Liu HC, Liu J. On the expressive power of logics on constraint databases with complex objects. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 34(4): 795–817 July 2019. DOI 10.1007/s11390-019-1943-7

# On the Expressive Power of Logics on Constraint Databases with Complex Objects

Hong-Cheu Liu and Jixue Liu

*School of Information Technology and Mathematical Sciences, University of South Australia, Adelaide 5095, Australia*

E-mail: honcheu.liu@gmail.com; jixue.liu@unisa.edu.au

Received September 1, 2018; revised April 23, 2019.

**Abstract** We extend the constraint data model to allow complex objects and study the expressive power of various query languages over this sort of constraint databases. The tools we use come in the form of collapse results which are well established in the context of first-order logic. We show that the natural-active collapse with a condition and the active-generic collapse carry over to the second-order logic for structures with o-minimality property and any signature in the complex value relations. The expressiveness results for more powerful logics including monadic second-order logic, monadic second-order logic with fix-point operators, and fragments of second-order logic are investigated in the paper. We discuss the data complexity for second-order logics over constraint databases. The main results are that the complexity upper bounds for three theories, $\mathcal{MSO}$ + Lin, $\mathcal{MSO}$ + Poly, and Inflationary $\text{Datalog}^{\text{cv},\neg}_{\text{act}}(SC, \mathfrak{M})$ without powerset operator are $\cup_i \sum_i^{NC^1}$, $\mathbf{NCH} = \bigcup_i \sum_i^{NC}$, and $\text{AC}^0$/poly, respectively. We also consider the problem of query closure property in the context of embedded finite models and constraint databases with complex objects and the issue of how to determine safe constraint queries.

**Keywords** constraint database, monadic second-order logic, natural-active collapse, Ramsey property, o-minimality structure

## 1 Introduction

Database systems are being widely used to support emerging applications such as engineering design, image or voice data management, spatial and spatial-temporal information systems, and bioinformatics applications. With the coming of the big data era, many advanced applications involve voluminous data, which may sometimes be convenient to be considered "infinite" data. The collected spatial and temporal data may be complex and, in most case, contains much redundant information. It needs to be simplified for database representation and the constraint data model is regarded as an appropriate approach. Constraint databases were designed to represent compact and well re-presentable databases as it involves mathematical equations and inequalities in a way that extend relational databases while preserving the simplicity of main functions of querying of stored infinite data[1]. To allow applications to process possibly infinite data, constraint databases have been proposed, which use a finite representation of an infinite dataset and allow users to express queries on such a representation as if the entire infinite set was stored. In particular, spatial and/or temporal databases often contain infinite datasets such as the set of real numbers in an interval. These datasets can be represented as a finite set of logical constraints, where the actual dataset is the set of data points which satisfy the set of constraints. For example, the constraint $x+2y = 0$ represents the line which is a set of points $\{(2a, -a) \mid a \in \mathbb{R}\}$ on the plane.

In many applications, spatial or temporal data is often more intuitively described as sets of constraints, and this may be the reason why constraint databases are emerging as a unifying paradigm for the representation and manipulation of spatial or temporal data. The constraint database model, introduced by Kanellakis *et al.* in their seminal paper[2], is a powerful generalization of the relational data model which uses logical constraints as a finite representation of possibly infinite sets of data points. The essential idea of the constraint

Regular Paper

This work was supported in part by the Taiwan Science Council, China, under Grant No. NSC90-2218-E-126-001.

data model is to generalize the notion of a tuple, based on the observation that a tuple $(a, b, c)$ in a relation $R$ with schema $(x, y, z)$ can be represented as a logical conjunction: $(x = a) \wedge (y = b) \wedge (z = c)$. Similarly, if $R$ contains more than one tuple as shown in Table 1, then $R$ can be represented by the following formula in first order logic.

$$(x = a_1 \wedge y = b_1 \wedge z = c_1) \vee (x = a_2 \wedge y = b_2 \wedge z = c_2).$$

**Table 1.** Relation $R$

| $x$ | $y$ | $z$ |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_2$ | $b_2$ | $c_2$ |

Applying the relational query $\exists x R(x, y, z)$ to the above instance, the result yields $(y = b_1 \wedge z = c_1) \vee (y = b_2 \wedge z = c_2)$. The constraint data model then generalizes the notion of a tuple by defining a generalized tuple in $n$ variables $x_1, ..., x_n$ to be a conjunctive formula over $x_1, ..., x_n$, where each literal in the formula can be an arbitrary inequality. For example, $x \leqslant y \wedge x \leqslant 0$ defines a binary generalized tuple over variables $x, y$. A generalized relation is then defined to be a finite set of generalized tuples. For example, the constraint relation $R(x, y) = (20 < x) \vee (y < 7) \vee (x = y)$ has three generalized tuples $(20 < x), (y < 7)$ and $(x = y)$ and $R$ represents an infinite set of data points.

The formula of a generalized relation $R$ can also be viewed as the formula of its generalized tuples put in disjunctive normal form (DNF), $\psi_1 \vee \psi_2 \vee \cdots \vee \psi_n$. We may use $\varphi_R$ to denote a quantified formula corresponding to the relation $R$. A generalized database is a finite set of generalized relations.

Constraint databases have been an active area of database research for many years. One of the most challenging questions in the theoretical development of constraint databases is the expressive power of their querying formalism: what are the limitations of query languages for constraint databases[3]? In particular, the classical techniques for analyzing the expressive power of relational query languages no longer work in the context of constraint databases. In the past two decades, most questions on the expressive power based on some characterization of structures and a relational schema have been settled. These questions were reduced to those of the expressiveness of query languages over ordinary finite relational databases, with additional condition that databases may store numbers, and arithmetic operations (which form linear or polynomial equations) may be used in queries[4]. To evaluate a query in this constraint setting, each occurrence of a database symbol $D$ will be replaced by its finite representation as a set of constraints, and then one can apply the quantifier-elimination procedure to the above obtained resulting formula, and get a quantifier-free formula which is in a form of finite representation of the query output[2].

In this paper we first introduce a new type of data model called the constraint complex value model which is based on generalizing the underlying database to allow complex values, rather than atomic values alone as in previous research into constraint databases, and then we investigate the expressive power of query languages defined over this new model. We also observe that the utility of allowing complex values is supported by the fact that all the major commercial database systems, such as IBM's DB2 and Oracle Server, permit the use of complex values.

Our study extends research into constraint databases in two main areas (to be detailed soon): 1) we allow constraint complex values and 2) we consider second-order query languages for complex values. Many features of standard constraint databases could be extended in an well organized way and several new properties can be developed for this extended model.

Regarding the expressive power for logics for constraint databases, there are well-established results. Many results on expressive power use the notion of generic queries, which comes from the classical relational database setting. Therefore, to begin our investigation, we first review the concept of generic queries in our new problem setting.

A generic query is the one whose result is unchanged by a permutation on the underlying domain of data values.

For example, the answer to the parity query is the same for the input $\{1, 2, 3, 4, 5\}$ and for another input $\{a, b, c, d, e\}$, which is obtained by the permutation $1 \mapsto a$, $2 \mapsto b$, $3 \mapsto c$, $4 \mapsto d$, $5 \mapsto e$. The notion of generic is also sometimes referred to as data independence.

Next, we will briefly review the concept of collapse results which is the main tool adopted in exploring expressive power of queries in constraint databases.

A collapse result means that query class $A$ has no more expressive power with respect to some characteristics (e.g., generic queries) than query class $B$, where query class $A$ may appear to be much larger than query class $B$[3−5]. For example, given the real order field $(\mathbb{R}, +, *, 0, 1, <)$ and a relational signature $SC$,

the classes of generic queries in first-order ($\mathcal{FO}$) logic over the real order field and $SC$, under active domain interpretation, and in $\mathcal{FO}$ logic over the universal domain are the same, i.e., $\mathcal{FO}(\mathrm{SC}) = \mathcal{FO}_{\mathrm{act}}(\mathrm{SC})$[5], where $\mathcal{FO}_{\mathrm{act}}$ denotes first-order queries under active domain interpretation. It is worth remarking that many researches have focused on database generic queries with embedded finite model theory, where finite structures are embedded in an infinite structure.

In many advanced database applications, the data has a natural hierarchical structure, and this feature is more naturally modeled by allowing complex values in the data model rather than atomic values alone. Intuitively, complex value relations are relations in which their entries themselves may be tuples or (nested) relations. In other words, the complex value model allows using the tuple and set constructors recursively. It should be remarked that this model provides the core structure of object-relational databases and comprises an important component of many semantic models.

In this article we extend previous research on constraint databases by allowing the database to store complex values. This extended model allows us to represent nested finitely re-presentable relations and sets. Thus, applications involving natural spatial-temporal objects can be easily modeled, without converting the objects into flat relations. For example, many spatial databases involve hierarchical data. In particular, populations of cities, rainfall of regions, areas of river, etc., are properties associated with sets of possibly infinite points. Multi-layered geographic information systems (GIS) may represent many regions and channels which in turn are represented by several atomic spatial objects, like lines and triangles. These properties occur naturally in many advanced GIS and applications of the constraint data model and are more easily modeled by allowing the use of complex values.

We illustrate two motivating examples as follows.

*Example* 1. Given a set of geometry objects which are stored in an object-relational constraint database as shown in Table 2, we represent it as a complex value constraint database.

In Table 2, the third attribute is a complex value which is a tuple value. We store three objects insight in the third attribute: the first and the second components describe figures like the "eyes", and the third describes a figure like the "mouth". This constitutes the objects in a nested relation form in the constraint setting.

*Example* 2. Let us show another example. It represents a simple patient's clinical data in a temporal constraint database in Table 3. The third attribute is a complex value which is a set of tuple values expressed in constraints.

In addition to the above practical considerations, we consider constraint query languages in the contexts of the embedded finite model and constraint databases, motivated primarily by their expressive power. It is well-known from the literature that there are a number of limitations of first-order logic. For example, queries such as parity, majority, connectivity, transitive closure and acyclicity property are not definable in $\mathcal{FO}$ logic, with linear or polynomial constraints. It is natural to turn one's attention to a more expressive query language to bypass these limitations. Second-order constraint query languages for complex values appear to be a promising approach. Our goal is to study a fragment of second-order logic, called monadic second-order

**Table 2**. Relation Geometry $R$

| Area | Scope | Objects |
|---|---|---|
| $A_1$ | $(x^2/36+y^2/25 = 1)$ | $<(x^2+4x+y^2-2y \leqslant 4),\ (x^2-4x+y^2-2y \leqslant -4),\ (x^2+y^2-2y = 8 \wedge y < -1)>$ |
| $A_2$ | $(x^2/72+y^2/36 = 1)$ | $<(x^2+4x+y^2-2y \leqslant 4), \cdots>$ |
| $\vdots$ | $\vdots$ | $\vdots$ |

**Table 3**. Relation Health Records

| Name | Symptom | Diagnosis | Medication | Treatment (hour) | Note |
|---|---|---|---|---|---|
| John | $\{temperature, vomit\}$ | Infection | Antibiotics | $\{<(2 < stay < 5)>\}$ | $\cdots$ |
| | $\{temperature, dizzy\}$ | Unknown | Antibiotics + drug A | $\{<(ward, 5 < stay < 20),\ (21 < surgery < 22)>,$ $(22 < treatB < 24),\ (treatC, 22 < treatC < 23)\}$ | $\cdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| Peter | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| Maria | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |

($\mathcal{MSO}$), for constraint databases with complex objects. We follow a popular technique in the research of the embedded finite model and constraint databases, which adopts the tool of collapse results to analyze the expressive power of query languages.

Relational database queries are required to have a certain closure property: they return finite outputs on finite inputs. This requirement is well known in the database theory under the name of query safety: we identify those formulas which return finite results. In this paper, we consider this safety issue in the context of embedded finite complex value model.

We summarize our main results of this paper as follows.

• We extend the constraint data model to allow constraint complex values and propose the most important constraint query languages for embedded finite complex value model and constraint complex value databases.

• Unlike in standard first-order constraint query languages, we show that three paradigms of query languages: Calculus, Algebra, and Datalog, are equivalent for constraint databases with complex objects.

• We study second-order logic over the embedded finite complex value model and the constraint complex value databases settings. We show that the natural-active collapse with a condition and the active-generic collapse carry over to second-order logic for structures with o-minimality property and any signature in the complex value relations.

• The expressiveness results and complexity bounds of more powerful logics including monadic second-order logic with fix-points, and second-order, and various important constraint query languages for the complex object model are investigated in the paper.

• We discuss the data complexity for second-order logics over constraint databases. The main results are that the complexity upper bounds for three theories, $\mathcal{MSO}+\text{Lin}$, $\mathcal{MSO}+\text{Poly}$, and Inflationary $\text{Datalog}^{\text{cv},\neg}_{\text{act}}(SC,\mathfrak{M})$ without powerset operator are $\cup_i \sum_i^{NC^1}$, $\mathbf{NCH}=\bigcup_i \sum_i^{NC}$, and $\text{AC}^0$/poly, respectively.

• We show that the complex value Datalog language with stratified negation over polynomial constraints is closed. Some topological properties, for example, connectivity, can be defined in second-order logic.

• We develop an algorithm for detecting query safety. The issue of safe query translation is comprehensively investigated in the paper.

We introduce notations in Section 2. In Section 3, we review the concept of complex values and extend the constraint data model to constraint complex values. Then we first give a formal definition of the embedded finite complex value setting and define the second-order logic over this setting. We also propose the most important constraint query languages for this extended constraint data model. We analyze expressive power of logics in the context of constraint complex value data model in Section 4. The techniques that we use normally come in the form of collapse results. These techniques reduce many questions over constraint databases or embedded finite models to the classical finite model theory setting. In Section 5, we explore the complexity of various constraint queries. In Section 6, we consider the problem of safety in both embedded complex value model and constraint settings. The related work is briefly stated in Section 7. Finally, we give a conclusion in Section 8.

## 2 Preliminaries

We briefly review basic concepts from standard constraint databases and then extend these concepts to a more powerful extended constraint data model with complex objects and corresponding query languages beyond the first-order logic in the next section. Most notations are standard and adopted from the literature on constraint databases.

### 2.1 Databases over Underlying Infinite Structures

*Vocabulary and Structures.* Let $\Omega$ be a vocabulary (or called signature) which consists of a set of interpreted function symbols $\mathscr{F}$, a set of predicate symbols $\mathscr{P}$, and a set of constant symbols $\mathscr{C}$. Each function and predicate symbol has an associated arity. For example, $\Omega=(+,<,0,1)$ is a signature with one function symbol $+$ of arity two, one predicate symbol $<$ of arity two, and two constant symbols, 0 and 1. Let $\mathfrak{M}=\langle \mathscr{U},\Omega\rangle$ be an infinite structure, where $\mathscr{U}$ is an underlying infinite set (we normally call the universe or domain of the structure). For example, real field $\boldsymbol{R}(\mathbb{R},\Omega)=\boldsymbol{R}(\mathbb{R},+,\cdot,0,1,<)$ and real linear order group $\boldsymbol{R}_{\text{lin}}(\mathbb{R},\Omega)=\boldsymbol{R}(\mathbb{R},+,<,0,1)$ are two common used structures in the context of constraint databases, where $\mathbb{R}$ is the set of real numbers and $+,\cdot,<$ are the usual addition the multiplication and the ordering on $\mathbb{R}$. $0,1\in\mathbb{R}$.

A database schema $SC$ is a finite nonempty set of relational names $\{R_1,...,R_k\}$, each with a given arity $p_i$, $1\leqslant i\leqslant k$, $p_i>0$. An instance of $SC$ over a given

structure $\mathfrak{M} = (\mathscr{U}, \Omega)$ with $X \subseteq \mathscr{U}$ is a family of finite sets, $\{r_1, ..., r_k\}$, where $r_i \subseteq X^{p_i}$. It means that each schema symbol $R_i$ of arity $p_i$ is interpreted as a finite $p_i$-ary relation $r_i$ over $X$. We denote $Inst(SC, \mathfrak{M})$ as the set of all instances defined on schema $SC$ over $\mathfrak{M}$.

### 2.2 Logic

In order to introduce the necessary logics adopted in the development of this paper, we briefly review, for the sake of self-completeness, well known logics which are used in standard constraint databases as follows.

*First-Order Logic.* We first define a first-order logic over a structure $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$. We assume there exists a countably infinitely set of variables $\mathcal{V}$ corresponding to the structure $\mathfrak{M}$. In logics, terms are inductively defined as follows:

- each variable is a term,
- each constant symbol $c$ is a term, and
- if $t_1, ..., t_k$ are terms and $f$ is a $k$-ary function symbol, then $f(t_1, ..., t_k)$ is a term.

There are only two atomic formula forms: $t = t'$ and $p(t_1, ..., t_n)$, where $t, t', t_1, ..., t_n$ are terms, and $p$ is a predicate symbol in $\mathscr{P}$. A first-order logic formula is then built up from atomic formulas by using the Boolean connectives $(\vee, \wedge, \neg)$ and two quantifiers, $\forall$ and $\exists$. We denote by $\mathcal{FO}(SC, \mathfrak{M})$ the first-order logic over the structure $\mathfrak{M}$ and a database $D$ with schema $SC$. We interchangeably denote $\mathcal{FO}(SC, \mathfrak{M})$ or $\mathcal{FO}(D, \mathfrak{M})$.

For example, $\forall x(R(x, y) \wedge x = z)$ is a first-order formula, where $R$ is a predicate symbol in $SC$, $x$ is a quantified variables and $y, z$ are free variables. A formula which has no free variables is called a sentence. We normally write $\varphi(x_1, ..., x_n)$ to denote a formula with free variables $x_1, ..., x_n$. It is a standard manner for defining a formula $\varphi$ which holds over a given structure $\mathfrak{M}$ and a database $D$ by $(\mathfrak{M}, D) \models \varphi(\boldsymbol{a})$, where $\boldsymbol{a}$ is a vector of variables over the domain $\mathscr{U}$. It is simply written as $D \models \varphi(\boldsymbol{a})$ when the underlying structure $\mathfrak{M}$ is understood.

In standard constraint databases, we normally consider relational calculus, or, $\mathcal{FO}$ logic, over the underlying structure $\mathfrak{M}$ and the database $D$ with schema $SC$, as a basic query language formalism. A $\mathcal{FO}(SC, \mathfrak{M})$ formula $\varphi(x_1, ..., x_n)$ applied to $D$ is defined as $\varphi(\mathfrak{M}, D) \stackrel{\text{def}}{=} \{\boldsymbol{a} \in \mathscr{U}^n \mid (\mathfrak{M}, D) \models \varphi(\boldsymbol{a})\}$.

*Fixed-Point Logic.* Next, we review fix-point logics which are often used in database queries. Let $A$ be a set and $k$ is a positive integer. A $k$-ary operator on $A$ is an injective mapping $\Phi$: $\mathcal{P}(A^k) \to \mathcal{P}(A^k)$, where $\mathcal{P}(A^k)$ is the power set of $A^k$. A $k$-ary relation $R$ is a fixed-point of the operator $\Phi$ if $R = \Phi(R)$. Thus, each element $\boldsymbol{a}$ of a fixed-point $R$ of $\Phi$ satisfies the recursive specification

$$(\boldsymbol{a}) \in R \Leftrightarrow (\boldsymbol{a}) \in \Phi(R).$$

A least fixed-point of $\Phi$ is a fixed-point $X$ of $\Phi$ such that $X \subseteq Y$ for each fixed-point $Y$ of $\Phi$. We denote it as $\textsc{Lfp}(\Phi)$.

A predicate symbol $p$ occurs positively in a formula $\varphi$ if it occurs under the scope of an even number of negations. The syntactic expression for the least fixed-point extensions of the first-order logic is stated as follows.

**Definition 1**. *If $R$ is an $n$-ary relation symbol not in the database schema $SC$, and $R$ occurs positively in a first-order formula $\varphi(x_1, ..., x_k, \boldsymbol{y}, R)$, and $\boldsymbol{t}$ is a $k$-vector of variables or terms, then*

$$[\textsc{Lfp}_{\boldsymbol{x},R}\ \varphi(\boldsymbol{x}, \boldsymbol{y}, R)](\boldsymbol{t})$$

*is a fixed-point formula. The semantics is as follows: given an instance $D$ of $Inst(SC, \mathfrak{M})$ and $\boldsymbol{a}$ of the same length with $\boldsymbol{y}$, there exists a sequence $R_i^{\boldsymbol{a}}$ such that $R_0^{\boldsymbol{a}} = \emptyset$, and*

$$R_{i+1}^{\boldsymbol{a}} = \{(b_1, ..., b_k) \in \mathscr{U}^k \mid (\mathfrak{M}, D) \models \varphi(\boldsymbol{b}, \boldsymbol{a}, R_i^{\boldsymbol{a}})\}.$$

*This sequence is known to be monotonic, i.e., $R_i^{\boldsymbol{a}} \subseteq R_{i+1}^{\boldsymbol{a}}$.*

Based on this monotonic property, the sequence will eventually reach a fixed point (fixpoint), denoted by $R_\infty^{\boldsymbol{a}}$[6,7]. We denote $(\mathcal{FO}{+}\textsc{Lfp})(\mathfrak{M}, D)$ as the set of first-order least fix-point logical language over the structure $\mathfrak{M}$ and the database $D$①. We show how transitive closure can be expressed as a fixpoint operator.

*Example* 3. Given a graph $G(x, y)$, consider the transitive closure of this graph with distance at most $n$. It can be defined inductively using the following formula;

$$\textsc{Lfp}(T) = G(x, y) \vee T(x, y) \vee \exists z(T(x, z) \wedge G(z, y))$$

as follows:

$$R_0 = \emptyset,$$
$$R_n = \textsc{Lfp}(R_{n-1}), n > 0.$$

Here $\textsc{Lfp}(R_{n-1})$ denotes the result of evaluating $\textsc{Lfp}(T)$ when the value of $T$ is $R_{n-1}$. The sequence of $\{R_n\}$ converges and there exists some $k$ such that $R_k = R_j$ for each $j > k$. Therefore $R_k$ is a fixpoint of $\textsc{Lfp}(T)$.

① We just write $\mathcal{FO}{+}\textsc{Lfp}$ when $D, \mathfrak{M}$ are well-known.

There are two other semantics for the fixpoint operator. We briefly state them as follows (detailed explanations will be provided in the subsequent sections when we adopt them in the context). The first one inflationary fixpoint logic, denoted as $\mathcal{FO}$ + IFP, is defined as

$$[\text{IFP}_{\boldsymbol{x},R}\varphi(\boldsymbol{x},\boldsymbol{y},R)](\boldsymbol{t}).$$

Note that there is no imposing restriction on the occurrences of $R$. The semantics is the same as LFP but the formula construction is as follows:

$$R^{\boldsymbol{a}}_{i+1}=R^{\boldsymbol{a}}_i\cup\{(b_1,...,b_k)\in\mathscr{U}^k|(\mathfrak{M},D)\models\varphi(\boldsymbol{b},\boldsymbol{a},R^{\boldsymbol{a}}_i)\}.$$

The other one is partial fixpoint logic, $\mathcal{FO}$+PFP, which is defined as LFP, and the fixpoint is $R^{\boldsymbol{a}}_\infty$ if it exists, and $\emptyset$ otherwise.

### 2.3 Standard Constraint Database Model

We now give a formal definition of "constraint" before we define what a constraint database is. Given a structure $\mathfrak{M}=\langle\mathscr{U},\Omega\rangle$, a constraint over $\Omega$ on $\mathscr{U}$, written as $\Omega$-constraint, is an atomic first-order formula or the negation of an atomic formula. For example, $x_1+x_2<y$ and $x_2-y<0$ are two constraints over the structure $\mathfrak{M}=\langle\mathbb{R},+,<,0,1\rangle$.

We now review the standard constraint database model[2,8]. Given a vocabulary $\Omega$,

• a constraint $k$-tuple over $\Omega$ with variables $x_1,...,x_k$, is a finite conjunction formula $\varphi_1\wedge\cdots\wedge\varphi_l$, where each $\varphi_i, 1\leqslant i\leqslant l$, is an $\Omega$-constraint, and the variables in each $\varphi_i$ are among $x_1,...,x_k$;

• a constraint relation $R$ of arity $k$ over $\Omega$ is a finite set $r(R)=\{\gamma_1,\cdots,\gamma_m\}$, where each $\gamma_i$, $1\leqslant i\leqslant m$, is a constraint $k$-tuple with same variables, and we denote this constraint relation as a quantifier free disjunctive formula $\varphi_r=\gamma_1\vee\cdots\vee\gamma_m$;

• a constraint database $D$ is a finite collection of $n$ constraint relations $\varphi_r$, i.e., $D=\varphi_{r_1}\cup\cdots\cup\varphi_{r_n}$.

Let us give a simple example using polynomial inequality constraints.

*Example* 4. The constraint relation $r$ consists of the two constraint tuples $(x^2+y^2=1)$ and $(y=2\times x^2\wedge y<4)$. The disjunctive normal form formula (abbreviated as DNF) corresponding to this relation $r$ is

$$\varphi_r(x,y)=(x^2+y^2=1)\vee(y=2\times x^2\wedge y<4).$$

This formula describes an infinite set of points which include a circle and a part of parabola with $y$ less than 4.

The set of points occurred in the constraint relations of a database $D$ is called the active domain of $D$, and is denoted by $adom(D)$. For example, given a formula $\forall x,yS(x,y)$, the active domain of this formula is all points $(x,y)$ belonging to $S$.

### 2.4 Constraint Queries

As stated before, the first-order queries over $\mathfrak{M}$ and $SC$, are denoted as $\mathcal{FO}(SC,\mathfrak{M})$. There are two different interpretations of quantification in first-order logic. One quantification, called natural semantics, is quantified over the universal domain $\mathscr{U}$ of the infinite structure $\mathfrak{M}$, and the other quantification ranges over the active domain $atom(D)$. It is called active domain semantics.

Given a structure $\mathfrak{M}=\langle\mathscr{U},\Omega\rangle$, a set $X\subseteq\mathscr{U}^n$ is called $\mathfrak{M}$-definable if there exists a formula $\varphi(x_1,...,x_n)$ in the language of $\Omega$ such that $X=\{\boldsymbol{a}\in\mathscr{U}^n\mid\mathfrak{M}\models\varphi(\boldsymbol{a})\}$[8].

Below is a formal definition for constraint queries. Let $\mathfrak{M}=\langle\mathscr{U},\Omega\rangle$ be a structure. Let $D$ be a constraint database and $k$ a natural number. A $k$-ary constraint query $Q$ on $(\mathfrak{M},D)$ is a function which maps a constraint database $D$ over $\mathfrak{M}$ to a $k$-ary constraint relation $Q(D)$ such that it is closed, i.e., it is defined in terms of $\Omega$. Note that here $Q$ could be a language $\mathbb{L}(\Omega,SC)$ which is beyond $\mathcal{FO}$. In constraint databases, we normally consider relational calculus, or, $\mathcal{FO}$ logic, as a basic query language.

As in classical relational databases, a query is regarded as a mapping from the underlying database to an answer relation. However, in the constraint setting we should also consider the consistency issue, i.e., a query on constraint representation level should correspond to a query on the unrestricted conceptual level. The term unrestricted relation refers to arbitrary finite or infinite sets of points in a $k$-dimensional space. In contrast, by constraint relation we mean that it is finitely definable on a structure with some class of constraints.

The following closure property plays an important role in constraint databases. Given a structure $\mathfrak{M}=\langle\mathscr{U},\Omega\rangle$, for each input constraint database $D$ which is definable on $\mathfrak{M}$ , if an unrestricted query $Q$ on $D$, $Q(D)$ is also definable using $\Omega$-constraints, then we say query $Q$ is closed on $\mathfrak{M}$.

*Example* 5. Given a database $D$ over a structure $\mathfrak{M}=(\mathscr{U},\Omega)$, $\Omega=\{+,\times,<,0,1\}$, let $Q$ be a constraint query which maps $D$ to the set of integers. This is not

a closed query as the answer (a subset of $\mathbb{Z}$) cannot be expressed as a formula in terms of signature $\Omega$.

There are two key properties, quantifier elimination and o-minimality, over a nice structure, where by a nice structure we mean it possesses good behaviors such as a collapse result which is a well-known concept in standard constraint databases[9]. The first one, quantifier elimination, is a very important property of structures for which constraint databases can behave well and admit collapse results. Given a structure $\mathfrak{M}$, if for every formula $\varphi(\boldsymbol{x})$ there exists a quantifier free formula $\psi(\boldsymbol{x})$ such that

$$\mathfrak{M} \models \forall \boldsymbol{x}.\varphi(\boldsymbol{x}) \leftrightarrow \psi(\boldsymbol{x}),$$

we say that the structure $\mathfrak{M}$ admits quantifier elimination[10,11]. The second notion is o-minimality. It is defined as follows. A structure $\mathfrak{M}$ is o-minimal, if the points of every formula definable over $\mathfrak{M}$ consist of a set of finite points and open intervals $\{x \mid a < x < b\}$, $\{x \mid a < x\}$, and $\{x \mid x < b\}$[12].

*Genericity.* Many results on expressive power use the notion of genericity. We now review the genericity of Boolean queries and non-Boolean queries[3]. Given a function $\pi : \mathscr{U} \to \mathscr{U}$, we extend it to a finite $SC$-structure $D$ by replacing each occurrence of $a \in adom(D)$ with $\pi(a)$.

• A Boolean query $Q$ is totally generic (order-generic) if for every partial injection function (partial monotone injection function, resp.) $\pi$ defined on $adom(D)$, $Q(D) = Q(\pi(D))$.

• A non-Boolean query $q$ is totally generic (order-generic) if for every partial injection function (partial monotone injection function, resp.) $\pi$ defined on $adom(D) \cup adom(Q(D))$, $\pi(Q(D)) = Q(\pi(D))$.

*Ramsey Property*[13,14]. Ramsey property plays an important role in investigating expressive power of query languages in the constraint setting. In particular, it applies to active semantics queries. Let $\mathbb{L}$ be a logic. We say it has a Ramsey property if, given an ordered structure $\mathfrak{M} = <\mathscr{U}, \Omega>$ and for any database $D$, the following holds: let $\varphi(\boldsymbol{x})$ be a $\mathbb{L}$-formula in a form of language $L(D, \mathfrak{M})$ with free variables $\boldsymbol{x}$, and $X$ an infinite subset of $\mathscr{U}$. Then there exists an infinite set $Y \subseteq X$ and an equivalent formula $\psi(\boldsymbol{x})$ in a form of $L(D, <)$ such that for any database $D$ with $adom(D) \subset Y$, and for any $\boldsymbol{a}$ over $Y$, the implication $(\mathfrak{M}, D) \models \varphi(\boldsymbol{a}) \leftrightarrow \psi(\boldsymbol{a})$ holds. It is well known that the first-order logic has the Ramsey property over any ordered structure[9,15]. In this paper, we will investigate whether Ramsey property still holds for higher-order logics.

Classical model theory provides us with many examples of well-behaved structures which have been considered in the field of constraint databases. A few are listed below:

- dense order constraints $(\mathbb{R}, <)$, $(\mathbb{Q}, <)$;
- linear constraints $\boldsymbol{R}_{\text{lin}} = (\mathbb{R}, +, 0, 1, <)$;
- polynomial constraints $\boldsymbol{R} = (\mathbb{R}, +, *, 0, 1, <)$;
- exponential constraints $\boldsymbol{R}_{\text{exp}} = (\mathbb{R}, +, *, e^x, <)$.

The above structures admit quantifier-elimination and are o-minimal structures. They all have the complete theory, i.e., the set of all true first-order sentences is effectively decidable.

## 3 Constraint Complex Object Model

We now start to present our constraint complex object model and its corresponding query languages. We first review the concept of complex values (CV) and extend the constraint relational data model to constraint complex object data model.

Complex values are formed by using two constructors: *tuple* and *set*, and associated with sorts. We define syntax and semantics for complex values as follows. We denote all constants appeared in databases as $\boldsymbol{dom}$. The sort $\tau$ of an attribute or a relation is the type of that specific attribute or structure of that relation. The set of (complex) values of sort $\tau$ (i.e., the interpretation of $\tau$) is defined as follows[16].

**Definition 2**. 1) *The abstract syntax of sorts is given by* $\tau = \boldsymbol{dom} \mid < A_1 : \tau_1, \cdots, A_k : \tau_k > \mid \{\tau\}$, *where* $k \geqslant 0$ *and* $A_1, \cdots, A_k$ *are distinct attributes.* 2) *The interpretation of sort* $\tau$ *(i.e., the set of values of* $\tau$*), denoted as* $[\![\tau]\!]$, *is defined recursively as follows.*

• $[\![\boldsymbol{dom}]\!] = \boldsymbol{dom}$,

• $[\![\{\tau\}]\!] = Powerset([\![\tau]\!]) = \{X | X \subseteq [\![\tau]\!]$ and $X finite\}$, and

• $[\![A_1 : \tau_1, \cdots, A_k : \tau_k]\!] = [\![\tau_1]\!] \times \cdots \times [\![\tau_k]\!]$.

A complex value database schema $SC$ is a finite set of relation names $\{R_1, ..., R_l\}$ with associated sorts $\tau_1, ..., \tau_l$.

*Example* 6. Assume that a database contains a single relation $R$ with the associated sort $\{< X : \boldsymbol{dom}, Y : \{< A : \boldsymbol{dom}, B : \{\boldsymbol{dom}\} >\} >\}$. One complex value of this $R$ is

$$\{< X : x_1, Y : \{< A : a_1, B : \{b_1, b_2, b_3\} > \\ < A : a_2, B : \{\} >\} >\}.$$

Most questions about the expressive power of query languages over constraint databases are easily reduced

to questions about the expressiveness of query languages over the embedded finite model, which are databases stored with numbers and arithmetic operations may be applied to numbers[17].

We first present this embedded finite model in the complex value case.

### 3.1 Embedded Finite CV Model

We define the embedded finite CV setting which is an extension of the embedded finite model. Then we define monadic second-order logic ($\mathcal{MSO}$) over this setting.

**Definition 3**. *Let $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$ be an infinite structure on a set $\mathscr{U}$, where $\Omega$ is a signature. Let $SC$ be a complex value relational signature $\{R_1, ..., R_l\}$ where each relation symbol $R_i$ has sort $\tau_i$. Then an embedded finite complex value model is a structure*

$$D = (A, R_1, ..., R_l),$$

*where each $R_i$ is a finite subset of $\mathscr{U}^{\tau_i}$ ($\mathscr{U}^{\tau_i}$ denotes the set of complex value constants with sort $\tau_i$), and the set $A$ is the $adom(D)$.*

Due to the hierarchical structure in the embedded finite complex value setting, we use a higher order logic. We first consider $\mathcal{MSO}$ which is a fragment of second-order logic. It has been widely adopted in several research areas such as database theory, computational complexity, and artificial intelligence.

We review the definition of monadic second-order logic as follows. It will be the main logic that we adopt for the foundation for query languages for constraint CV-databases.

**Definition 4**. *Given a structure $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$ and a complex value relational signature $SC$, the monadic second-order logic $\mathcal{MSO}$ over $\mathfrak{M}$ and $SC$, denoted by $\mathcal{MSO}(SC, \mathfrak{M})$, is defined as follows.*

- *Any atomic $\mathcal{FO}$ formula in the language of $\mathfrak{M}$ is an atomic formula of $\mathcal{MSO}(SC, \mathfrak{M})$.*
- *If $R \in SC$ is a $k$-ary relation variable and $t_1, ..., t_k$ are first-order terms, then the expression $R(t_1, ..., t_k)$ is an atomic formula.*
- *If $S$ is a unary set variable that ranges over sets of elements of domain and $t$ is a term in the language of $\Omega$, then the expression $t \in S$ (also written as $S(t)$) is an atomic $\mathcal{MSO}(SC, \mathfrak{M})$.*
- *Formulas of $\mathcal{MSO}(SC, \mathfrak{M})$ are closed under the Boolean connectives ($\wedge, \vee$, and $\neg$).*
- *If $\varphi$ is a first-order formula that contains a unary set variable $S$, then the following: $\exists S\varphi$, $\forall S\varphi$ are $\mathcal{MSO}(SC, \mathfrak{M})$ formulas.*

Note that $\mathcal{MSO}$ is a restriction of second-order logic in which the quantification over unary relations (i.e., sets) is allowed. Quantification over functions is not allowed. If $R$ and $S$ are set variables, then $R \subseteq S$ can be derived from the above definition.

The active-domain semantics of monadic second-order logic, denoted by $\mathcal{MSO}_{\text{act}}(SC, \mathfrak{M})$, is those formulas in which all first-order and second-order quantifiers range over the active domain. We allow terms of the form

$$\{x \mid \psi(x, y_1, ..., y_k)\},$$

where the free variables are $x, y_1, ..., y_k$. We denote it as $Q(x; \boldsymbol{y})$ which is called a parameterized query. Intuitively, the variables $y_i$ are provided by binding values and we obtain an $\mathcal{MSO}$ query with a distinguished free variable $x$.

We study monadic second-order logic, $\mathcal{MSO}(SC, \mathfrak{M})$, and investigate its applications to query languages for the constraint CV-databases. As in the first-order logic formalism, the expressive power and query evaluation issues need to be solved by new techniques and their solutions depend heavily on the model-theoretic properties of the underlying structure $\mathfrak{M}$. The fundamental complex value structure and the second-order logic also greatly impact on these solutions.

### 3.2 Constraint Complex Object Model

In this subsection, we extend the constraint relational data model to the constraint complex object data model. The constraint complex object data model has been proposed in [18]. However, it received less attention in the literature. Therefore in this subsection, we formally present its formalism and corresponding query languages.

A complex value database schema $SC$ is a finite set of relation names $\{R_1, ..., R_l\}$ with associated sorts $\tau_1, ..., \tau_l$. We define the constraint complex value database model which allows us to represent finitely re-presentable infinite objects in a nested setting, i.e., an attribute of the schema can contain many finitely re-presentable objects in a tuple form or a set format. Therefore, constraint complex values are built using tuple and set constructors recursively from generalized constraint tuples and finite re-presentable sets.

Many properties of natural spatial/temporal objects can be easily modeled as constraint complex values.

**Definition 5**. *In the context of the constraint complex value model, for each sort $\tau$, the domain of $\tau$ denoted as $dom(\tau)$ is defined recursively as follows.*

• *If $\tau$ is an n-ary flat tuple type, i.e., the attributes are ranged over universal domain, then $dom(\tau)$ is the set of all generalized constraint n-ary tuples.*

• *If $\tau = \{\tau'\}$ is a set type, then* $dom(\tau) \stackrel{\text{def}}{=} \{\bigvee_{i=1}^{k} \psi_i | k \geqslant 1, \forall i \in \{1, \cdots, k\}, \psi_i \in dom(\tau')\}$.

• *If $\tau = <\tau_1, \cdots, \tau_k>$ is a tuple type, then*

$$dom(\tau) \stackrel{\text{def}}{=} \{\bigwedge_{i=1}^{k} \psi_i | \psi_i \in dom(\tau_i), \text{ if } \tau_i \text{ is not a set tuple}; \psi_i \equiv x_i = \{\phi_i\}, \text{where } \phi_i \in dom(\tau_i{}') \text{ if } \tau_i \text{ is a set tuple } \{\tau_i{}'\}\}.$$

*Example* 7. Let $\tau = <x : \{<x_1 : \mathbb{Q}, x_2 : \mathbb{Q}>\}, y : \mathbb{Q}>$ be a tuple type. $\mathbb{Q}$ stands for rational numbers. A constraint complex value of type $\tau$ is

$$\psi(x, y) \stackrel{\text{def}}{=} (x = \{<x_1, x_2> | \phi\} \wedge y = 10),$$
where $\phi = (0 < x_1 < 5) \wedge (x_1 + x_2^2 < 10)$.

### 3.3 Query Languages for Constraint CV-Databases

We briefly describe three paradigms for query languages that can be used for querying constraint CV-databases.

#### 3.3.1 Calculus Queries

In the logic paradigm, we denote $\mathcal{CALC}^{\text{cv}}(SC, \mathfrak{M})$ as some sort of $\mathcal{SO}$ formulas in the language that contains symbols of schema $SC$ and the language of $\mathfrak{M}$. That is, $\mathcal{CALC}^{\text{cv}}(SC, \mathfrak{M})$, $(\mathfrak{M} = \langle \mathcal{U}, \Omega \rangle)$ formulas are built up from the atomic positive literal

$$R(t),\ t = t',\ t \in t',\ \text{or } t \subseteq t',$$

and $\Omega$ formulas by using Boolean connectives $\vee$, $\wedge$, $\neg$, and quantifiers $\forall$, $\exists$.

We refer to the above syntactic query languages as complex value calculus with $\Omega$ constraints. For a structure $\mathfrak{M}$ and an $SC$-database instance $D$, the notion of $(D, \mathfrak{M}) \models \varphi$ is defined in a standard way. If $\mathfrak{M}$ is understood, we write $D \models \varphi$.

*Example* 8. Let $\mathfrak{M} = (\mathbb{R}, +, -, 0, 1, <)$. The following calculus query applies to schema $SC = \{R, S\}$ with sort $\tau_R = \tau_S = <\mathbb{R}, \{\mathbb{R}\}>$.

$$\varphi \equiv \exists u, v. R(x, u) \wedge S(y, v) \wedge x \in u \wedge y \in v \wedge x + y < 10.$$

It defines a join operation with the condition that for each pair of joining tuples, $(x, u)$ and $(y, v)$, $x$ and $y$ must be a member of the second component $u$ and $v$ respectively, and $x$ plus $y$ is less than 10.

#### 3.3.2 Algebra Queries

The algebra paradigm adopts the classical complex value relational algebra approach which provides some algebraic operators for manipulating constraint CV-databases. We denote this algebra as $\mathcal{ALG}^{\text{cv}}$. In this subsection, we give a general definition of the constraint algebra operators.

The complex value algebraic expressions over a structure $\mathfrak{M} = <\mathscr{U}, \Omega>$ and a database schema $SC$ are inductively defined as follows. Let $r$ be a relation of sort $\tau$. $\varphi_r = \psi_1 \vee \cdots \vee \psi_n$ is the constraint formula which corresponds to $r$.

• Constant values: each $a \in \mathcal{U}$ is an algebraic query.

• Each relation $R \in SC$ is an algebraic query.

• Set operators $\cap$, $\cup$, and $-$ are defined in the obvious manner.

We illustrate "$-$" as follows. Suppose that $q = q_1 - q_2$, $\lambda$, $\alpha$, and $\beta$ are algebraic expressions of $q$, $q_1$ and $q_2$ respectively.

$$\lambda \equiv (\bigvee_{\varphi \in \alpha} \varphi) \wedge \neg(\bigvee_{\psi \in \beta} \psi).$$

Then put $\lambda$ in a disjunctive normal form.

• Selection operator: the output of the selection operator is the conjunction of the constraint tuples and the selection condition. That is,

$$\sigma_\theta \varphi_r = \bigvee_{1 \leqslant i \leqslant n} (\psi_i \wedge \theta).$$

The selection condition $\theta$ is of the form $x_i = d$, $x_i = x_j$, $x_i \in x_j$ or $x_i = x_j.C$.

• When applying projection operator, we use the existential quantifier to eliminate required variables from each constraint tuple. For example, let $X = \{x_1, ..., x_k\}$ be the set of variables in $\varphi_r$, $Y = \{y_{j_1}, ..., y_{j_p}\}$. After renaming the variables $x_1, ..., x_p$ to $y_{j_1}, ..., y_{j_p}$,

$$\pi_Y \varphi_r = \chi(x_{p+1}, ..., x_k) \equiv \exists y_{j_1}, ..., \exists y_{j_p} \varphi_r \bigwedge_{i=1}^{p} x_i = y_{j_i}.$$

• The join operator pairs each constraint tuple from two relations[19].

• Other constructive operations:

*Powerset*: $powerset(r)$ is a relation of sort $\{\tau\}$ where $powerset(r) = \{\nu \mid \nu \subseteq r\}$;

*Tuple Creation*: if $A_1, ..., A_n$ are distinct attributes, $tup\_create_{A_1,...,A_n}(r_1, ..., r_n)$ is of sort $<A_1 : \tau_1, ..., A_n : \tau_n>$, and $tup\_create_{A_1,...,A_n}(r_1, ..., r_n) = \{<A_1 : \nu_1, ..., A_n : \nu_n> | \forall i(\nu_i \in r_i)\}$;

*Set Creation*: $set\_create(r)$ is of sort $\{\tau\}$, and $set\_create(r) = \{r\}$;

*Tuple Destroy*: if $r$ is of sort $< A : \tau' >$, $tup\_destroy(r)$ is a relation of sort $\tau'$ and $tup\_destroy\ (r) = \{\nu \mid < A : \nu > \in r\}$;

*Set Destroy*: if $\tau = \{\tau'\}$, then $set\_destroy(r)$ is a relation of sort $\tau'$ and $set\_destroy(r) = \cup r = \{w \mid \exists \nu \in r, w \in \nu\}$.

The detailed description of these complex value algebraic operations can be found in [1].

*Example* 9. Given a graph with natural numbers representing its nodes, we wish to retrieve the pairs $(x, y)$ of the graph paths $E(x, z)$ and $E(z, y)$ in which $x, y, z$ satisfy the condition, $z = x^2 + y^2$. According to the above algebra definition, it is straightforward to write the query as the following algebraic expression:

$$\pi_{x,y}(\sigma_{E_1 \cdot z = E_2 \cdot z \wedge (z = x^2 + y^2)}(E_1(x, z) \times E_2(z, y)),$$

where $E_1$ and $E_2$ are two copies of the graph paths.

### 3.3.3 Datalog Queries

It is natural to consider a deductive paradigm for use as a constraint query language. This paradigm provides a logic programming style for reasoning query results. Indeed the constraint databases are originally inspired by deductive databases and constraint logic programming. In the following we briefly give formal definition for Datalog$^{cv}$ for CV-databases with constraints, unrestricted query expressed by Datalog$^{cv}$ program, and Datalog$^{cv}$ program with stratified negation.

**Definition 6**. *Let $SC_2$ be an intentional schema disjoint from a given schema $SC_1$. A* Datalog$^{cv}$ *program over a given structure $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$ with schema $SC_1$ is a set of rules in the form of*

$$H(...) \leftarrow L_1, \cdots, L_n,$$

*where the head $H(...)$ is a derived predicate which is a relation name in $SC_2$, and each $L_i$ of the body is a literal which is an atomic formula over the combined vocabulary $(\Omega, SC_1, SC_2)$.*

A Datalog$^{cv}$ query is a pair $(P, \alpha)$ where $P$ is a finite set of rules, and $\alpha$ is a derived answer relation.

*Example* 10. Let $\tau_p = \tau_t = \mathbb{R}$, $\tau_r = \tau_s = (\mathbb{R}, \{\mathbb{R}\})$ and $\tau_q = (\mathbb{R}, \mathbb{R})$. The following is a constraint Datalog query.

$$\begin{aligned} &s(x, z) \leftarrow r(x, y) \wedge z \subseteq y \wedge 5 \notin z, \\ &q(x, v) \leftarrow s(x, z) \wedge v = count(z), \\ &t(x) \leftarrow p(x) \vee (q(x, c) \wedge x + c < 100). \end{aligned}$$

Now we define the semantics of the unrestricted query $(P, \alpha)$ expressed by a Datalog program as follows.

**Definition 7**. *Let $(P, \alpha)$ be a* Datalog$^{cv}$ *query program over a structure $\mathfrak{M}$ with intentional schema $SC_2$, and let $D$ be an unrestricted complex object database over $\mathfrak{M}$ with schema $SC_1$. In the following we define an operator, denoted by $T_P$. Let $D'$ be a CV-database. Let $H$ be a relation name in $SC_2$. Then $T_P(D') \mid_H$ generates all tuples $(a_1, ..., a_k)$ over $\mathfrak{M}$ such that there exists a rule*

$$\alpha : H(x_1, ..., x_k) \leftarrow L_1, \cdots, L_n$$

*in $P$, and a valuation $v$ of $\alpha$ in $D'$ such that*

$$(a_1, ..., a_k) = (v(x_1), ..., v(x_k)).$$

*This valuation makes all $L_i$ true in $(\mathfrak{M}, D, D')$.*

$T_P$ plays the role of evaluation mechanism.

**Definition 8** (Semi-Positive Datalog$^{cv}$). *Recalling Definition* 6*, we allow each $L_i$ in the body to be also a negated atomic formula in the form of $\neg S$, where $S$ is a relation name of the input schema $SC_1$. We call this extended* Datalog *as semi-positive* Datalog$^{cv}$.

**Definition 9** (Datalog with Stratified Negation). *Suppose that $P_1, ..., P_k$ are semi-positive programs. For each $i < j \leqslant k$, the right hand side literal $L$ and the intentional schema of $P_i$ become a part of the input schema for $P_j$. Such a sequence of semi-positive programs is called a* Datalog$^{cv,\neg}$ *with stratified negation.*

Note that by Definition 9 we can infer that a stratification of a program $P$ is a partition $P_1, \cdots, P_n$ of the program such that no relation symbol $R$ that is negated in a $P_i$ is a derived relation in any $P_j$ with $j \geqslant i$.

A program is stratified, if there is a stratification for it. The output of a stratified Datalog program query is called the perfect model.

## 4 Expressive Power

As in standard CV relational databases, a constraint calculus formula and a corresponding constraint relational algebra are regarded as equivalent. It means that they define the same unrestricted query. We first explore whether both calculus and algebra paradigms defined in the previous section indeed possess this fundamental property. Our main motivation is to understand deeply the expressive power of higher-order logic applied in the realm of topology and geometry areas.

With constraint setting, we observe that it does not give different insight results. Recall that the complete theory means the set of all first-order sentences true in the underlying structure is effectively decidable.

We state our findings as follows.

**Theorem 1**. *Let $\mathfrak{M} = <\mathscr{U}, \Omega>$ be an infinite structure with the complete theory. For each complex value*

*calculus formula over* $(SC, \mathfrak{M})$*, there is an effective algorithm to convert it into an equivalent constraint complex value algebra over* $(SC, \mathfrak{M})$*, and vice versa.*

*Proof.* In standard (CV) relational databases, the safe-range (defined in Section 6) calculus and algebra expression coincide. Adding atomic constraints or combination of constraint terms would not generate a new problem in terms of conversion and will not affect coinciding the property of both paradigms. It means that for each calculus formula

$$\varphi = (\bigwedge_{i=1}^{n} R_i) \wedge (\bigwedge_{i=1}^{n} \beta_i)$$

is equivalent to

$$\alpha \wedge (\bigwedge_{i=1}^{n} \beta_i),$$

where $\alpha$ is an algebraic expression and $\beta_i$ are constraint terms.

Without considering the safety issue, every terminated constraint calculus formula and its corresponding CV constraint algebra do have same expressive power. It is well-known that there is an effective translation algorithm to convert standard (CV) calculus formula into equivalent algebraic expression[16]. Accordingly, there is also an effective algorithm to transform every terminated constraint calculus formula into algebra without violation by adding constraint setting. □

**Theorem 2**. *Let* $\mathfrak{M}$ *be the theory of dense order constraints over the rationals. Then the following two language classes are equivalent: stratified* $Datalog^{\mathrm{cv},\neg}(SC, \mathfrak{M}) \equiv \mathcal{CALC}^{\mathrm{cv}}(SC, \mathfrak{M})$.

*Proof.* In complex value databases without constraints, we know that $\mathcal{CALC}^{\mathrm{cv}}$ is equivalent to $\textsc{Datalog}^{\mathrm{cv},\neg}$ with stratified negation[20]. As the underlying structure $\mathfrak{M}$ is dense order constraints over the rationals, the $\textsc{Datalog}^{\mathrm{cv}}$ program will terminate and is closed[3]. Under this circumstance, the embedded constraints will hold for two paradigms without changing equivalence property. □

From Theorem 1 and Theorem 2, we easily obtain the following results.

**Corollary 1**. *For any structure* $\mathfrak{M}$*, stratified* $\textsc{Datalog}^{\mathrm{cv},\neg}(\mathfrak{M})$ ≡ *closed* $\mathcal{CALC}^{\mathrm{cv}}$ + $\textsc{Lfp}(\mathfrak{M})$ ≡ $\mathcal{CALC}^{\mathrm{cv}}(\mathfrak{M})$.

*Proof.* We know that closed $\mathcal{CALC}^{\mathrm{cv}}$ + Lfp and $\mathcal{CALC}^{\mathrm{cv}}$ are equivalent[16,20]. Adding constraint setting, it does not affect equivalence relationship. By Theorem 1 and Theorem 2, stratified $\textsc{Datalog}^{\mathrm{cv},\neg}(\mathfrak{M})$ ≡ closed $\mathcal{CALC}^{\mathrm{cv}}$ + $\textsc{Lfp}(\mathfrak{M})$ ≡ $\mathcal{CALC}^{\mathrm{cv}}(\mathfrak{M})$ for any structure $\mathfrak{M}$. □

In database theory, we often tackle the issue of expressive power of some logic (or query language). The expressive power of a logic deals with a specific problem: what can or what cannot be expressed on the logic. In the literature, the methods used for analyzing expressive power of constraint databases normally adopt techniques in the form of collapse results[4,15,21], complexity-theoretic techniques[22,23] or topological methods[24,25]. The purpose of the first and the second methods is to reduce several important problems over constraint databases or the embedded finite model to the classical finite model theory setting[3]. The topological method is to investigate its fundamental geometry property and applications. Most work focuses on the expressive power of generic queries. However, beyond generic queries, in the constraint setting we obviously gain extra expressive power beyond the $\mathcal{FO}$ or $\mathcal{MSO}$ capability.

We first provide a general definition of the various collapse phenomena occurred in logics from the literature as follows. The main goal of this section is to explore whether the $\mathcal{SO}$ logic and its fragments still possess the collapse results which have been investigated in depth in the $\mathcal{FO}$ logic. We then investigate the issue of what properties of the underlying structure will impact these collapse results. We will also investigate how the power-set operator affects the extensional aspect of expressive power of queries in the constraint setting[26].

**Definition 10**. *Given a structure* $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$ *and SC standing for any (CV-) relation schema, we say that a logic* $\mathbb{L}$ *has:*

- *a natural-active collapse over the structure* $\mathfrak{M}$ *if* $\mathbb{L}(SC, \Omega) = \mathbb{L}_{\mathrm{act}}(SC, \Omega)$*, that is, for every schema SC, and for every formula* $\varphi(\boldsymbol{x})$ *in the language* $\mathbb{L}(SC, \Omega)$*, there exists an equivalent formula* $\varphi_{\mathrm{act}}(\boldsymbol{x})$ *in the same language under active-semantics*;
- *an active-generic collapse over the structure* $\mathfrak{M}$ *if the classes of order-generic queries in* $\mathbb{L}_{\mathrm{act}}(SC, \Omega)$ *and* $\mathbb{L}_{\mathrm{act}}(SC, <)$ *are the same, that is, for every schema SC, every* $\mathbb{L}_{\mathrm{act}}(SC, \Omega)$*-definable generic query is already* $\mathbb{L}_{\mathrm{act}}(SC, <)$*-definable*;
- *natural-generic collapse if the classes of order-generic queries in* $\mathbb{L}(SC, \Omega)$ *and* $\mathbb{L}(SC, <)$ *are the same, that is, for every schema SC, every* $\mathbb{L}(SC, \Omega)$*-definable generic query is already* $\mathbb{L}(SC, <)$*-definable.*

Intuitively, the basic concepts of the above definition are that 1) natural-active collapse: adding a more powerful form of quantification (e.g., from quantification over the active domain to the universal domain) does not add any expressive power to the language; 2)

generic collapse: adding additional operations to the signature gains no power to express any generic (pure) queries.

### 4.1 Embedded Finite CV Model

It is known that the Ramsey property is applicable beyond the first-order logic case[8]. Therefore, $\mathcal{FO}$+Lfp and $\mathcal{SO}$ have locally generic collapse property and the proof for $\mathcal{FO}$+Lfp has been shown in [8]. Next we provide a detailed proof of active-generic collapse for the $\mathcal{MSO}$ language which is omitted in the literature.

Before we proceed to do it, two needed lemmas are presented first.

**Lemma 1**[8]. *Let $\varphi(\boldsymbol{x})$ be an atomic $\mathcal{FO}(\Omega)$ formula. Then $\varphi$ has the Ramsey property.*

Then we state the fundamental results of collapse as facts which we will adopt them in the proofs of our new findings.

Fact 1 (Active-Generic Collapse)[9,27]. *Every ordered structure admits active-generic collapse.*

**Lemma 2**. *Let $\varphi(\boldsymbol{x})$ be an $\mathcal{MSO}(SC,\Omega)$ formula. Then there exists an equivalent formula $\psi(x)$ such that every subformula of $\psi$ is either an $\mathcal{MSO}(SC)$ formula which comprises $R(t)$, $t \in R$, $S = R$, where $S$, $R$ are set variables or one predicate in $SC$, or an $\mathcal{MSO}(\Omega)$ formula.*

*Proof.* For any atomic formula of the form $R(t_i(\boldsymbol{y}))$, $t \in R(t_i(\boldsymbol{y}))$, $S(t_i(\boldsymbol{y})) = R(s_i(\boldsymbol{y}))$, we replace it as the following forms:

1) $\exists z_1 \in adom, ..., \exists z_k \in adom$

$$\bigwedge_i (z_i = t_i(\boldsymbol{y})) \wedge R(z_1, ..., z_k),$$

2) $\exists z_1 \in adom, ..., \exists z_k \in adom$

$$\bigwedge_i (z_i = t_i(\boldsymbol{y})) \wedge t \in R(z_1, ..., z_k),$$

3) $\exists z_1 \in adom, ..., \exists z_k \in adom, \exists x_1 \in adom, ..., \exists x_l \in adom$

$$\bigwedge_i (z_i = t_i(\boldsymbol{y})) \wedge (x_i = s_i(\boldsymbol{y})) \wedge R(z_1, ..., z_k) = S(x_1, ..., x_l).$$ □

We then present the following result.

**Proposition 1**. *$\mathcal{MSO}$ has the Ramsey property over any ordered structure $\mathfrak{M} = \langle U, \Omega \rangle$.*

*Proof.* The main ideas of the proof follow the proof for first-order logic case. The proof is by induction on the formula. By Lemma 2, we can trivially transform $\mathcal{MSO}$ to that every atomic formula is an $\mathcal{MSO}(SC)$ formula or an $\mathcal{MSO}(\Omega)$ formula.

The proof is by induction on the total number of occurrences of basic atomic formulas, the Boolean connectives, quantifiers for first-order individual elements, and quantifiers for unary set variables.

The base cases for the induction are those of an $\mathcal{MSO}(SC)$, where there is no need to rearrange the formula.

By Lemma 1, any atomic $\mathcal{FO}(\Omega)$ formula has the Ramsey property.

The only issues we should consider are those of monadic second-order quantifiers over unary set variables. For the existential case, let $\varphi(\boldsymbol{x}) = \exists S \subseteq adom(D)\varphi_1(S, \boldsymbol{x})$. By the hypothesis, we find $Y \subseteq X$ and $\psi_1(S, \boldsymbol{x})$ such that for any database $D$ and $\boldsymbol{a}$ over $Y$ and some $T \subset Y$ we have $D \models \varphi_1(T, \boldsymbol{a}) \leftrightarrow \psi_1(T, \boldsymbol{a})$.

Let $\psi(x) = \exists S \subseteq adom(D)\psi_1(S, \boldsymbol{x})$. Then for any $D$ and $\boldsymbol{a}$ over $Y$, $D \models \psi(\boldsymbol{a})$ iff $D \models \psi_1(T, \boldsymbol{a})$, for some $T \subseteq adom(D)$ iff $D \models \varphi_1(T, \boldsymbol{a})$, and for some $T \subseteq adom(D)$ iff $D \models \varphi_1(\boldsymbol{a})$. □

**Corollary 2**. *The active-generic collapse holds over every structure $\mathfrak{M}$ and complex value relational signature $SC$ for monadic second-order logic. That is, every order-generic query definable in $\mathcal{MSO}_{\mathrm{act}}(SC, \mathcal{M})$ is definable in $\mathcal{MSO}_{\mathrm{act}}(SC, <)$.*

*Proof.* By Proposition 1, $\mathcal{MSO}$ has the Ramsey property over any ordered structure $\mathfrak{M} = \langle U, \Omega \rangle$. Let $Q$ be any order-generic query definable in $\mathcal{MSO}_{\mathrm{act}}(SC, \mathcal{M})$. We can find an infinite $X \subseteq U$ and an $\mathcal{MSO}_{\mathrm{act}}(SC, <)$-definable $Q1$. Now we prove that $Q$ and $Q1$ coincide everywhere in any set $Y \subseteq X$. Let $D$ be an $SC$-structure. Because $Y$ is infinite, we can find a partial monotone injective map $\pi$ from $adom(D)$ into $Y$. Since $Q1$ is $\mathcal{MSO}_{\mathrm{act}}(SC, <)$-definable and order-generic, and thus co-domains of $Q$ and $Q1$ are within $adom(D)$. Hence, $\pi(Q(D)) = Q(\pi(D)) = Q1(\pi(D)) = \pi(Q1(D))$, from which $Q(D) = Q1(D)$ follows. By the Ramsey property, there exists an infinite subset $X \subseteq U$, where $U$ is a universe domain, and an $\mathcal{MSO}_{\mathrm{act}}(SC, <)$-definable $Q1$ that coincides with $Q$ on $X$. □

We now turn to consider natural semantics queries which are the main realms of constraint databases in the context of geometric setting. Unfortunately, the natural-active collapse fails for the second-order logic over the real field[8]. To overcome this problem, we investigate a fragment of $\mathcal{MSO}(SC, \mathfrak{M})$ in which all second-order variables are produced by $\mathcal{FO}$ formulas by induction. We denote this fragment as $\tilde{\mathcal{MSO}}$.

We will show that natural-active collapse holds for some fragment of $\mathcal{MSO}$. We first recall that the first-order logic admits natural-active collapse as follows.

**Definition 11**. *A structure is called complete if the set of first-order true sentences in the structure is effectively decidable.*

FACT 2 (Natural-Active Collapse)[5,15,28]. *Let $\mathfrak{M}$ be an o-minimal structure with dense order that admits quantifier elimination. Let $\varphi$ be an arbitrary $\mathcal{FO}(SC, \Omega)$ query, and then there exists an equivalent $\mathcal{FO}_{\text{act}}(SC, \Omega)$ active-semantics query $\varphi_{safe}$. Moreover, if the first-order theory of $\mathfrak{M}$ is complete and quantifier elimination procedure is effective, then $\varphi \rightarrow \varphi_{\text{safe}}$ is effective. The above states the fact: $\mathcal{FO}(SC, \mathfrak{M}) = \mathcal{FO}_{\text{act}}(SC, \mathfrak{M})$ for any relational schema $SC$.*

We then present the following natural-active collapse theorem for the $\tilde{\mathcal{MSO}}$ language.

**Theorem 3**. *Let $\mathfrak{M} = (\mathscr{U}, \Omega)$ be an o-minimal structure with dense order that admits quantifier elimination. Then it admits a weaker form of the natural-active collapse in the setting of embedded finite CV model, i.e., $\tilde{\mathcal{MSO}}(SC, \mathfrak{M}) = \mathcal{MSO}_{\text{act}}(SC, \mathfrak{M})$ for any CV-schema $SC$.*

*Proof.* Suppose the input formula is $\varphi$. In the monadic second-order logic, we need to consider an subformula $\exists S\alpha(\boldsymbol{x})$ case in addition to all cases in the first-order logic, where $S$ is a set-value variable not in $SC$, and it can be produced by $\mathcal{FO}$ formulas by induction.

Given $\varphi(X) = \exists S\ \bar{\alpha}(X, S)$, where $X$ is a complex-value variable and $S$ is a set-value variable. Let $\alpha(X, S)$ be a formula equivalent to $\bar{\alpha}_{\text{act}}$, which is of the form

$$\Phi\mathcal{Y}_1 \in adom, ..., \Phi\mathcal{Y}_n \in adom\gamma(X, \mathcal{Y}, Z),$$

where $\Phi$ is a quantifier symbol $\exists$ or $\forall$, $\mathcal{Y}$ is an atomic or set-value variables in $\mathcal{MSO}$, and $\gamma$ is a quantifier-free formula with the following key properties: every atomic sub-formula of $\gamma$ is an $L(\mathfrak{M})$ formula or an $L(SC)$ formula, and $SC$ is a CV-schema.

Those $\mathcal{FO}$ formulas can be expressed by $\mathcal{FO}_{\text{act}}(SC, \mathcal{M})$ formulas based on o-minimal structure property and FACT 1. Therefore there is an $\mathcal{MSO}_{\text{act}}(SC, \mathfrak{M})$ formula which is equivalent to $\varphi$. □

### 4.2 Constraint CV Databases

We now study the expressive power of constraint CV query languages such as $\mathcal{CALC}^{\text{cv}}$ and $\mathcal{MSO}$.

**Theorem 4**. *The active-generic collapse holds over every structure $\mathfrak{M}$ for complex value logic including $\mathcal{CALC}^{\text{cv}}$ and $\mathcal{MSO}$. That is, every order-generic query definable in $\mathcal{SO}_{\text{act}}(SC, \mathfrak{M})$ is definable in $\mathcal{SO}_{\text{act}}(SC)$.*

*Proof.* $\mathcal{CALC}^{\text{cv}}$ is a many-sorted calculus which is formed from a standard first-order logic. However, it facilitates *set* variables and has a second-order flavor. The key features, second-order quantification and existential and subset predicates are needed to be reviewed carefully in regards to the active generic collapse property. Let $\varphi(\boldsymbol{x}, \boldsymbol{X})$ be a formula in $\mathcal{SO}_{\text{act}}(SC, \mathfrak{M})$ with free first-order variables $\boldsymbol{x}$ and second order variables $\boldsymbol{X}$. Given an infinite $X \subseteq \mathscr{U}$, by Ramsey theorem, we can find an infinite $Y \subseteq X$ and a $\mathcal{SO}_{\text{act}}(SC, <)$ formula $\chi(\boldsymbol{x}, \boldsymbol{X})$ such that for any database $D \in Inst(SC, X)$ it is the case that $D \models \varphi(\boldsymbol{a}, B_i) \leftrightarrow \chi(\boldsymbol{a}, B_i)$ for all $\boldsymbol{a}$ is a tuple of elements of $adom(D)$ and $B_i \subseteq adom(D)^k$, where $B_i$ is a $k$-ary relation that belongs to one of $X$. By induction, we can conclude that for every active-generic query definable in $\mathcal{SO}_{\text{act}}(SC, \mathfrak{M})$ is definable in $\mathcal{SO}_{\text{act}}(SC)$. □

In the standard constraint databases, we sometimes want to write queries against a linear constraint input database in $\mathcal{FO}$ + POLY. It is known that $\mathcal{FO}$ + POLY has more expressive power than $\mathcal{FO}$ + LIN although $\mathcal{FO}$+ POLY has more costly evaluation procedures. One sometimes may want to use $\mathcal{FO}$ + POLY to write queries against semi-linear sets. Similarly, we may want to write queries against a linear constraint input database or a polynomial constraint database in a more expressive higher order query language.

The fundamental topological connectivity property is important in many applications of spatial databases. As standard query languages for constraint databases lack the power to express connectivity properties[9], researchers attempted to enrich query languages by adding some extra functions, like transitive closure or fixpoint operators. However, this extension may cause closure property to fail for the extended languages. In [29], authors added topological connectivity property to the first-order constraint query languages and obtained new languages which are closed. However, this extension may cause query evaluation at high cost. The alternative approach is to adopt a higher-order query language to overcome this deficiency.

**Definition 12**. *Given a structure $\mathfrak{M} = \langle U, \Omega \rangle$, a set of complex values $O \subseteq U^\tau$ with complex value sort $\tau$ is called $\mathfrak{M}$-definable if there exists a formula $\varphi(x)$ in the language of $\mathcal{M}$ such that $O = \{o \in U^\tau \mid SO(SC, \mathfrak{M}) \models \varphi(o)\}$.*

We show that connectivity is definable in $\mathcal{MSO}(SC, \mathfrak{M})$ where the input database $SC$ contains just an $\mathfrak{M}$-definable set $S \subseteq \mathcal{R}^k$.

**Proposition 2**. *Some fundamental topological queries such as majority, connectivity, transitive closure, acyclicity, having exactly one hole and having exactly k connected components, are definable in* $\mathcal{SO}(SC, \mathfrak{M})$.

*Proof.* It is well known that every computable query over $Inst(SC, \mathcal{N})$ is definable in $\mathcal{FO}(SC, \mathcal{N})$. Accordingly, it is also definable in $\mathcal{FO}(SC, \mathcal{R})$ and $\mathcal{SO}(SC, \mathcal{R})$. As transitive closure can be expressed by $\mathcal{SO}$[20], by combining the above two properties we can imply that topological queries such as majority, connectivity, transitive closure, acyclicity, having exactly one hole and having exactly $k$ connected components, can be definable in $\mathcal{SO}(SC, \mathfrak{M})$ for any ordered structure $\mathfrak{M}$. □

**Theorem 5**. Datalog$^{cv,\neg}$ + Poly *is closed, that is, on an* $\mathfrak{M}$*-definable complex value constraint databases, an Datalog*$^{cv}$ + Poly *query produces an* $\mathfrak{M}$*-definable complex value set.*

*Proof.* A query is expressible in Datalog$^{cv}$ with stratified negation if and only if it is expressible in $\mathcal{MSO}$[16]. We know that $\mathcal{SO}(SC, \mathfrak{M})$ is closed under Boolean connectives. As $\mathcal{MSO} \subset \mathcal{SO}$, $\mathcal{MSO}(SC, \mathcal{M})$ is also closed under Boolean connectives. Therefore $\mathcal{MSO}(SC, \mathfrak{M})$ is closed for any input $\mathfrak{M}$-definable complex value constraint databases. Accordingly, Datalog$^{cv,\neg}$ + Poly is closed on an $\mathfrak{M}$-definable complex value constraint databases. □

## 5 Complexity of $\mathcal{MSO}$ on Constraint Databases

It is normal practice to investigate the complexity of various constraint query languages when we explore the expressive power of logics expressing these languages. The goal of this section is to study the complexity of queries expressible in $\mathcal{SO}$ logic and its fragments over constraint CV-databases. We want to understand how a logic formalism, for example, $\mathcal{MSO}$ is related to computational complexity classes and conduct a comprehensive investigation in order to get insight of knowledge about complexity theory in the context of constraint queries.

Computational complexity theory is concerned with issues of computational efficiency — determining the amount of computational resources such as time, space, or hardware resources that are required to carry out a given task. We first briefly review a computational model called the Boolean circuit and some complexity classes which are adopted in this section. The detailed description can be found in textbooks[20,30].

**Definition 13**. *A single Boolean circuit* $\mathcal{C}$ *is a directed acyclic graph with k inputs and one output, where* $k \in \mathbb{N}$. *All internal vertices are labeled with one of* $\vee$, $\wedge$ *or* $\neg$. *Then for any Boolean formula* $\varphi$ *in the language* $\mathcal{L} \subseteq \{0,1\}^*$, *the output of* $\mathcal{C}$ *is defined by normal logical inference. We say a circuit* $\mathcal{C}$ *accepts an input string* $l \in \mathcal{L}$ *iff* $\mathcal{C}$ *outputs* 1. *We denote the size of each circuit* $\mathcal{C}$ *as* $\mid \mathcal{C} \mid$ *which is the number of vertices in the graph.*

**Definition 14**. *Let* $f$ *be a function from* $\mathbb{N}$ *to* $\mathbb{N}$. *A family of circuits* $\mathcal{C}_n$ *is said to be* $f(n)$*-size if each circuit of* $\mathcal{C}_n$ *has n inputs and a single output, and its size is at most* $f(n)$, *i.e.,* $\mid \mathcal{C}_n \mid \leqslant f(n)$ *for every n.*

Then we give definitions for $\boldsymbol{P}_{/\text{poly}}$, $\boldsymbol{NC}$ and $\boldsymbol{AC}$ classes as follows. These definitions are adopted from the literature and books[2,30].

**Definition 15**. *The class of languages that are decidable by a family of polynomial-sized circuits is denoted as* $\boldsymbol{P}_{/\text{poly}}$.

The depth of a circuit is the longest length from an input node to the root of the circuit graph. We now give the formal definitions of two important complexity classes which play an important role in the computational complexity theory related to parallel computation.

**Definition 16**. *For every d, we classify a language* $\mathcal{L}$ *in a specific class if* $\mathcal{L} \in \boldsymbol{P}_{/\text{poly}}$ *with depth* $O(\log^d n)$. $\boldsymbol{P}_{/\text{poly}} = U_c SIZE(n^c)$, *where n and c are natural numbers. We denote this class as* $\boldsymbol{NC}^d$, *where n is the number of inputs of a sequence of* $\{C_n\}_{n\in\mathbb{N}}$ *of Boolean circuits. The class* $\boldsymbol{NC}$ *is* $\bigcup_{i\geqslant 1} \boldsymbol{NC}^i$.

**Definition 17**. *The class* $\boldsymbol{AC}^i$ *is defined similar to* $\boldsymbol{NC}^i$ *except a difference of internal vertices that can be allowed to have unbounded fan-in.*

In the context of standard constraint databases, it has been shown that the complexity upper bounds for $\mathcal{FO}(<)$, $\mathcal{FO}$ + Lin, and $\mathcal{FO}$ + Poly are $\boldsymbol{AC}^0$, $\boldsymbol{NC}^1$, and $\boldsymbol{NC}$ respectively[2,31,32].

We now review the polynomial hierarchy complexity class as follows.

**Definition 18**. *Given a language* $\mathcal{L}$ *and a polynomial function q, if there exists a polynomial-time Turning machine T such that*

$$l \in \mathcal{L} \text{ iff } Q_1 Q_2 ... Q_i T(l, x_1, ...x_i) = 1,$$

*where* $Q_i$ *denotes* $\exists x_i \in \{0,1\}^{|p|}$ *or* $\forall x_i \in \{0,1\}^{|q|}$, *depending on whether i is even or odd, respectively, then we classify this class as a polynomial hierarchy, denoted as* $\boldsymbol{PH} = \bigcup_i \Sigma_i^p$.

We now investigate data complexity of higher-order logics in the context of constraint CV-databases. We

start with the general definition of the size of a constraint CV-database with semi-algebraic sets as its components.

The main results are that the complexity upper bounds for $\mathcal{MSO}(<)$, $\mathcal{MSO}$ + Lin and $\mathcal{MSO}$ + Poly are $\cup_i \sum_i^{\boldsymbol{AC}^0}$, $\cup_i \sum_i^{\boldsymbol{NC}^1}$, and $\cup_i \sum_i^{\boldsymbol{NC}}$, respectively.

We first define the size of a semi-linear set.

**Definition 19**. *An $\mathcal{MSO}$ + Lin relation $R$ of arity $k$ is said to be of bound $n$, if $R$ can be represented as*

$$\bigvee_{i\in I}\bigwedge_{j\in J}(a_{ij}^k x_k + \cdots + a_{ij}^1 x)\Theta c_{ij},$$

*where*

- *each $\Theta_{ij}$ is one of $=$ and $<$;*
- *the number of disjuncts is at most $n$;*
- *each disjunct has at most $|J_i| \leqslant n$ conjuncts;*
- *the coefficients $a_{ij}^l$ and $c_{i,j}$ are integers with absolute value less than $2^n$.*

It is well known that $\mathcal{MSO}$ logic is closely connected to PH. In the following we show that $\mathcal{MSO}$ + Lin over constraint CV-databases has $\cup_i \sum_i^{\boldsymbol{NC}}$ data complexity.

**Theorem 6**. *Let $\mathfrak{M} = <\mathscr{U}, \Omega>$ be an o-minimal structure. $\mathcal{MSO}$ + Lin has $\cup_i \sum_i^{\boldsymbol{NC}^1}$ data complexity over $\mathfrak{M}$.*

*Proof.* We know that $\mathcal{FO}$ + Lin in $\boldsymbol{NC}^1$ and $\mathcal{MSO}$ can express each level of $\Sigma_i^p$ or $\Pi_i^p$. Let $Q$ be an $\mathcal{MSO}$ Boolean query whose input consists of a single CV-database. For each $n$, we construct a Boolean circuit $\mathcal{C}_\setminus$ of depth $d$ and a polynomial amount of size. To simulate $Q$, we express it in a constraint algebra form. Recall that constraint algebra has been clearly described in Section 3 which consists of tuple and set constructor, destroy, and power-set operators in addition to the basic first-order algebra operators. The operator with the highest complexity is powerset only. It is well known that the data complexity of powerset is $EXP$. $EXP = U_{c\geqslant 1} DTIME(2^{n^c})$, which is the exponential time analogy of $P$[30]. $EXP \in \Sigma_2^{\boldsymbol{NC}}$. Therefore, $Q$ can be implemented by constructing a Boolean circuit and obtain data complexity of $\Sigma_i^{\boldsymbol{NC}}$, where $i$ depends on the number of quantifiers. □

Note that the data complexity of $\mathcal{FO}$ with linear constraints is in PTIME (polynomial time) and the quantifier elimination procedure is much less expensive than the case of semi-algebraic one. It has been shown that the full formalism of computational complexity of linear constraints over the integers in temporal reasoning systems is NP-hard[33].

**Theorem 7**. *Under both the arithmetic and the bit model, $\mathcal{MSO}$ + POLY has $\mathbf{NCH} = \bigcup_i \sum_i^{\boldsymbol{NC}}$ data complexity.*

*Proof.* $\mathcal{FO}$ + POLY has complexity $\boldsymbol{NC}^{[2]}$. Similar to Theorem 6, a query in $\mathcal{MSO}$ + Poly can be implemented by constructing a Boolean circuit. Therefore $\mathcal{MSO}$ + POLY has $\mathbf{NCH} = \bigcup_i \sum_i^{\boldsymbol{NC}}$ data complexity. □

We present the following observation.

**Theorem 8**. *Every generic query in Inflationary Datalog$_{\text{act}}^{\text{cv},\neg}(SC, \mathfrak{M})$ without the powerset operator can be evaluated in polynomial hierarchy time given that every generic sentence in $\mathfrak{M}$ can be evaluated in $AC^0$/poly.*

*Proof.* All operations in $\mathcal{MSO}$ and accordingly in Datalog$_{\text{act}}^{\text{cv},\neg}$ can be computed in polynomial-time in the size of their arguments except for power-set which takes exponential time. □

## 6 Query Safety with Constraints

In Sections 4 and 5, we proposed several paradigms for querying constraint databases with complex objects. As in the classical notion of standard relational databases, we normally require that queries return finite outputs on finite inputs. This class of queries is called safe queries[16,34–37].

We investigate whether or not the classical safety notion for $\mathcal{FO}$ formulas with underlying interpreted functions structure carries over smoothly to second-order formulas in the constraint setting.

We first investigate the embedded finite complex value model and then try to reduce most of the results to the infinite case. For the finite case, we consider whether or not there are any new well-behaved underlying structures, and/or new assumptions required, for the formula on which safe queries can be syntactically characterized. We develop a procedure that takes as input an $\mathcal{CALC}^{\text{cv}}(SC, \mathfrak{M})$ formula and determines if it is safe or not. Recall that $\mathcal{CALC}^{\text{cv}}$ is a subset of $\mathcal{SO}$. When we talk about finite CV-databases with constraints we adopt the logic language — $\mathcal{CALC}^{\text{cv}}$. On the other hand, we investigate $\mathcal{SO}$ over general infinite CV-databases in constraint settings. Then we apply rang-restriction concept for $\mathcal{CALC}^{\text{cv}}$ queries and show that for any o-minimal structure with dense order, the class of range-restricted queries is the same as the class of safe active-semantics complex value queries, $\mathcal{CALC}^{\text{cv}}$. In Subsection 6.1.3, we will discuss the Dichotomy theorem for $\mathcal{SO}$.

For the second part of problem of query safety over infinite objects, we investigate whether $\tilde{\mathcal{SO}}$ posses "tame" behavior in the complex object model. Informally, the notion of tame behavior of queries is that

their output is of the same sort as the input. It can also be defined as whether or not a query preserves some geometric property[8]. It is also called closure property. A more detailed involved definition of tame behavior will be presented later.

### 6.1 Safe Constraint $\mathcal{CALC}^{\mathrm{cv}}$ Queries: Finite Case

We first consider safe queries in the embedded finite model. In (monadic) second-order logic, some set variables are quantified over subsets of the universal domain as noted in Section 3. A query is an expression $\{x \mid \varphi(x)\}$, where the formula $\varphi$ has exactly one free variable $x$ with sort $\tau_x$. Suppose that we have a $\mathcal{CALC}^{\mathrm{cv}}$ query $\varphi(x)$ over some underlying structure $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$ and a CV $SC$-database $D$. The result of $\varphi$ on $D$ is $\varphi_{\mathcal{M}}(D) \stackrel{\mathrm{def}}{=} \{a \in U^{\tau} \mid D \models \varphi(a)\}$, where $U^{\tau}$ denotes the set of complex value constants with the corresponding sort $\tau$.

We formally define safe queries as follows.

**Definition 20**. *A $\mathcal{CALC}^{\mathrm{cv}}(SC, \mathfrak{M})$ formula $\varphi(x)$ is safe for a finite complex value $SC$-structure $D$ over an underlying structure $\mathfrak{M}$ if $\varphi_{\mathcal{M}}(D)$ is finite. A query is safe if it is safe for every finite complex value database.*

#### 6.1.1 Safe Decidability Algorithm

It is well known that the safety problem is undecidable even for a simple structure $\mathfrak{M} = \langle \mathscr{U}, \emptyset \rangle$, and a simple formula, $\varphi$, an $\mathcal{FO}_{\mathrm{act}}(SC)$ formula. However, as in the first-order logic, we can find a procedure to identify a recursive subset of safe formulas that capture the query class that returns finite outputs with appropriate sort structures.

We now turn to the development of a syntactic condition that ensures $\mathcal{CALC}^{\mathrm{cv}}$ formulas with constraints to be evaluable in a closed-form. Our general approach to checking the safety property is to check whether each input constraint sub-formula contains not-allowed negative constraints. For example, a negative constraint in $\exists y R(y) \wedge \neg(x^2 = y)$ will cause the formula output an infinite set of $x$ values.

Intuitively, if a query is in a variable-bounded form, then we can find an algebraic formula(s) which plays the role of providing an upper bound for the output of the query[34]. That means each $\mathcal{FO}$ variable is an individual value that belongs to a bounded range, i.e., $x \in S$, where $S$ is an upper bound of the range of $x$ values, and each set variable of $\mathcal{SO}$ contains a finite set of values which are from some bounded set.

As in [20], we develop an algorithm, Algorithm 1, which can identify such a set of bounded variables of a formula. In the algorithm, we need to consider three key points: 1) $(t \in S) \wedge \phi$, 2) $(R \subseteq S) \wedge \phi$, and 3) $\neg C$, where $C$ is a constraint. In order to make the formula safe, the variables in the above three key points should be bounded.

**Algorithm 1**. Bounded-Variables ($bd$)

Input: a constraint $\mathcal{CALC}^{\mathrm{cv}}(SC, \Omega)$ formula $\varphi(x)$
Output: a subset of the free variables of sub-formulas in $\varphi$ or $\perp$

**begin**
($pred$ is a predicate in $\{SC, =, \in, \subseteq\}$;
$f$ is a function of $\Omega$;
$R(t)$ is an atomic formula)
**if** for some parameterized query $\{z \mid \psi\}$ occurring as a term in $\varphi$, $z \notin bd(\psi)$ **then return** $\perp$
  **case** $\varphi$ **of**

| | |
|---|---|
| $R(t)$ | : $bd(\varphi) = free(t)$ |
| $(t\ pred\ t') \wedge \psi$ | : **if** $\psi$ is safe and $free(t') \subseteq free(\psi)$ **then** $bd(\varphi) = free(t) \cup free(\psi)$ |
| $t\ pred\ t'$ | : **if** $free(t') = bd(t')$ **then** $bd(\varphi) = free(t') \cup free(t)$ **else** $bd(\varphi) = \emptyset$ |
| $\psi_1 \wedge \psi_2$ | : $bd(\varphi) = bd(\psi_1) \cup bd(\psi_2)$ |
| $\neg\psi_1$ | : **then** $bd(\varphi) = \emptyset$ |
| $\neg\psi_1$ | : **if** $\psi_1 \equiv x \subseteq y$ or $\psi_1 \equiv x \subseteq C$ **then** $\perp$ |
| $\exists x \psi_1$ | : **if** $x \in bd(\psi_1)$ **then** $bd(\varphi) = bd(\psi_1) - \{x\}$ **else return** $\perp$ |
| $y = f(x)$ | if $x \in bd(\varphi)$ **then** $bd(\varphi) \cup \{y\}$ |
| $\neg(y = f(x))$ | if $y \notin bd(\varphi)$ **then return** $\perp$ |

**end**

Note: We replace the dis-junction of the two formulas $\varphi_1 \vee \varphi_2$ by the equivalent $\neg\varphi_1 \wedge \neg\varphi_2$ before we apply the above algorithm.

Intuitively if every variable is bounded, in the sense that it is restricted by the formula to lie in the active domain, we call the formula is a safe range. We identify the set of bounded variables of a formula using the following algorithm which returns such safe range variables or returns the symbol $\perp$, which indicates that some quantified variable is not bounded or some constraint sub-formula does not lie in the active domain.

We say that a formula is in safe range form if $bd(\varphi) = free(\varphi)$. We consider a query in the form of $Q \equiv \{y \mid \psi(y)\}$ as a legal term which can occur in $\mathcal{CALC}^{\mathrm{cv}}$ formulas like any other term.

*Example* 11. Let $\tau_p = \tau_q = (\{R\}, R)$.

$$p(x, y) \stackrel{\mathrm{def}}{=} (x = \{x_1 \mid \phi\} \wedge y = 10),$$
$$\text{where } \phi = (0 \leqslant x_1 \leqslant 10).$$

Consider the formula

$$\psi = p(x, y) \wedge \exists t(u = x \wedge \neg q(u, t) \wedge t \in u).$$

Then $bd(\psi) = free(\psi) = \{x, y, u\}$. Therefore $\psi$ is a safe formula.

*Example* 12. Let $\tau_p = (\{R\}, R)$; $\tau_s = (\{R\}, \{R\})$. Consider the formula

$\varphi = \exists y(\neg p(x, y) \wedge s(y, z))$,

where

$p(x, y) \stackrel{\text{def}}{=} (x = \{x_1 \mid \phi\} \wedge y = 10)$, $\phi = (0 \leqslant x_1 \leqslant 10)$; $s(y, z) \stackrel{\text{def}}{=} y \subset z \wedge z = \{1, 2, 3\}$.

By Algorithm 1, $\neg p(x, y)$ is not a safe formula as it contains negation. $bd(\neg p(x, y)) = \emptyset$. $s$ contains $y \subset z$ and $z = \{1, 2, 3\}$ which is a safe formula. By Algorithm 1, $bd(\varphi) = \{z\} \neq free(\varphi) = (x, z)$. Therefore $\varphi$ is not a safe formula.

*Example* 13. Let $\tau_p = \tau_q = (\{R\}, R)$. $p(x, y) \stackrel{\text{def}}{=} (x = \{x_1 \mid \phi\} \wedge y = 10)$, where $\phi \equiv (0 \leqslant x_1 \leqslant 10) \wedge x_1^2 = y$. Consider the formula

$$\psi = p(x, y) \wedge \exists t(u = x \wedge \neg q(u, t) \wedge t \in u).$$

$free(\psi) = \{x, y, u\}$ and all variables are bounded. Therefore $\psi$ is a safe formula.

### 6.2 Safe Translation

Following the line of classical well-developed theory for standard $\mathcal{FO}$ queries with interpreted functions on finite case, we first start to investigate the $\mathcal{SO}$ query safety in the finite case. We try to find under what assumption on the underlying structure, $\mathcal{SO}$ queries have the property of providing effective safe translation from arbitrary queries into safe ones. At first glance, there seem no insight findings from the extension from $\mathcal{FO}$ logic. However, we proceed to investigate for what kind of well-behaved structures, safety property can be decidable and how we can find a syntactically defined class which captures such a property and is complete in this respect. We define a general idea about safe translation which can convert an arbitrary $\mathcal{SO}$ query into a safe one and produce the same finite results if the output of this query is finite.

**Definition 21**. *If there is a translation of active-semantics $\mathcal{SO}$ (including $\mathcal{FO}$) query $\varphi$ over some given structure $\mathfrak{M}$ and input finite database $D$ into a formula $\phi$, i.e., a function $\varphi \to \phi$, such that the following conditions hold:*

- *the output of $\phi$ is finite,*
- *if $\varphi(D)$ is finite, then $\varphi(D) = \phi(D)$,*

*then we say that there is a safe translation for constraint query. A translation is canonical if $\phi(D) = \emptyset$ whenever $\varphi$ cannot halt to generate a finite output.*

It is well-known that there exists no such safe translation of arbitrary active-semantics queries over a well-behaved structure, for example, a Turing machine with appropriate coding which consists of disjoint inputs and trace[3]. It does not exist even over natural number domain, for example, $\mathfrak{M} = \langle \mathbb{N}, \Omega \rangle$[8].

However, we find that not only o-minimality and quantifier elimination properties, like standard constraint databases, but also the restriction of second-order set variables, play a crucial role for the safety issue relevant to most applications of constraint CV-databases. This role makes safe translation do exist for $\tilde{\mathcal{SO}}$ queries. Recall that the set variables of $\tilde{\mathcal{SO}}$ range over a restricted domain which is generated from all first-order variables in the formula and the active domain *adom*.

**Proposition 3**. *Let $\mathfrak{M}$ be an o-minimal structure with a dense order and it admits effective quantifier elimination. $\mathfrak{M}$ also has a decidable theory. Then there exists a recursive canonical safe translation of the extended active-domain $\tilde{\mathcal{SO}}$ formulas over $\mathfrak{M}$.*

*Proof.* Let $\varphi(x)$ be a $\tilde{\mathcal{SO}}$ formula over $\mathfrak{M}$ and a given database $D$. The output is a single variable $x$ with complex value sort $\tau_x$. We denote the active domain of the output of $\varphi$ as $adom(\varphi(D))$. We assume $\alpha(y)$ be a $\mathcal{FO}$ formula defining the active domain of the output of the fragment of $\varphi$, which consists of first-order variables and constants only. Applying natural-active collapse theorem from the $\mathcal{FO}$ theory, we can provide the following active-semantics sentence $\Theta$ equivalent to

$$\forall y_1, y_2.((y_1 < y_2) \wedge \neg y(\forall y, y_1 < y < y_2 \to \alpha(y))). \quad (1)$$

The above formula guarantees that $\alpha(y)$ cannot be infinite points. It is the same concept with "non-continuous" in the real field of mathematics. And we let $\chi$ be the fragment of $\varphi$ which consists of second-order variables, i.e., variables of the subset of individual elements of universal domain, and quantifiers over such set variables. $\beta(z)$ defines the active domain of the output of $\chi$. As these set variables are produced from some $\mathcal{FO}$ formula, we can find the following active-semantics sentence $\Upsilon$ equivalent to

$$\neg z_1, z_2((z_1 < z_2) \wedge \forall z, Z, (z \in Z, z_1 < z < z_2 \to \beta(z))), \quad (2)$$

where $Z$ is a set variable in $\varphi$.

Then translated safe formula $\phi$ can be defined as $\varphi \wedge \Theta \wedge \Upsilon$. It can be easily verified that $\varphi \wedge \Theta \wedge \Upsilon$ produces finite output since $D \models \Theta \wedge \Upsilon$ iff $\varphi(D)$ is finite.

We now prove that $\varphi(D)$ is finite. Assume that either $D \models \neg\Theta$ or $D \models \neg\Upsilon$. In both cases, $\varphi(D)$ is infinite because of the dense property of the structure.

We then look at all occurrences of $SC$ predicates in $\alpha$ and $\beta$ and replace each of them with a disjunctive norm form. The results generate $\alpha'(y)$ and $\beta'(z)$ in the language of $\Omega$ and constants from the extended active domain. Further $D \models \alpha(a)$ iff $\mathfrak{M} \models \alpha'(y)$; $D \models \beta(z)$ iff $\mathfrak{M} \models \beta'(z)$. Let $\Theta'$ and $\Upsilon'$ be obtained from $\Theta$ and $\Upsilon$ respectively by substituting $\alpha'$ for $\alpha$ and $\beta'$ for $\beta$. We then have $\mathfrak{M} \models \Theta'$; $\mathfrak{M} \models \Upsilon'$, since $\alpha'(\mathfrak{M}) = \{a \mid \mathfrak{M} \models \alpha'(a)\}$ is a union of finite intervals and finite points.

Similarly, $\beta'(\mathfrak{M}) = \{b \mid \mathfrak{M} \models \beta'(b)\}$ contains a union of finite intervals, finite points and finite sets. (1) and (2) guarantee that there is no "continuous" interval in the field $\boldsymbol{R}$. Thus, from o-minimality property, we get $\alpha'$ and $\beta'$ which contain a finite union of points or a union of finite sets. This concludes that $\varphi(D)$ is finite. □

**Corollary 3**. *Let $\varphi$ be a $\mathcal{SO}$ query over structure $\boldsymbol{R}$ or $\boldsymbol{R}_{Lin}$, and let $D$ be a complex object database. Then it is decidable if $\varphi(D)$ is finite.*

*Proof.* In the above proof, we demonstrate that the active-semantics $\Theta \wedge \Upsilon$ tests whether $\varphi(D)$ is finite. □

### 6.2.1 Range-Restriction Theorem

We now show a more involved notion, range-restriction, which demonstrates concrete characterization of safe queries. This notion plays an important role for exploring the query safety issue. We first informally describe the concept of range-restriction for finite CV-databases over interpreted structures. Then we show that the range-restriction theorem and the Dichotomy theorem for standard $\mathcal{FO}$ constraint queries carry over to $\mathcal{CALC}^{\mathrm{cv}}$ queries on finite CV-databases.

We illustrate a simple example to introduce the range-restriction notion.

*Example* 14. Consider a $\mathcal{CALC}^{\mathrm{cv}}$ query $\varphi(x)$ over a relation $R$ with sort $(A : \boldsymbol{dom}, B : \boldsymbol{dom})$ which produces results with sort $(dom, \{\})$.

$$\varphi(x) = \{x \mid x = (z, \{y \mid R(z, y)\}) \wedge \exists y'(R(z, y')) \wedge (10 < z + (y')^2 < 100)\}.$$

This query defines the nesting of the second column $B$ into a set, and satisfies the condition that $A$ column value plus the square of one value from $B$ column should be greater than 10 and less than 100. The relation $R$ restricts the range of variables $z$ and $y$. The parameterized query $x = (z, \{y \mid R(z, y)\})$ can be used safely, i.e., its output within a finite scope of CV values. Similarly, $\exists y'(R(z, y'))$ is also range-restricted. Thus this query is in a range-restriction form. Every element in the output is a solution of $R(z, y)$ and $10 < z + y^2 < 100$. This is a classical notion of range-restriction and both $(\exists y'(R(z, y')))$ and $(10 < z + (y')^2 < 100)$ provide an upper bound on the output.

It is well known that the safety problem is undecidable even for a simple structure $\mathfrak{M} = (U, \emptyset)$, and a simple formula, $\varphi$, an $\mathcal{FO}_{\mathrm{act}}(SC)$ formula[34,38].

However, as in the first-order logic, we can find a procedure to identify a recursive subset of safe formulas that capture the query class that returns finite complex value outputs and every formula with this property belongs to this query class.

Intuitively, if a query is in range-restricted form, then we can find algebraic formulas $\gamma$ which play the role of providing an upper bound for the output of the query[34]. That means each set variable is range-restricted, i.e., the values assigned to set variables are finite.

We first define the notion of algebraic set.

**Definition 22**. *Let $\gamma(x; \boldsymbol{y})$ be a $\mathcal{CALC}^{\mathrm{cv}}$ formula and $\boldsymbol{y}$ a vector of parameter variables with associated sorts $\tau_{y_i}$. We call the formula $\gamma$ as CV-algebraic if for each complex value $b_i \in \mathscr{U}^{\tau_{b_i}}$, the set $\{a \in \mathscr{U}^{\tau_x} \mid \mathfrak{M} \models \gamma(a; \boldsymbol{b})\}$ is finite.*

Note that $a$ is also a complex value.

Let a database $D$ over a structure $\mathfrak{M} = \langle U, \Omega \rangle$ and a $\mathcal{CALC}^{\mathrm{cv}}$ formula $\gamma(x; \boldsymbol{y})$ in the language of $\Omega$ and schema $SC$ of $D$.

$$\gamma(D) = \{a \in \mathscr{U}^{\tau_x} \mid \exists \boldsymbol{b} \in adom(D)^{\tau_{b_i}} \Rightarrow \mathfrak{M} \models \gamma(a; \boldsymbol{b})\}.$$

If $\mathcal{F}$ is a collection of formulas with variables of $x$; $\boldsymbol{y}$, then

$$\mathcal{F}(D) = \bigcup_{\gamma \in \mathcal{F}} \gamma(D).$$

If $\mathcal{F}$ is finite and each formula in $\mathcal{F}$ is CV-algebraic, then $\mathcal{F}(D)$ is finite.

**Definition 23**. *Let $D$ be a database with schema $SC$ over an interpreted structure $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$. A range-restricted $\mathcal{CALC}^{\mathrm{cv}}$ query is a pair $\mathcal{Q} = (\mathcal{F}, \varphi(x))$ where $\mathcal{F}$ is a finite collection $\{\gamma_1(x; \boldsymbol{y}_1), ..., \gamma_m(x; \boldsymbol{y}_m)\}$ of CV-algebraic formulas in the language of $\Omega$, and $\varphi(x)$ is a $\mathcal{CALC}^{\mathrm{cv}}(SC, \Omega)$ query. The semantics of $\mathcal{Q}$ is defined as follows*:

$$\mathcal{Q}(D) = \{a \in \mathscr{U}^{\tau_x}, a \in \mathcal{F}(D) \mid D \models \varphi(a)\}.$$

The above collection of formulas $\mathcal{F}$ provides an upper bound on the output of query $\varphi$.

*Example* 15. Consider a $\mathcal{CALC}^{\text{cv}}$ query $\varphi(x)$ over a relation $R$ with sort $((z : \boldsymbol{dom}, S : \{\})$ which produces results with sort $(dom, \{\})$.

$$\varphi(x) = \{x \mid x = (z, \{y \mid R(z, y)\}) \wedge \exists y'(R(z, y')) \wedge y > 100\}.$$

Clearly, it is not a safe formula as $y > 100$ will generate infinite $y$ values. Now if we set $\gamma(z; y) \equiv (z^2+y^2 = 100)$ and $\mathcal{Q} = (\{\gamma\}, \varphi)$, then for any database $D$, $Q(D)$ is the set of CV $x$ values such that $(z^2 + y^2 = 100)$ and either $y, z \in R$ or $y > 100$. It is clear that $Q(D)$ is a finite set.

**Theorem 9**. *For any o-minimal structure $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$ based on a dense order, there is a function that takes an active-domain formula $\varphi$ as input, and outputs a range-restricted active query $Q = (\mathcal{F}, \varphi)$, where $\mathcal{F}$ is a collection of formulas in $x$, $\boldsymbol{y}$. If $\mathcal{F}$ is finite and CV-algebraic, then $\mathcal{F}(D)$ is finite.*

*Proof.* We consider the one-variable case here. The extension to multi-variables case is similar by using the natural-active collapse. We assume

$$\varphi(x) \equiv Q_1 y_1 \in adom, ..., Q_l y_l \in adom \Phi(x, \boldsymbol{y}),$$

where each $Q_i$ is $\exists$ or $\forall$, and $y_i$ is an individual element variable or a set variable. $\Phi(x, \boldsymbol{y})$ is quantifier-free and all atomic sub-formulas $R(.)$ of $\Phi$ are predicates from the schema, containing only variables different from $x$.

We divide the proof into two parts. The first part deals with the sub-formulas in pure $\mathcal{FO}$ form. Then we investigate the sub-formulas containing set variables in the second part.

Let $\chi_1 = \{\xi_i(x, \boldsymbol{y}) \mid 1 \leqslant i \leqslant k\}$ be the collection of all $\mathcal{FO}$ $\Omega$-atomic sub-formulas of $\Phi$. By applying techniques from [3], we define

$$\xi(a, b, \boldsymbol{s}) \equiv \bigwedge_{i=1}^{k} (\xi_i(a, \boldsymbol{s}) \leftrightarrow \xi_i(b, \boldsymbol{s})),$$

where $a, b$ are atomic values. Then we generate a formula $\lambda$ in the following form:

$$\lambda(x, \boldsymbol{s}) \equiv \forall x_1, x_2(x_1 < x < x_2 \rightarrow \exists y(x_1 \leqslant y \leqslant x_2 \wedge \neg\xi(x, y, \boldsymbol{s}))).$$

We denote $\Lambda$ consisting of $\lambda(x, \boldsymbol{s})$.

Let $\chi_2 = \{\eta_i(x; \boldsymbol{y}) \mid 1 \leqslant i \leqslant l\}$ be the collection of all $\mathcal{MSO}$ $\Omega$-atomic subformulae of $\Phi$ except $\chi_1$. We define

$$\zeta(a, b, D, E, \boldsymbol{t}) \equiv \bigwedge_{i=1}^{j} (\zeta_i(a, D, \boldsymbol{t}) \leftrightarrow \zeta_i(b, E, \boldsymbol{t})),$$

where $a, b$ are atomic values, and $D, E$ are set values. Now we define

$$\gamma(z, \boldsymbol{t}) = \neg z_1, z_2((z_1 < z_2) \wedge \forall z, Z, (z \in Z, z_1 < z < z_2 \wedge \zeta(a, b, D, E, \boldsymbol{t}))).$$

We denote $\Gamma$ consisting of $\gamma(x, \boldsymbol{s})$. We then finally produce $\mathcal{F} = \Lambda \cup \Gamma$. Therefore it shows that there is a function that takes as input an active-domain formula $\varphi$ , and outputs a range-restricted active domain query $Q = (\mathcal{F}, \varphi)$, while $\mathcal{F} \wedge \varphi$ is safe.

We have to guarantee that $\{a \mid D \models \varphi(a)\} = \{a \in \mathcal{F}(D) \mid D \models \varphi(a)\}$. It is trivial as $\mathcal{F}$ is CV-algebraic and based on the finiteness of the active domain and the density of order. □

### 6.2.2 Dichotomy Theorem

We now prove that dichotomy theorem holding for $\mathcal{FO}$ queries carries over to $\mathcal{SO}$ queries. We denote $\mathbf{size}(D)$ as the size of a CV-database, measured by the cardinality of the total number of tuples of $D$.

**Lemma 3**. *Let $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$ be o-minimal and based on a dense order, and let $\varphi(x; \boldsymbol{y})$ be CV-algebraic. Then there exists a number $\mathcal{K}$ such that for any $\{b_i \in U^{\tau_{b_i}}\}$, the size of the set $\{a \in U^{\tau_a} \mid \mathfrak{M} \models \varphi(a; \boldsymbol{b})\}$ is less than $\mathcal{K}$.*

*Proof.* By the Uniform Bounds Theorem[39], there is an integer $K_{\gamma_1}$ such that, for each tuple $\boldsymbol{b}$ from $U$, the set $\{x \mid \mathfrak{M} \models \gamma_1(\boldsymbol{b}, x)\}$ is composed of fewer than $K_{\gamma_1}$ intervals, where $\gamma_1$ is a first-order formula. We take $K_{\gamma_1}$ as an upper bound. As $\varphi(x; \boldsymbol{y})$ is CV-algebraic and $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$ is o-minimal, the elements of the set $\{a \in U^{\tau_a} \mid \mathfrak{M} \models \varphi(a; \boldsymbol{b})\}$ are less than $\mathcal{K}_{\gamma_1}$. Let $K_{\gamma_1} \leqslant K$. □

**Theorem 10**. *Let $D$ be a finite CV-database over an underlying structure $\mathfrak{M}$ which is o-minimal based on a dense order. Let $\varphi$ be a $\mathcal{SO}$ query without inclusion predicate. Then there exists a polynomial $\rho_\varphi : \mathbb{R} \to \mathbb{R}$ such that, for any CV-database, either $\varphi_M(D)$ is infinite or $\mathbf{size}(\varphi_M(D)) \leqslant \rho_\varphi(\mathbf{size}(D))$.*

*Proof.* Considering $\mathcal{F}$ derived in Theorem 9, because every formula $\lambda$ in $\Lambda$ and every formula $\gamma$ in $\Gamma$ is CV-algebraic, by Lemma 3, the cardinality of each formula $\lambda$ or $\gamma$ is less than some fixed number $\mathcal{K}$. Therefore if $adom(\varphi(D))$ is finite, then its cardinality is at most

$$(\sum_{f \in \mathcal{F}} \mathcal{K}_f \times 2^{n^{m_\lambda}} + \sum_{g \in \mathcal{F}} \mathcal{K}_g \times 2^{n^{m_\gamma}}) \times \parallel \tau_x \parallel,$$

where $m_\lambda$ is the number of $\boldsymbol{y}$ variables in $\lambda$, $m_\gamma$ is the number of $\boldsymbol{y}$ variables in $\gamma$, $\parallel \tau_x \parallel$ is the depth of the sort of $x$, and $n$ is the size of the active domain of $D$. □

### 6.3 Safe Constraint CV Queries: Infinite Case

In regards to the safety problem for constraint CV-databases, we consider three issues. The first issue is whether we still can express the output of our constraint CV query in terms of the class of constraints used to define the input database. For example, when the input database is a polynomial constraint database expressed in $\mathcal{FO}$ + Poly (without complex object structure), then we intend to use $\mathcal{SO}$ language to query input database. The output of a $\mathcal{SO}$ query may fail to be expressed in $\mathcal{FO}$ + Poly. Although we gain extra expressive power for our query languages, we encounter the safety (closed) problem.

The second issue is whether the output of constraint $\mathcal{SO}$ query preserves certain geometry properties of regions in $\mathcal{R}^k$. Is there effective syntax for the class of constraint $\mathcal{SO}$ queries which preserve some geometry property? Can this problem be reduced to finite query safety for embedded finite CV models?

The third issue is whether the output of constraint $\mathcal{SO}$ query preserves some certain hierarchy geometry properties. This question is very complicate. We simplify it to the question of whether the output of constraint $\mathcal{SO}$ query preserves certain geometry property in a component of the output result.

We first provide a formal definition of "tame" behavior of constraint database queries.

**Definition 24**. *In the context of constraint setting, we call a query $\varphi$ capturing the tame behavior if* 1) *the query is closed, i.e., the same sort for both input and output data,* 2) $\boldsymbol{size}(fr(\varphi_M(D))) \leqslant \boldsymbol{size}(f(fr(D)))$ *holds for the output of the query, where $fr(\boldsymbol{.})$ denotes the finite representation, $f$ is a polynomial function, and $\boldsymbol{size}(\boldsymbol{.})$ stands for the size of the finite representation of a constraint database, and* 3) $\varphi_M(D)$ *preserves certain geometry property $\mathcal{C}$ if $D$ belongs to some geometry class $\mathcal{C}$.*

Note that we call a query $\varphi$ in the above definition admits "tame" behavior as long as it preserves one geometric property in condition 3.

*Example* 16. Assume that there are a finite set of spatial objects stored in a constraint database. A simple query "return the convex hull with the vertexes at most equal $k$ is a safe query" returns convex polytopes which are represented as $\mathcal{FO}$+Poly.

**Theorem 11**. *Given a database $D(SC)$ with an underlying o-minimality structure $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$ and a dense order, there is an effective syntax characterization for the class of queries definable in $\tilde{\mathcal{SO}}(SC, \Omega)$ which possess tame behavior for constraint databases with the class of polynomial constraints.*

*Proof.* 1) It is well known that $\mathcal{FO}$ + Poly is closed for o-minimal structure. For the input database, it is finitely represented by constraint $\tilde{\mathcal{SO}}(SC, \Omega)$ formulas. Applying the procedure described in Subsection 6.2.2 by slightly modification, we will get a finitely representable constraint $\mathcal{SO}$ output. Therefore $\tilde{\mathcal{SO}}(SC, \Omega)$ is a closed language.

2) By the definition, each set variable in a $\tilde{\mathcal{SO}}(SC, \Omega)$ formula is produced by computing a finite set of $\mathcal{FO}$ formulas. Therefore in query evaluation, the size of the output finitely representable formulas will grow up at most by a polynomial size.

3) By using appropriate coding for the input database, it is well known that there is an effective syntax for $\mathcal{C}$-preserving $\mathcal{FO}$ queries. As $\tilde{\mathcal{SO}}$ has more expressive power than $\mathcal{FO}$, the most $\mathcal{C}$-preserving classes including convex polytopes, convex polyhedral, compact semi-linear sets and semi-linear sets will still hold in the output of this powerful $\mathcal{SO}$ + Poly query language. □

**Corollary 4**. *Given a database $D(SC)$ with the underlying an o-minimality structure $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$ and a dense order, there is no language $\mathbb{L}$ such that* $\mathbb{L}$+Lfp *possess tame behavior.*

*Proof.* It is well known that $\mathcal{FO}$ + LIN and $\mathcal{FO}$ + Poly are not closed under the fixpoint operator Lfp. Therefore, for any logic language $\mathbb{L}$, it is not closed under Lfp. □

**Corollary 5**. *Given a database $D(SC)$ with an underlying o-minimality structure $\mathfrak{M} = \langle \mathscr{U}, \Omega \rangle$ and a dense order, there is an effective syntax characterization for the class of queries definable in $\tilde{\mathcal{SO}}(SC, \Omega)$ which possess tame behavior for a component of constraint CV-databases (for example, one element of the relation, R.A., where R is a CV-relation and A is an attribute).*

*Proof.* By Theorem 11, there is an effective syntax characterization for the class of queries definable in $\tilde{\mathcal{SO}}(SC, \Omega)$ which possess tame behavior for constraint databases with the class of polynomial constraints. It implies that there is an effective syntax characterization for the class of queries definable in $\tilde{\mathcal{SO}}(SC, \Omega)$ which possess tame behavior for each component of the CV-database if that component of the input CV-database $D$ belongs to some geometry class $\mathcal{C}$. □

## 7 Related Work

The finite model theory is the foundation of this paper. This theory was generally introduced in [40-42]. The seminal paper of constraint databases is from Kanellakis *et al.*'s work[2]. A comprehensive treatment of this topic was presented in [8].

Various forms of collapse results in the constraint setting were introduced in [5, 9, 21, 43]. A collapse result for constraint queries over structures of a small degree was investigated by [21]. Libkin[21] gave an easy proof of a powerful form of collapse for a large class of constraints without a linear order, namely those in which all basic relations are of a small degree. Ramsey property plays a crucial role in the active-generic collapse results and was proposed in [14]. The active-generic collapse was proved in [9, 27]. In this paper, we prove that the active-generic collapse holds in monadic second-order logic.

The notion of o-minimality was extensively studied in the model-theoretic literature[44,45]. The nature-active collapse based on o-minimality structure is from [15]. A different proof of nature-active collapse for $\mathcal{FO}$ + Poly was presented in [10]. A general algorithm for nature-active collapse which converts a $\mathcal{FO}$ + Poly query in natural semantics into a formula in active semantics was given in [42]. In this paper, we propose a version of the algorithm for nature-active collapse based on a specific underlying structure for $\mathcal{CALC}^{\mathrm{cv}}$ + Poly.

Topological properties of constraint databases were studied in [46–49]. In [50], many examples of constraint queries that are expressible and inexpressible in $\mathcal{FO}$ + Lin are proposed.

Query safety notion has been extensively investigated in relational databases theory. Safety with scalar functions was studied in [35, 37]. The concept of safe translation and the decidability result for the safety of conjunctive queries over o-minimal structures are from [34]. Some results on the Dichotomy Theorem for embedded finite models are from dichotomy[34].

## 8 Conclusions

The constraints provide a sound mathematical framework to define both data models and query languages. We studied logics on classes of complex object structures and their expressive power. In addition to first-order logic, various other logics including monadic second-order logic, logics with fix-point operators, logics with transitive closure and fragments of second-order logic have been extensively investigated. We also investigated the expressive power of various query languages over constraint complex value databases. A fragment of monadic second-order logic, $\mathcal{M}\tilde{\mathcal{S}}\mathcal{O}$, admits a weaker form of the natural-active collapse in the setting of embedded finite CV model, i.e., $\mathcal{M}\tilde{\mathcal{S}}\mathcal{O}(SC, \mathfrak{M}) = \mathcal{MSO}_{\mathrm{act}}(SC, \mathfrak{M})$ for any CV-schema $SC$. However, under what conditions natural-generic collapse will hold needs to be further investigated.

We showed that the complexity upper bounds for three theories of constraints in higher order logics. Some of the classical topological queries, for example, connectivity, which cannot be expressed in $\mathcal{FO}(SC, \mathfrak{M})$, can easily be expressed in $\mathcal{SO}(SC, \mathcal{M})$, given that $\mathfrak{M}$ is an o-minimal expansion of the real field $\mathcal{R}$. We considered a syntactic condition that ensures constraint complex value calculus queries to be evaluable in closed-form in the embedded finite model.

The main results of this work not only give us deeper understanding of various logics on complex value databases with constraint settings, but also could be helpful for providing guidelines for constraint queries over advanced constraint database applications.

## References


[1] Revesz P. Big data and spatial constraint databases. In *Encyclopedia of GIS*, Shekhar S, Xiong H, Zhou X (eds.), Springer, 2017, pp.118-122.

[2] Kanellakis P, Kuper G, Revesz P. Constraint query languages. *Journal of Computer and System Sciences*, 1995, 51(1): 26-52.

[3] Libkin L. Embedded finite models and constraint databases. In *Finite Model Theory and Its Applications,* Grädel E, Kolaitis P G, Libkin L, Marx M, Spencer J, Vardi M Y (eds.), Springer, 2007, pp.257-337.

[4] Libkin L. Query languages with arithmetic and constraint databases. *SIGACT News*, 1999, 30(4): 41-50.

[5] Hull R, Su J. Domain independence and the relational calculus. *Acta Informatica*, 1994, 31(6): 513-524.

[6] Moschovakisl Y. Elementary Induction on Abstract Structures. North-Holland, 1974.

[7] Chandra A, Harel D. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 1982, 25(1): 99-128.

[8] Kuper G, Libkin L, Paredaens J. Constraint Databases. Springer, 2000.

[9] Benedikt M, Dong G, Libkin L, Wong L. Relational expressive power of constraint query languages. *Journal of the ACM*, 1998, 45(1): 1-34.

[10] Basu S. New results on quantifier elimination over real closed fields and application to constraint databases. *Journal of the ACM*, 1999, 46(4): 537-555.

[11] Tarski A. A Decision Method for Elementary Algebra and Geometry. University of California Press, 1951.

[12] van den Dries L. O-minimal structures. In *Logic: From Foundations to Applications: European Logic Colloquium*, Hodges W, Hyland M, Steinhorn C, Truss J (eds.), Oxford Science Publications, 1996, pp.99-108.

[13] Nešetřil J, Rödl V. Mathematics of Ramsey Theory. Springer-Verlag Berlin Heidelberg, 1990.

[14] Graham R L, Rothschild B L, Spencer J H. Ramsey Theory (2nd edition). Wiley-Interscience, 1990.

[15] Benedikt M, Libkin L. Relational queries over interpreted structures. *Journal of the ACM*, 2000, 47(4): 644-680.

[16] Abiteboul S, Beeri C. The power of languages for the manipulation of complex values. *The Very Large Data Bases Journal*, 1995, 4(4): 727-794.

[17] Grädel E, Gurevich Y. Metafinite model theory. *Information and Computation*, 1998, 140(1): 26-81.

[18] Rose E, Segev A. TOODM — A temporal object-oriented data model with temporal constraints. In *Proc. the 10th International Conference on the Entity Relationship Approach*, October 1991, pp.205-229.

[19] Revesz P. Safe query languages for constraint databases. *Transactions on Database Systems*, 1998, 23(1): 58-99.

[20] Abiteboul S, Hull R, Vianu V. Foundations of Databases. Addison-Wesley, 1995.

[21] Libkin L. A collapse result for constraint queries over structures of small degree. *Information Processing Letter*, 2003, 86(5): 277-281.

[22] Goldin D, Kanellakis P C. Constraint query algebras. *Constraints*, 1996, 1(1/2): 45-83.

[23] Grädel E, Kreutzer S. Descriptive complexity theory for constraint databases. In *Proc. the 13th International Workshop on Computer Science Logic*, September 1999, pp.67-81.

[24] Kuijpers B, Paredaens J, van den Bussche J. Lossless representation of topological spatial data. In *Proc. the 4th Symposium on Spatial Databases*, August 1995, pp.1-13.

[25] Segoufin L, Vianu V. Querying spatial databases via topological invariants. In *Proc. the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, June 1998, pp.89-98.

[26] Wong L. A dichotomy in the intensional expressive power of nested relational calculi augmented with aggregate functions and a powerset operator. In *Proc. the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, June 2013, pp.285-295.

[27] Otto M, van den Bussche J. First-order queries on databases embedded in an infinite structure. *Information Processing Letters*, 1996, 60(1): 37-41.

[28] Aylamazyan A K, Gilula M M, Stolboushkin A P, Schwartz G F. Reduction of the relational model with infinite domains to the case of finite domains. *Doklady Akademii Nauk SSSR*, 1986, 286(2): 308-311. (in Russian)

[29] Benedikt M, Grohe M, Libkin L, Segoufin L. Reachability and connectivity queries in constraint databases. *Journal of Computer and System Sciences*, 2003, 66(1): 169-206.

[30] Arora S, Barak B. Computational Complexity: A Modern Approach (1st edition). Cambridge University Press, 2009.

[31] Kanellakis P C, Goldin D Q. Constraint programming and database query languages. In *Proc. the 1994 International Conference on Theoretical Aspects of Computer Software*, April 1994, pp.96-120.

[32] Karp R, Ramachandran V. Parallel algorithms for shared-memory machines. In *Handbook of Theoretical Computer Science: Volume A: Algorithm and Complexity*, van Leeuween J (ed.), 1990, pp.869-941.

[33] Jonsson P, Lööw T. Computational complexity of linear constraints over the integers. *Artificial Intelligence*, 2013, 195: 44-62.

[34] Benedikt M, Libkin L. Safe constraint queries. *SIAM Journal on Computing*, 2000, 29(5): 1652-1682.

[35] Escobar-Molano M, Hull R, Jacobs D. Safety and translation of calculus queries with scalar functions. In *Proc. the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, May 1993, pp.253-264.

[36] Avron A, Lev S, Levi N. Safety, absoluteness, and computability. In *Proc. the 27th EACSL Annual Conference on Computer Science Logic*, September 2018, Article No. 8.

[37] Liu H C, Yu J X, Liang W F. Safety, domain independence and translation of complex value database queries. *Inf. Sci.*, 2008, 178(12): 2507-2533.

[38] Stolboushkin A, Taitslin M. Finite queries do not have effective syntax. In *Proc. the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, May 1995, pp.277-285.

[39] Pillary A, Steinhorn C. Definable sets in ordered structures. III. *Transactions of the American Mathematical Society*, 1988, 309(2): 469-476.

[40] Ebbinghaus H D, Flum J. Finite Model Theory (2nd edition). Springer-Verlag Berlin Heidelberg, 1995.

[41] Immerman N. Descriptive Complexity. Springer-Verlag New York, 1999.

[42] Libkin L. Elements of Finite Model Theory. Springer-Verlag Berlin Heidelberg, 2004.

[43] Paredaens J, van den Bussche J, van Gucht D. First-order queries on finite structures over the reals. *SIAM Journal on Computing*, 1998, 27(6): 1747-1763.

[44] Pillay A, Steinhorn C. Definable sets in ordered structures. *Bulletin of the American Mathematical Society*, 1984, 11(1): 159-162.

[45] van den Dries L. Tame Aopology and O-Minimality Structures. Cambridge University Press, 1998.

[46] Kuijpers B, Paredaens J, van den Bussche J. Topological elementary equivalence of closed semi-algebraic sets in the real plane. *Journal of Symbolic Logic*, 2000, 65(4): 1530-1555.

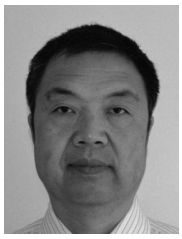
[47] Kuijpers B, van den Bussche J. On capturing first-order topological properties of planar spatial databases. In *Proc. the 7th International Conference on Database Theory*, January 1999, pp.187-198.

[48] Papadimitriou C, Suciu D, Vianu V. Topological queries in spatial databases. J*ournal of Computer and System Sciences*, 1999, 58(1): 29-53.

[49] Segoufin L, Vianu V. Querying spatial databases via topological invariants. *Journal of Computer and System Sciences*, 2000, 61(2): 270-301.

[50] Vandeurzen L, Gyssens M, van Gucht D. On the expressiveness of linear-constraint query languages for spatial databases. *Theoretical Computer Sciences*, 2001, 254(1/2): 423-463.

**Hong-Cheu Liu** received his B.S. degree in mathematics and his M.S. degree in engineering science from "National" Cheng Kung University, Tainan, in 1977 and 1985 respectively. He also received his M.Sc. degree in computer science from The University of Melbourne, Melbourne, and his Ph.D. degree in computer science from The Australian National University, Canberra. His main research areas include database theory, data mining and logic in computer science. Recently his research interests lie in biomedical informatics. In particular, he focuses on machine learning in Alzheimer's disease research. Dr. Liu is now a semi-retired researcher.



**Jixue Liu** got his Ph.D. degree in computer science from the University of South Australia, Adelaide. He is a senior lecturer in the University of South Australia, Adelaide. His research interests include integrity constraint discovery, data analytics in texts and timeseries, entity linking, algorithmic discrimination detection, and privacy in data publication. He has done work in XML functional dependencies and query translation, trust management on the Internet, integration and transformation of data, and view maintenance. He has published widely in database and data mining areas.