity.*

A Syntactic Criterion: Weak Set-Acyclicity

- ‘large’ sets are constructed recursively as unions of smaller sets
- idea: track propagation of sets along positions of a program
- if there are no cycles involving unions, sets must be bounded
- like Weak Acyclicity, this can be checked using a propagation graph

A Syntactic Criterion: Weak Set-Acyclicity

- ‘large’ sets are constructed recursively as unions of smaller sets
- idea: track propagation of sets along positions of a program
- if there are no cycles involving unions, sets must be bounded
- like Weak Acyclicity, this can be checked using a propagation graph

Example

$$\mathsf{e}(x) \to \mathsf{s}(\{x\}) \qquad \mathsf{s}(X) \wedge \mathsf{s}(Y) \to \mathsf{p}(X \cup Y)$$

has WSA graph

$$\begin{array}{ccccc} & & s\cdot\epsilon & \Rightarrow & p\cdot\epsilon \\ e\cdot\epsilon & \longrightarrow & s\cdot 1 & \longrightarrow & p\cdot 1 \end{array}$$

no cycle along ‘special’ edges
$\rightsquigarrow$ WSA

A Syntactic Criterion: Weak Set-Acyclicity

- ‘large’ sets are constructed recursively as unions of smaller sets
- idea: track propagation of sets along positions of a program
- if there are no cycles involving unions, sets must be bounded
- like Weak Acyclicity, this can be checked using a propagation graph

Example

$$\mathsf{e}(x) \to \mathsf{s}(\{x\}) \qquad\qquad \mathsf{s}(X) \land \mathsf{s}(Y) \to \mathsf{s}(X \cup Y)$$

has WSA graph

$$\begin{tikzcd}
 & s \cdot \epsilon \arrow[loop right, Rightarrow] \\
e \cdot \epsilon \arrow[r] & s \cdot 1 \arrow[loop right]
\end{tikzcd}$$

cycle along ‘special’ edge $s \cdot \epsilon \Rightarrow s \cdot \epsilon$

$\rightsquigarrow$ not WSA

A Syntactic Criterion: Weak Set-Acyclicity

- 'large' sets are constructed recursively as unions of smaller sets
- idea: track propagation of sets along positions of a program
- if there are no cycles involving unions, sets must be bounded
- like Weak Acyclicity, this can be checked using a propagation graph

Example

$$\mathsf{e}(x) \to \mathsf{s}(\{x\}) \qquad \mathsf{s}(X) \wedge \mathsf{s}(Y) \wedge \mathsf{p}(P) \to \mathsf{s}(P \cap (X \cup Y)) \qquad \mathsf{s}(X) \wedge \mathsf{s}(Y) \to \mathsf{p}(X \cup Y)$$

has WSA graph

$$\begin{tikzcd} p \cdot \epsilon \arrow[r] & s \cdot \epsilon \arrow[loop right, Rightarrow] & \\ e \cdot \epsilon \arrow[r] & s \cdot 1 \arrow[loop right] & p \cdot 1 \arrow[l] \end{tikzcd}$$

cycle along 'special' edge $s \cdot \epsilon \Rightarrow s \cdot \epsilon$

$\rightsquigarrow$ not WSA (but 2-bounded cardinality)

Cardinality Constraints

- For every rule, bound the maximal cardinality of sets derived (w.r.t. sets in the match) $\rightsquigarrow$ system of inequalities
- If the sum of the bounds w.r.t. the inequalities has a minimum, the program has bounded cardinality

Cardinality Constraints

- For every rule, bound the maximal cardinality of sets derived (w.r.t. sets in the match) $\rightsquigarrow$ system of inequalities
- If the sum of the bounds w.r.t. the inequalities has a minimum, the program has bounded cardinality

Example

$$\mathsf{e}(x) \to \mathsf{s}(\{x\}) \qquad \mathsf{s}(X) \wedge \mathsf{s}(Y) \wedge \mathsf{p}(S) \to \mathsf{s}(S \cap (X \cup Y)) \qquad \mathsf{s}(X) \wedge \mathsf{s}(Y) \to \mathsf{p}(X \cup Y)$$

has cardinality constraints

$$x_{s\cdot\epsilon} \geq 0 \qquad x_{p\cdot\epsilon} \geq 0 \qquad x_{s\cdot\epsilon} \geq 1 \qquad x_{p\cdot\epsilon} \geq 2 \qquad x_{s\cdot\epsilon} \geq \min(x_{p\cdot\epsilon}, x_{s\cdot\epsilon} + x_{s\cdot\epsilon})$$

with optimal solution $x_{s\cdot\epsilon} = 2 = x_{p\cdot e}$ and optimal value $k = 4 \rightsquigarrow$ bounded cardinality

Conclusions & Outlook

Conclusions:

- Datalog$^{\mathsf{CV}}$ is a highly expressive query language
- TGD translation offers a path to reasoning with existing software
- Bounded cardinality Datalog$^{\mathsf{CV}}$ is a tractable fragment
- Weak Set-Acyclicity and Cardinality Constraints are sufficient conditions for bounded cardinality

Outlook:

- Can Datalog$^{\mathsf{CV}}$ express all decidable monotonic queries?
- Are there other tractable fragments?
- What happens if we add stratified negation?

Axiomatising Set Features

Powerset:

$$\begin{aligned}
&\rightarrow \mathsf{PSU}_\tau(x, \emptyset, \emptyset)\\
\mathsf{PSU}_\tau(x, P, Q) &\rightarrow \mathsf{PSU}_\tau(x, P \cup \{S\}, Q \cup \{S \cup \{x\}\})\\
&\rightarrow \mathsf{PS}_\sigma(\emptyset, \{\emptyset\})\\
\mathsf{PS}_\sigma(S, P) \wedge \mathsf{PSU}_\tau(x, P, Q) &\rightarrow \mathsf{PS}_\sigma(S \cup \{x\}, P \cup Q)
\end{aligned}$$

Subsets, Containment, Inequality:

$$\begin{aligned}
&\rightarrow S \subseteq_\sigma S \cup T\\
S \cup \{x\} \subseteq_\sigma S &\rightarrow x \in_\sigma S\\
S \cap \{x\} \subseteq_\sigma \emptyset &\rightarrow x \notin_\sigma S\\
\{x\} \cap \{y\} \subseteq_{\{\tau\}} \emptyset &\rightarrow x \neq_\tau y\\
S \subseteq_\sigma T \wedge x \in_\sigma T \wedge x \notin_\sigma S &\rightarrow S \subset_\sigma T
\end{aligned}$$

Enumerating Powersets

$$
\begin{aligned}
\mathsf{first}_\tau(x) \wedge \mathsf{last}_\tau(z) &\to \mathsf{first}_\sigma(\emptyset_\sigma) \wedge \\
&\quad\ \mathsf{mkStep}_\sigma(\emptyset_\sigma, x, x) \wedge \\
&\quad\ \mathsf{last}_\sigma(\{z\}) \\
\mathsf{mkStep}_\sigma(S, t, t) &\to \mathsf{next}_\sigma(S, S \cup \{t\}) \\
\mathsf{mkStep}_\sigma(S, t, x) \wedge \mathsf{last}_\tau(x) &\to \mathsf{step}_\sigma(S, t, x, S \cup \{x\}) \\
\mathsf{mkStep}_\sigma(S, t, x) \wedge \mathsf{next}_\tau(x, y) &\to \mathsf{mkStep}_\sigma(S \cup \{x\}, y, y) \wedge \\
&\quad\ \mathsf{mkStep}_\sigma(S, t, y) \\
\mathsf{mkStep}_\sigma(S, t, x) \wedge \mathsf{next}_\tau(x, y) \wedge & \\
\mathsf{step}_\sigma(S \cup \{x\}, y, y, X) \wedge \mathsf{step}_\sigma(S, t, y, Z) &\to \mathsf{next}_\sigma(X, S \cup \{y\}) \wedge \\
&\quad\ \mathsf{step}_\sigma(S, t, x, Z)
\end{aligned}
$$

Translating to TGDs

Tuple sorts:

$$\bigwedge_{i=1}^{\ell} \mathsf{sort}_{\tau_i}(x_i) \to \exists z.\mathsf{tuple}_{\pi}(z, x_1, \ldots, x_\ell) \wedge \mathsf{sort}_{\pi}(z)$$

Set sorts:

$$\begin{aligned}
&\to \exists V.\mathsf{empty}_{\sigma}(V) \wedge \mathsf{sort}_{\sigma}(V) \wedge \mathsf{done}_{\sigma}(V)\\
\mathsf{done}_{\sigma}(V) \wedge \mathsf{sort}_{\tau}(x) &\to \exists W.\mathsf{SU}_{\sigma}(x, V, W) \wedge \mathsf{sort}_{\sigma}(W)\\
&\wedge \mathsf{todo}_{\sigma}(W, W)\\
\mathsf{todo}_{\sigma}(V, W) \wedge \mathsf{SU}_{\sigma}(x, U, V) &\to \mathsf{SU}_{\sigma}(x, W, W) \wedge \mathsf{todo}_{\sigma}(U, W)\\
\mathsf{todo}_{\sigma}(V, W) \wedge \mathsf{empty}_{\sigma}(V) &\to \mathsf{done}_{\sigma}(W)
\end{aligned}$$

Translating to TGDs II

Subsets, Unions, Not-in:

$$\begin{aligned}
\mathsf{sort}_\sigma(V) \wedge \mathsf{sort}_\sigma(W) &\to \mathsf{ckSub}_\sigma(V, V, W)\\
\mathsf{ckSub}_\sigma(U, V, W) \wedge \mathsf{SU}_\sigma(x, U', U) \wedge \mathsf{SU}_\sigma(x, V, V) &\to \mathsf{ckSub}_\sigma(U', V, W)\\
\mathsf{ckSub}_\sigma(U, V, W) \wedge \mathsf{empty}_\sigma(U) &\to \mathsf{subset}_\sigma(V, W)\\
\mathsf{subset}_\sigma(V, W) \wedge \mathsf{subset}_\sigma(W, V) &\to \mathsf{eq}_\sigma(V, W)\\
\mathsf{empty}_\sigma(V) \wedge \mathsf{sort}_\sigma(W) &\to \mathsf{U}_\sigma(V, W, W)\\
\mathsf{SU}_\sigma(x, W, W') \wedge \mathsf{U}_\sigma(V, W', U) \wedge \mathsf{SU}_\sigma(x, V, V') &\to \mathsf{U}_\sigma(V', W, U)\\
\mathsf{sort}_\sigma(V) \wedge \mathsf{sort}_\tau(x) &\to \mathsf{ckNIn}_\sigma(V, x, V)\\
\mathsf{ckNIn}_\sigma(U, x, V) \wedge \mathsf{SU}_\sigma(y, U', U) \wedge \mathsf{NEq}_\tau(x, y) &\to \mathsf{ckNIn}_\sigma(U', x, V)\\
\mathsf{ckNIn}_\sigma(U, x, V) \wedge \mathsf{empty}_\sigma(U) &\to \mathsf{NIn}_\sigma(x, V)
\end{aligned}$$

Translating to TGDs III

Tuples, Inequality, Intersections:

$$
\begin{aligned}
\mathsf{tuple}_\pi(z, x_1, \ldots, x_\ell) \wedge \mathsf{tuple}_\pi(z', y_1, \ldots, y_\ell) \wedge \mathsf{NEq}_{\tau_i}(x_i, y_i)& \\
\rightarrow \mathsf{NEq}_\pi(z, z')& \\
\mathsf{SU}_\sigma(x, V, V) \wedge \mathsf{NIn}_\sigma(x, W) &\rightarrow \mathsf{NEq}_\sigma(V, W) \wedge \mathsf{NEq}_\sigma(W, V) \\
\mathsf{empty}_\sigma(V) \wedge \mathsf{sort}_\sigma(W) &\rightarrow \mathsf{I}_\sigma(V, W, V) \\
\mathsf{I}_\sigma(U, V, W) \wedge \mathsf{SU}_\sigma(x, U, U') \wedge \mathsf{NIn}_\sigma(x, V) &\rightarrow \mathsf{I}_\sigma(U', V, W) \\
\mathsf{I}_\sigma(U, V, W) \wedge \mathsf{SU}_\sigma(x, U, U') \wedge \mathsf{SU}_\sigma(x, V, V')& \\
\wedge \mathsf{SU}_\sigma(x, W, W') &\rightarrow \mathsf{I}_\sigma(U', V', W')
\end{aligned}
$$

Linear Datalog$^{\text{CV}}$ is still intractable

Linear Datalog$^{\text{CV}}$:

- rules $\varphi \to \psi$ where both φ and ψ are single-atom
- we need to allow non-flat databases

Theorem

*Linear Datalog^{CV} with set-height $k > 0$ is $(k-1)$*ExpTime*-hard (data complexity) and $(k+1)$*ExpTime*-hard (combined complexity; k*ExpTime*-hard for bounded tuple-height).*

Idea:

- encode each predicate p as a position of sets of ar(p)-tuples in a single facts predicate
- a rule $\rho = \varphi \to \psi$ turns into

$$\text{facts}(y_1 \cup \mathrm{ts}_\varphi(p_1), y_2 \cup \mathrm{ts}_\varphi(p_2), \dots) \to \text{facts}(y_1 \cup \mathrm{ts}_\rho(p_1), y_2 \cup \mathrm{ts}_\rho(p_2), \dots)$$

with $\mathrm{ts}_\Phi(p)$ the set of terms t for which $p(t)$ occurs in Φ