# SEDatalog: A Set Extension of Datalog

Qing Zhou
*Software Institute*
*Zhongshan University*
*Guangzhou,Guangdong*
*510275,P.R. China*
*lnszq@zsulink.zsu.edu.cn*

Ligong Long
*Software Institute*
*Zhongshan University*
*Guangzhou,Guangdong*
*510275,P.R. China*
*longligong@163.net*

**Abstract**

In this paper we propose an extension, SEDatalog, of Datalog so that sets can be constructed naturally in logic programming. In SEDatalog, sets can be defined by statements so it has a strong capability in creating sets, and we can use sets in SEDatalog exactly in the way we use them in mathematics or other fields. With the notion of "order" introduced in this paper, confusions among set construction levels can be avoid in SEDatalog. Three deductive rules are also introduced in this paper, which make SEDatalog being able to do deductions and to make programs even when sets are involved in deductions. The syntactical description and the semantic interpretation of SEDatalog are comprehensively discussed in detail. The soundness and completeness theorem of SEDatalog is proved, which provides a solid foundation of SEDatalog.

## 1 Introduction

In this paper we propose an extension, SEDatalog, of Datalog so that sets can be constructed in logic programming. As the extension is entirely based on what is common in every logic programming language, the extension could apply to Prolog and other logic programming languages almost without any modification. It is even theoretically possible that the extension could also apply to other categories of programming languages, such as $C^{++}$.

There have been quite a number of papers published in the field of complex object logic programming. At the beginning of 90s, *LPS* (Logical Programming with Sets) and *LDL* (logical Data language) were proposed in [3] and [8]. In *LDL*, the operators "*set grouping* $< x >$" and "*set enumeration* $\{x_1, x_2, ..., x_n\}$" were used to construct sets. The "*set enumeration* $\{x_1, x_2, ..., x_n\}$" enumerates elements in a set; and the "*set grouping* $< x >$" collect those elements with certain properties into a set. Obviously, the "*set enumeration* $\{x_1, x_2, ..., x_n\}$" can only construct finite sets. As there was no syntactical restriction on the levels of the set construction in *LDL*, the "*set grouping* $< x >$" can only be used to verify those objects in the set which are proved having the required properties; all the others are assumed not with the required properties, and hence are not in the set. So in *LDL*, only finite sets can actually constructed, and the power of defining sets are quite limited. *LPS* could only deal with those sets which contains individual objects not other sets, and it does not have a way to construct sets, instead, it uses statements such as $A :- (\forall x_1 \in X_1)...(\forall x_n \in X_n)(B_1 \wedge ... \wedge B_n)$ to simulate the operator "*set grouping*" in *LDL*. Soon after that, the Complex Object Language (*COL*) [1] was proposed. This language uses a special predicate $\in$ and $f(x) \ni a$ to collect all elements of set-valued function $f(x)$ i.e. $f(x)$ contains all elements $a$ satisfying $f(x) \ni a$. Every variable in this language has to be assigned type to distinguish different objects, this could avoid confusion on the levels of set construction. But it also obvious that the way of defining sets in *COL* is not like what we are used when we want to define a set in other areas. Also because the restriction on defining $f(x) \ni a$ is similar to *LDL*, it has also a limited power of set constructions. Later two approaches were made: {*log*} [4] and *SuRE* [6]. {*log*} uses constructors $\{a|b\}$ to construct sets $\{a\} \cup b$, so it can only accept very limited amount of sets. *SuRE* uses $\supseteq$ and $f(x) \supseteq e$, where $e$ is an expression, to collect all elements in expression $e$ into $f(x)$. In addition to that, it has constructors $\{a/b\}$ and $\{a \backslash b\}$ which indicate $\{a\} \cup b$ for $a \notin b$ or $a$ is possibly in $b$. Still it is quite obvious that the power of set construction in *SuRE* is weak. Finally, {*Relationlog*} [10] were proposed in 1998. This approach takes type, types and schema to avoid confusions on the levels of set constructions. Sets in it are partial sets and complete sets, it has a grouping operator to convert partial sets into complete sets. Its set constructor is similar to *LDL*.

The above approaches certainly contribute a lot in the research of complex object logic programming, but they have some obvious disadvantages. First, they are limited in defining sets, and most of them only allow finite sets or sets of individuals to be constructed. But in many cases when we want to use sets in programming, we might have to consider sets such as the set of natural numbers, which is infinite, or the set of classes in a school, which is a set of sets as "a class" is a set of students. Second, the ways of using sets in these approaches is different from the way we define sets when we are working in other areas. Most time when we want to use a set we define it by using a statement, for example, the set of students can be defined as $\{x : x \text{ has registered in a school}\}$, instead of inserting students' names into the set one by one. But in all the approaches above it is not possible to define sets in this way. Finally, all the approaches above need an assumption that any object without proving in a set is not in the set, i.e. a "closed world" assumption. Practically, this could causes a lot of extra work: if a set are found having an element which was not in it before from the old information, all the knowledge in database concerning the set have to be modified. As now a lot of information is put into database everyday this modification could become a regular job with this assumption.

So in our opinion, a programming language with sets has to satisfy: 1. It is convenient to use; 2. it can construct sets as much as possible; 3. the "closed world" assumption is no longer needed, i.e. it has a least model for it.

Taking the considerations above, we intend to make it possible in this paper to create sets by using statements in SEDatalog, i.e. we can define a set $A$ such that $A = \{x : p(x)\}$ for every formula $p$ in SEDatalog. By defining sets by statements we can not only construct finite sets but also infinite sets or more complicated sets such as a set of sets with certain properties. Also this is the way we define sets when we are working in mathematics, or in other areas. This definition, if without proper restrictions, would involve confusions among set construction levels and would lead to Russell's paradox. To avoid this, we restrict the elements to be contained in a set to only those which already exist, thus helping achieve clear indication of a set hierarchy to be constructed in SEDatalog. As hierarchies constructed this way are in line with the underlying principles of the axiomatic set theory i.e. $ZF$, the Zermelo-Fraenkel theory, avoidance of such paradoxes as "the set of all sets" or "a set containing itself" can be assured.

For this purpose we need an "order" for every set which indicates the level in which the set is constructed. Then statements in Datalog can be used to define sets. We consider that these sets are of the first order. Using these sets and individuals in Datalog, which are of order 0, we can construct a group of statements, which are not in Datalog in general, by which second order sets can be defined. Continuing by this way, we may have all the sets constructed. To ensure avoidance of any possible confusion in the construction, it is obvious that the construction levels of statements in SEDatalog are also needed to be clearly indicated, that is, we have to give an order to every statement in SEDatalog, too. This means that every predicate in SEDatalog can only allow those sets with order less than a constant integer (the order of the predicate) as its variables. So every predicate in SEDatalog is a partial predicate, i.e. its domain can not contain any set which has the order larger than or equal to the order of the predicate (cf. definitions in Section 2). This takes care of the problem mentioned in the last paragraph.

In $ZF$, the complement of a set is not allowed, i.e. we can not take the complement operator to a set in $ZF$. This may be good enough for dealing with mathematical matters. But we are working in the field of computer science rather than mathematics. In many cases, we need to consider the complement of a set. So the operator of complements has to be allowed in SEDatalog. Then here comes a problem: the union of a set and its complement would be the set of all sets, which would lead to Russell's paradox. To solve this problem, let us take a look at how the concept of "the complement of a set" is understood and used in our everyday life. When we say, for instance, "This is not a cat.", this "This", which is in the complement of the set of cats, here does not refer to any group of dogs or the whole world of animals but only to the thing we are pointing at, which is an individual instead of a group of something. This means that in our everyday life the complement of a set is always considered in certain limited domain rather than the whole universe. From this view point, all the complements of sets in SEDatalog are "relative complements". With the concept "order" as defined in this paper, this problem can easily be handled. Let $A$ be a set in SEDatalog with the order $= n$, then the complement of $A$ contains all the elements which are not in $A$ and have orders less than $n$. (cf. definitions in Section 2.)

As discussed above, the notion "order" is important in our work here, it is also quite technical for many users. But from the work in this paper, one can easily find out that the order of a set or a formula can be automatically created, it can be put into the background so that user even do not have to know the notion.

Another approach taken in literature is to consider higher order programming languages. With the sets defined by statements, we can consider sets as statements and hence SEDatalog has a capability to represent higher order formulas if we think that a set is identical with the formula which defines it.

The paper consists of three sections. The syntactical definitions of items in SEDatalog are given in the first section. Terms, atoms, formulas and sets are defined associated with their orders to ensure that there is no confusion in their construction

levels. In addition to that, we introduce three deductive rules, which are obviously needed to make reasoning possible in SEDatalog, and hence we can make programs with SEDatalog. The second section contains the semantic interpretations of various items in SEDatalog. The semantic interpretation of SEDatalog is introduced and models of programs are also defined in this section. In the last section we mainly discuss the soundness and completeness theorem on SEDatalog. This is a necessary part of the paper, since it provides a heuristic justification of our work. Of course, the soundness and completeness theorem must be built on the base of consistence. This is an extremely hard problem: after almost a century mathematicians still can not prove the consistence of $ZF$. In this paper we do not try to work on this issue, instead, we just make our work rely on the consistence of $ZF$. So we only prove those necessary axioms of $ZF$ in SEDatalog. This is also included in this section.

## 2 The Syntactical Description Of SEDatalog

### 2.1 Alphabet, terms, and atoms

The **alphabet** of SEDatalog consists of:

(1) $Var$ and $Const$, which are the sets of *variables* and *constants* of SEDatalog, respectively. Here $Var = \bigcup_{k=0}^{\infty} V_k$, $Const = \bigcup_{k=0}^{\infty} C_k$, and $V_0$ and $C_0$ are the sets of *variables* and *constants* of Datalog, respectively. For each non-negative integer $n$, $V_n$ consists of countably many symbols called $n$-th *order variables;* $C_n$ consists of countably many symbols called $n$-th *order constants*. We use $o(x)$ (=$n$) to denote the order of $x$ when $x \in V_n$ or when $x \in C_n$. We also require that $V_m \cap V_n = \emptyset$, $C_m \cap C_n = \emptyset$, if $m \neq n$ for all $m, n$, and $Var \cap Const = \emptyset$.

(2) *Symbols of functions*. Each $n$-ary function symbol is *associated with* a couple of non-negative integers $\langle k_1, k_2\rangle$, where $k_1 \geq k_2$. One of these symbols is, for instance, $f_{\langle 2,1\rangle}$.

(3) *Symbols of predicates*. Each $n$-ary predicate symbol is *associated with* a positive integer $\langle k\rangle$. A predicate symbol $p$ of Datalog is associated with $\langle 1\rangle$. A symbol of predicate is, for instance, $p_{\langle 2\rangle}$.

(4) *Built-in predicate symbols*. $\forall m > 0, \in_{\langle m\rangle}$, $\forall m > 0, \subseteq_{\langle m\rangle}$, and $\forall m \geq 0, =_{\langle m\rangle}$ must be included in SEDatalog language. To agree with our habits, we will use symbols $\in$, $\subseteq$ and $=$ instead of $\in_{\langle m\rangle}$, $\subseteq_{\langle m\rangle}$ and $=_{\langle m\rangle}$. They will be considered as abbreviations of the latters. Then the statement "$x \in y$" should be understood as "$x \in_{\langle m\rangle} y$ where $m$ is a fixed integer and $m = o(y) + 1$"; the statement "$x \subseteq y$" should be understood as "$x \subseteq_{\langle m\rangle} y$ where $m$ is a fixed integer and $m = o(y) + 1$"; and the statement "$x = y$" stands for "$x =_{\langle m\rangle} y$ where $m$ is a fixed integer and $m = o(y) + 1$".

(5) *Connectives* and *punctuation symbols.*

As noted in the Introduction, we need the concept *order* in SEDatalog, which is a non-negative integer expressed as $o(t)$ for each term $t$ in SEDatalog. If $o(t) = n$, then $t$ is a $n$-th order term, in particular, 0-th order terms represent *terms* in Datalog. 0-th order ground terms represent *individuals;* and $n$-th($n > 0$) order ground terms represent $n$-th ($n > 0$) order *sets*. The formal definitions of "terms" and their "orders" are given as in the following:

**Definition 1** *Terms and their orders are defined as follows:*
*(1) $x$ is a $n$-th order term and $o(x) = n$, if $x \in V_n$;*
*(2) $c$ is a $n$-th order term and $o(c) = n$, if $c \in C_n$;*
*(3) If $f$ is an $n$-ary function symbol associated with $\langle k_1, k_2\rangle$, and for every $i$, $1 \leq i \leq n$, $t[i]$ is a term with $o(t[i]) \leq k_2$, then $f(t[1], \dots, t[n])$ is a term, and $o(f(t[1], \dots, t[n])) = k_1$.*
*A term $t$ is ground if and only if no element of $Var$ occurs in $t$.*

The concept "order" has to be extended to atoms as in the following definition:

**Definition 2** *Atoms and their orders are defined as follows:*
*(1) If $p$ is a $n$-ary predicate symbol associated with $\langle k\rangle$, and for every $i$, $1 \leq i \leq n$, $t[i]$ is a term with $o(t[i]) < k$, then $p(t[1], \dots, t[n])$ is an atom and $o(p(t[1], \dots, t[n])) = k$;*
*(2) If $t[1]$ and $t[2]$ are terms, and $o(t[1]) < o(t[2])$, then $t[1] \in t[2]$ is an atom and $o(t[1] \in t[2]) = o(t[2]) + 1$;*

*(3) If $t[1]$ and $t[2]$ are terms, and $0 < o(t[1]) \leq o(t[2])$, then $t[1] \subseteq t[2]$ is an atom and $o(t[1] \subseteq t[2]) = o(t[2]) + 1$;*
*(4) If $t[1]$ and $t[2]$ are terms, and $o(t[1]) = o(t[2])$, then $t[1] = t[2]$ is an atom and $o(t[1] = t[2]) = o(t[2]) + 1$.*
*If $t[1], t[2], ..., t[n]$ are ground terms, then the atom is a ground atom.*

**Remark 1** *Note that in the above definitions, unlike Datalog, only those terms with a restriction (on orders) can be fed into a function symbol or predicate symbol to form a term or an atom. Any expression, which does not satisfy this restriction, will not be considered as a term or an atom in SEDatalog.*

**Remark 2** *There is no function symbols in Datalog, but for the sake of completeness we add functions into SEDatalog. Therefore in SEDatalog there are functions with orders $> 0$, but not functions of order $0$.*

### 2.2 Formulas and sets

Since we want to create sets in SEDatalog as much as possible, we need to extend atoms to formulas to reach our goal. In the present stage, formulas are only used to create sets, they will not take place in deductions.

**Definition 3** *Formulas are defined recursively as follows:*
*(1) An atom is a formula;*
*(2) If $F$, $G$ are formulas, then $\neg F$, $F \wedge G$, $F \vee G$ are formulas and $o(\neg F) = o(F)$, $o(F \wedge G) = max(\{o(F), o(G)\})$, $o(F \vee G) = max(\{o(F), o(G)\})$.*
*A closed formula is a formula with no free variables.*

**Remark 3** *In the above definition, $F(x) \vee G(x)$ might not be well defined from Definitions 1 and 2 for some $x$ with order $o(F(x)) \leq o(x) < o(G(x))$ when $o(F(x)) < o(G(x))$. To let the definition make sense, we consider it as $F'(x) \vee G(x)$, where $F'(x) \leftrightarrow F(x)$ and $o(F'(x)) = o(G(x))$, i.e. the domain of $F'(x)$ is extended to all terms with the order up to $o(G(x))$, although $F'(x) \leftrightarrow F(x)$. The same treatment is made for $F(x) \wedge G(x)$.*

**Definition 4** *For each formula $F(x)$ with only one free variable $x$, the set defined by $F(x)$ is a constant $C_{F(x)} \in C_n$ with $n = o(F(x))$ such that for all $t$, $t \in C_{F(x)}$ if and only if $F(t)$.*
*We often write $C_{F(x)} = \{t : F(t)\}$ to indicate the definition of $C_{F(x)}$.*

Perhaps readers unfamiliar with recursive definitions might value a quick comment here. Let $T_0 = \{t : t \text{ is a term in Datalog}\}$, $F_1 = \{P : P \text{ is a formula in Datalog}\}$. Then $T_0$ and $F_1$ are well defined. For each $n \geq 1$, let $S_n = \{s : \exists P \in F_n,\ s = C_{P(x)}\}$, and $T_n = T_{n-1} \cup \{\text{terms built of elements from } S_n \cup T_{n-1} \text{ and functions symbols with the associated couples of non-negative integers } \langle n, k_2 \rangle\}$, $F_{n+1} = \{\text{formulas built of elements from } T_n \text{ and predicate symbols with the associated positive integers } n \text{ and } \neg, \vee, \wedge\}$. Then for each $n$, $T_n$ and $F_{n+1}$ are defined from those which were already defined. So they are sound. Finally, $S = \bigcup_{n=1}^{\infty} S_n$ is the class of all sets in SEDatalog, and $F = \bigcup_{n=1}^{\infty} F_n$ is the class of all formulas in SEDatalog. So the definition here is sound, not a "cycle" definition.

Directly from our definitions, it is not hard to see that for every formula $F(x)$, $o(C_{F(x)})$ is a finite integer.

**Remark 4** *Note that in the above definition, the order of a set is the order of the formula which defines it, so the order of a set can be automatically created when we know the order of the formula which defines it. Also since the order of a formula can be easily found out from the orders of the predicates involved, it can be created automatically. So users do not have to know the order of a set or a formula when he/she makes programs. It can be put into background.*

### 2.3 Facts, rules and programs

Of course, from the title of this paper, we are presenting a set extension of Datalog. Then a natural question is what kinds of deductive rules we could have in SEDatalog to guarantee that deductions can be done? Now we discuss this issue.

As an extension of Datalog, we first extend the deduction rule of Datalog to be used on statements which contain sets as their variables. We call it "the ordinary rule". In addition to this ordinary rule, we add two more deduction rules, "the universal rule" and "the existential rule". These two rules are needed since many properties of a set are determined by properties of its elements. For example, (1) A set is a class "teacher" if everyone in the set teaches some courses in a school; (2) If a set contains an element, then we say that it is not empty. So such deductions must be included in SEDatalog.

An important issue on deductions in logic programming is to deal with the negations. In this paper we do not consider this problem. Then a difficulty arises: since the complement operator is allowed in SEDatalog, and the complement of a set $A$ is corresponding to the negation of the formula which defines $A$, the negation of the formula might be involved in a deduction in which a formula might have the complement of $A$ as its entries. To avoid this problem, we simply take all the complements of sets and all the negations of statements out of deductions (cf. definitions below.) and leave this problem to other papers. Thus the complement of a set can only be represented in this paper, but they can not take place in deductions at the present stage. This leads to:

**Definition 5** *Positive terms are defined recursively as follows:*

*(1) Each constant in $C_0$ is a positive term;*

*(2) Each variable is a positive term;*

*(3) A formula is a positive formula if it is built up from atoms and $\wedge$, $\vee$. If $F(x)$ is a positive formula and each term occurring in $F(x)$ is a positive term, then $C_{F(x)}$ =$\{x : x$ is a positive term and $F(x)$ holds$\}$ is a positive term;*

*(4) A function $f$ is a positive function if $f(t[1], ... , t[n])$ is positive whenever $t[1], ... , t[n]$ are positive. If $t[1], ... , t[n]$ are positive terms and $f$ is a positive function, then $f(t[1], ... , t[n])$ is a positive term.*

**Definition 6** *A literal is an atom in which each term is a positive term.*

**Definition 7** *A rule of SEDatalog is of the form $H : -A_1, ... , A_n$ where $n \geq 0$. The left hand side of $: -$ is a literal, called the head of the rule, while the right hand side is a conjunction of literals, called the body of the rule .*

*A fact is a special rule, whose head is a ground literal and whose body is empty.*

For convenience, we use the notation $vars(T)$ to indicate all variables occurring in $T$, where $T$ is a term or a formula. Then $vars$ is a mapping from the set of terms and formulas to the power set of $Var$. We use $H(y)$ to represent a literal with variable $y$ and $A(x)$ to represent a literal with variable $x$. Now we give the following three rules, which will be called **safe.** They are the basic deductive rules in SEDatalog**:**

(1) **Ordinary rule** is of the form

$$H : -_O \ A_1, ... , A_n$$

where $n > 0$, $vars(H) \subseteq vars(A_1 \wedge ... \wedge A_n)$ and $o(H) = o(A_1 \wedge ... \wedge A_n)$.

The informal semantics of this rule is to mean that "for every assignment to each variable, if $A_1$, ... , $A_n$ are true, then $H$ is true"

(2) **Universal rule** is of the form

$$H(y) : -_U \ A_1(x), ... , A_m(x), A_{m+1}, ... , A_n$$

where $vars(H(y)) - \{y\} \subseteq vars(A_1(x) \wedge ... \wedge A_m(x) \wedge A_{m+1} \wedge ... \wedge A_n) - \{x\}$, $y \neq x$, $o(H(y)) > o(A_1(x) \wedge ... \wedge A_m(x) \wedge A_{m+1} \wedge ... \wedge A_n)$ and $y \subseteq \{x : A_1(x) \wedge ... \wedge A_m(x) \wedge A_{m+1} \wedge ... \wedge A_n\}$ .

The informal semantics of this rule is to mean that "if every element $x$ in $y$ has properties $A_1(x)$, ... , $A_m(x)$, then $y$ has the property $H$".

In this case, $y$ is called **of the universal property** $H$.

(3) **Existential rule** is of the form

$$H(y) : -_E \ A_1(x_1, ..., x_k), ..., A_m(x_1, ..., x_k), A_{m+1}, ...A_n$$

where $vars(H(y)) - \{y\} \subseteq vars(A_1(x_1, ... , x_k) \wedge ... \wedge A_m(x_1, ... , x_k) \wedge A_{m+1} \wedge ... \wedge A_n) - \{x_1, ... , x_k\}$, $y \neq x_1$, ... , $y \neq x_k$, $o(H(y)) > o(A_1(x_1, ... , x_k) \wedge ... \wedge A_m(x_1, ... , x_k) \wedge A_{m+1} \wedge ... \wedge A_n)$ and $x_1 \in y$, ... , $x_k \in y$.

The informal semantics of this rule is to mean that "if some elements $x_1, ... , x_k$ in $y$ have properties $A_1(x_1, ... , x_k)$, ... , $A_m(x_1, ... , x_k)$, then $y$ has the property $H$".

In this case, $y$ is called **of the existential property** $H$.

**Definition 8** *A SEDatalog program is a finite sequence of rules.*

**Example 1** *By this definition, following are SEDatalog programs:*

*(1) integer*$(x) : -_O$ *even*$(x)$

*int_set*$(y) : -_U$ *integer*$(x)$ *where* $y \subseteq \{t : integer(t)\}$

*(2) working_team*$(z) : -_E$*worker*$(x)$, *worker*$(y)$, *work_together*$(x, y)$, *where* $x \in z$, $y \in z$

*area*$(z, s) : -_O$ *disc*$(z)$, $y \subseteq z$, *radius*$(y)$, $s = PI \times y \times y$

**Definition 9** *A substitution* $\theta$ *is a finite set of the form* $\{x_1/t_1, ... , x_n/t_n\}$, *where* $x_1, ... , x_n$ *are distinct variables and each* $t_i$ *is a positive term such that* $x_i \neq t_i$, *and* $o(t_i) \leq o(x_i)$.

*The set of variables* $\{x_1, ... , x_n\}$ *is called the domain of* $\theta$.

*If* $t$ *is a term, then* $t\theta$ *denotes the term which is obtained from* $t$ *by simultaneously replacing each* $x_i$ *that occurs in* $t$ *by the corresponding term* $t_i$, *if* $x_i/t_i$ *is an element of* $\theta$.

*If* $L$ *is a literal and* $L\theta$, *which is obtained from* $L$ *by simultaneously replacing each* $x_i$ *that occurs in* $L$ *by the corresponding term* $t_i$, *is also a literal, then* $\theta$ *is a legal substitution for* $L$

*If* $r : H : -A_1, ... , A_n$ *is a rule and* $\theta$ *is a legal substitution for* $H$, $A_1$, ... , $A_n$, *then* $\theta$ *is a legal substitution for the rule* $r$

*If each* $t_i$ *is ground, then* $\theta$ *is a ground substitution .*

## 3 The semantic Interpretations of SEDatalog

Let $M_0 = \langle \mathcal{V}, \mathcal{P}_0, \mathcal{T}_0 \rangle$ be an interpretation of Datalog, where $\mathcal{V}$ is the universe of $M_0$; $\mathcal{P}_0$ is the set of the interpretations of predicate symbols of Datalog; $\mathcal{T}_0$ is the set of interpretations of those ground atoms of Datalog which are interpreted as true; respectively. We define:

$$\mathcal{U}_0 = \mathcal{V}, \ and \, \mathcal{U}_n = \mathcal{U}_{n-1} \cup \wp(\mathcal{U}_{n-1})$$

where $\wp(\mathcal{U}_{n-1})$ is the power set of $\mathcal{U}_{n-1}$, i.e. $\wp(\mathcal{U}_{n-1}) = \{A : A \subseteq \mathcal{U}_{n-1}\}$. Then we give the full description of the interpretation of SEDatalog as follows:

An interpretation $M$ of SEDatalog is a tuple: $M = \langle \mathcal{U}, \mathcal{F}, \mathcal{P}, \mathcal{T} \rangle$, here $\mathcal{U} = \bigcup_{k=0}^{\infty} \mathcal{U}_k$ is the *universe* of $M$; $\mathcal{F}$ is the set of the interpretations of function symbols; $\mathcal{P}$ is the set of the interpretations of predicate symbols; $\mathcal{T}$ is the set of interpretations of those ground literals which are interpreted as true, respectively, such that:

(1) Each $n$-ary function symbol $f_{\langle k_1, k_2 \rangle}$ is interpreted as a function $f_M \in \mathcal{F} : \mathcal{U}_{k_2}^n \longrightarrow \mathcal{U}_{k_1}$;

(2) Each $n$-ary predicate symbol $q_{\langle k \rangle}$ is interpreted as a predicate $q_M \in \mathcal{P}$, i.e. $q_M \subseteq \mathcal{U}_k^n$ , and $q_{\langle 1 \rangle}$ is interpreted like in $M_0$, i.e. a predicate $q_M \in \mathcal{P}_0$;

Especially, $\in_{\langle m \rangle}$is the membership relation between elements of $\mathcal{U}_{m-1}$ and elements of $\mathcal{U}_m$, $\subseteq_{\langle m \rangle}$is the inclusion relation between elements of $\mathcal{U}_m$, and $=_{\langle m \rangle}$is the equal relation on $\mathcal{U}_m$.

(3) Each constant $c$ in $C_n$ $(n > 0)$ is interpreted as an object (set) $M(c)$ of $\mathcal{U}_n$; and each constant $c$ in $C_0$ is interpreted like in $M_0$, i.e. an object (individual) $M(c)$ of $\mathcal{U}_0$;

(4) A ground term $f(t[1], ... , t[n])$ is interpreted as $M(f(t[1], ..., t[n])) = f_M(M(t[1]), ... , M(t[n]))$;

(5) A ground atom $q(t[1], ..., t[n])$ is interpreted as $M(q(t[1], ..., t[n])) \iff q_M(M(t[1]), ... , M(t[n]))$;

(6)

$\mathcal{T}_1 \subseteq \mathcal{T}_0 \cup \{q_M(M(t[1]), ... , M(t[n])) : q_{<1>}(t[1], ... , t[n])$ is a ground literal$\}$;

$\mathcal{T}_k \subseteq \mathcal{T}_{k-1} \cup \{q_M(M(t[1]), ... , M(t[n])) : q_{<k>}(t[1], ... , t[n])$ is a ground literal$\}$ such that:

i) $M(c \in C_{A(x)})) \in \mathcal{T}_{k+1}$ if and only if $M(A(c)) \in \mathcal{T}_k$ ;

ii) $M(c \in C_{F(x) \wedge G(x)}) \in \mathcal{T}_k$ if and only if $M(c \in C_{F(x)}) \in \mathcal{T}_k$ and $M(c \in C_{G(x)}) \in \mathcal{T}_k$ ;

iii) $M(c \in C_{F(x) \vee G(x)}) \in \mathcal{T}_k$ if and only if $M(c \in C_{F(x)}) \in \mathcal{T}_k$ or $M(c \in C_{G(x)}) \in \mathcal{T}_k$ ;

iv) $M(C_{F(x)} \subseteq C_{G(x)}) \in \mathcal{T}_k$ if and only if for all $x \in \mathcal{U}$, $M(F)(x) \in \mathcal{T}_k$ implies that $M(G)(x) \in \mathcal{T}_k$,

and Finally, we set $\mathcal{T} = \bigcup_{k=1}^{\infty} \mathcal{T}_k$.

With the interpretation of SEDatalog described above, our next job is to give the description of the model of a SEDatalog program:

Let $P$ be a program. An interpretation $M = \langle \mathcal{U}, \mathcal{F}, \mathcal{P}, \mathcal{T} \rangle$ of SEDatalog is a *model* of $P$ if and only if

(1) If $A$ is a fact in $P$, then $M(A) \in \mathcal{T}$;

(2) If $r : H : -_O\ A_1, ... , A_n$ is an ordinary rule in $P$, then for each ground and legal substitution $\theta$ with $domain(\theta) \supseteq vars(r)$, if $M(A_1\theta) \in \mathcal{T}$, ... , $M(A_n\theta) \in \mathcal{T}$, then $M(H\theta) \in \mathcal{T}$;

(3) If $r : H(y) : -_U\ A_1(x), ... , A_m(x), A_{m+1}, ... , A_n$ is a universal rule in $P$, then for each ground and legal substitution $\theta$ with $domain(\theta) = vars(r) - \{x\}$, if $M(A_{m+1}\theta) \in \mathcal{T}$, ... , $M(A_n\theta) \in \mathcal{T}$, and $M(y\theta \subseteq C_{A_1(x)\theta}) \in \mathcal{T}$, ... , $M(y\theta \subseteq C_{A_m(x)\theta}) \in \mathcal{T}$, then $M(H(y)\theta) \in \mathcal{T}$, here $M(y\theta \subseteq C_{A_i(x)\theta}) \in \mathcal{T}$, $1 \leq i \leq m$, means that for any ground and legal substitution $\theta'$ with $x$ in its domain, $M((A_i(x)\theta)\theta') \in \mathcal{T}$ whenever $M(((x \in y)\theta)\theta') \in \mathcal{T}$;

(4) If $r : H(y) : -_E\ A_1(x_1, ... , x_k), ... , A_m(x_1, ... , x_k), A_{m+1}, ... , A_n$ is a existential rule in $P$, then for each ground and legal substitution $\theta$ with $domain(\theta) = vars(r)$, if $M(A_1(x_1, ... , x_k)\theta) \in \mathcal{T}$, ... , $M(A_m(x_1, ... , x_k)\theta) \in \mathcal{T}$, $M(A_{m+1}\theta) \in \mathcal{T}$, ... , $M(A_n\theta) \in \mathcal{T}$, and $M((x_1 \in y)\theta) \in \mathcal{T}$, ... , $M((x_k \in y)\theta) \in \mathcal{T}$ then $M(H(y)\theta) \in \mathcal{T}$.

**Definition 10** *Let $A$ be a ground literal. An interpretation $M = \langle \mathcal{U}, \mathcal{F}, \mathcal{P}, \mathcal{T} \rangle$ is a model of $A$ if and only if $M(A) \in \mathcal{T}$.*

## 4 Discussions On Theoretical Issues Of SEDatalog

A very important problem in dealing with sets is, of course, the consistence. As indicated in the Introduction, we do not intend to prove the consistence of SEDatalog, instead, we prove that axioms of $ZF$ hold in SEDatalog. Among the axioms of $ZF$, some of them such as the axiom of infinity, asserts the existence of certain sets. These axioms are important for mathematics, but not important to the consistence. So we do not need to have such axioms in SEDatalog, and hence they are not under our consideration. Similar ideals on this issue were represented in [4] and [7].

(1) Axiom of Extensionality

If $o(X) > 0$ and $o(Y) > 0$, then $X = Y \leftrightarrow \forall t(t \in X \leftrightarrow t \in Y)$

Proof. By definition there exist formulas $F(x)$ and $G(y)$ such that $X = C_{F(x)}$ and $Y = C_{G(y)}$.

Then $t \in X \leftrightarrow F(t)$ and $t \in Y \leftrightarrow G(t)$. So $X = Y \leftrightarrow C_{F(x)} = C_{G(x)} \leftrightarrow (F(x) \leftrightarrow G(x)) \leftrightarrow \forall t(t \in X \leftrightarrow t \in Y)$

(2) Axiom of Regularity

If $o(X) > 0$ and $\exists a(a \in X))$, then $\exists Y(Y \in X \wedge \forall Z(Z \in Y \rightarrow Z \notin X))$

Proof. Because $o(X) > 0$ and $\{o(x) : x \in X\} \subset \mathcal{N}$ is not empty, $min(\{o(x) : x \in X\})$ exists.

Choose $Y \in X$ such that $o(Y) = min(\{o(x) : x \in X\})$. Then

i) if $o(Y) = 0$, there is no $Z \in Y$

ii) if $o(Y) > 0$, then for every $Z \in Y$, $o(Z) < o(Y)$. But we also have that for every $x \in X$, $o(x) \geq min(\{o(x) : x \in X\}) = o(Y)$. So $Z \notin X$.

(3) Axiom of Replacement

If $o(X) > 0$, then $(\forall x(x \in X)\exists! y P(x, y) \rightarrow \exists Y(o(Y) > 0 \wedge \forall y(y \in Y \leftrightarrow \exists z(z \in X \wedge P(z, y)))))$

Proof. By definition there exists a formula $F(x)$ such that $X = C_{F(x)}$. Let $Q(y)$ is defined by the rule $Q(y) : -_O\ F(z), P(z, y)$ and $Y = C_{Q(y)}$.

Then $Y$ is the witness of the consequence.

Since we have proved axioms of $ZF$ in SEDatalog, it can be assumed that SEDatalog is consistent, and hence it makes sense to consider the soundness and completeness theorem of SEDatalog. In order to prove this let us introduce some related notions first.

**Definition 11** *A ground literal $A$ is a consequence of a SEDatalog program $P$ (denoted by $P \models A$) if and only if each model $M$ of $P$ is also a model of $A$.*

**Definition 12** *A ground literal $A$ is inferred from a SEDatalog program $P$ (denoted by $P \vdash A$) is defined as follows:*

*(1) If $A = H$ and $H$ is a fact in $P$, then $P \vdash A$;*

*(2) If there exists an ordinary rule $r : H : -_O\ A_1, .... , A_n$ in $P$ and a ground and legal substitution $\theta$, where $domain(\theta) = vars(r)$, such that $A = H\theta$ and $P \vdash A_1\theta$, ... , $P \vdash A_n\theta$, then $P \vdash A$;*

*(3) If there exists a universal rule $r : H(y) : -_U A_1(x), ... , A_m(x), A_{m+1}, ... , A_n$ in $P$ and a ground and legal substitution $\theta$, where $domain(\theta) = vars(r) - \{x\}$, such that $A = H(y)\theta$ and $P \vdash A_{m+1}\theta$, ... , $P \vdash A_n\theta$, and $P \vdash y\theta \subseteq C_{A_1(x)\theta}$, ... , $P \vdash y\theta \subseteq C_{A_m(x)\theta}$, then $P \vdash A$, here $P \vdash y\theta \subseteq C_{A_i(x)\theta}$, $1 \leq i \leq m$, means that for any ground and legal substitution $\theta'$ with $x$ in its domain, $P \vdash (A_i(x)\theta)\theta'$ whenever $P \vdash ((x \in y)\theta)\theta'$;*

*(4) If there exists an existential rule* $r : H(y) : -_E A_1(x_1, ..., x_k), ..., A_m(x_1, ..., x_k), A_{m+1}, ..., A_n$ *in* $P$ *and a ground and legal substitution* $\theta$*, where* $domain(\theta) = vars(r)$*, such that* $A = H(y)\theta$ *and* $P \vdash A_1(x_1, ..., x_k)\theta$*, ...,* $P \vdash A_m(x_1, ..., x_k)\theta$*,* $P \vdash A_{m+1}\theta$*, ...,* $P \vdash A_n\theta$*,* $P \vdash x_1\theta \in y\theta$*, ...,* $P \vdash x_k\theta \in y\theta$*, then* $P \vdash A$*.*

Let $infer(P) = \{A : P \vdash A\}$ and $cons(P) = \{A : P \models A\}$. It is easy to show that, $cons(P) = \bigcap_M \{A : M = \langle \mathcal{U}, \mathcal{F}, \mathcal{P}, \mathcal{T} \rangle$ is a model of $P$ and $M(A) \in \mathcal{T}.\}$.

**Theorem 1 (The Soundness and Completeness Theorem)** $infer(P) = cons(P)$.

**Proof.** (a) $infer(P) \subseteq cons(P)$

Assume $M = \langle \mathcal{U}, \mathcal{F}, \mathcal{P}, \mathcal{T} \rangle$ is a model of $P$. Given a statement $A$, we prove, by induction on the order of the statement $A$, that $P \vdash A \Rightarrow M(A) \in \mathcal{T}$.

*Induction Base*: If $P \vdash A$ and $o(A) = 1$, since $A$ is a statement of Datalog, $P \models A$, $M(A) \in \mathcal{T}_0 \subset \mathcal{T}$, by the soundness of Datalog.

*Induction Step*: Suppose that $P \vdash A \Rightarrow M(A) \in \mathcal{T}$ for all $A$ with $o(A) \leq i$, and assume that $P \vdash A$ and $o(A) = i + 1$. Then we prove, by induction on the proof tree, that

$$M(A) \in \mathcal{T}. \qquad (*)$$

*Induction Base*: $A$ is a fact in $P$, by the definition of models, $M(A) \in \mathcal{T}$.

*Induction Step*: If $A$ is obtained by using deduction rules of SEDatalog, we have three possibilities:

(1) Let $A$ be obtained by an ordinary rule $r : H : -_O A_1, ..., A_n$. Then in $P$ there exists a ground and legal substitution $\theta$, where $domain(\theta) = vars(r)$, such that $A = H\theta$ and $P \vdash A_1\theta$, ..., $P \vdash A_n\theta$. By the induction hypothesis of the proof tree $M(A_1\theta) \in \mathcal{T}$, ..., $M(A_m\theta) \in \mathcal{T}$. By the definition of models, $M(A) \in \mathcal{T}$.

(2) Let $A$ be obtained by a universal rule $r : H(y) : -_U A_1(x), ..., A_m(x), A_{m+1}, ..., A_n$. Then in $P$ there exists a ground and legal substitution $\theta$, where $domain(\theta) = vars(r) - \{x\}$, such that $A = H(y)\theta$ and $P \vdash A_1(x)\theta$, ..., $P \vdash A_m(x)\theta$, $P \vdash A_{m+1}\theta$, ..., $P \vdash A_n\theta$; also from the conditions of the universal rule, we have that $P \vdash y\theta \subseteq C_{A_1(x)\theta}$, ..., $P \vdash y\theta \subseteq C_{A_m(x)\theta}$. By the induction hypothesis of the proof tree $M(A_1\theta) \in \mathcal{T}$, ..., $M(A_n\theta) \in \mathcal{T}$, and since $P \vdash y\theta \subseteq C_{A_k(x)\theta}$, $1 \leq k \leq m$, we have that for any ground and legal substitution $\theta'$ with $x$ in its domain, $P \vdash ((x \in y)\theta)\theta'$ implies that $P \vdash (A_k(x)\theta)\theta'$ for all $1 \leq k \leq m$. Now it is easy to see that $o(H(y)\theta) > o(((x \in y)\theta)\theta')$, $o((A_k(x)\theta)\theta')$, for all $1 \leq k \leq m$. Then by the induction hypothesis of the order, we have that $M((A_k(x)\theta)\theta') \in \mathcal{T}$ whenever $M(D(x)\theta') \in \mathcal{T}$, where $y\theta = C_{D(x)}$. Because of the arbitrary of $\theta'$, we have that for all $x \in \mathcal{U}$, $M(D)(x) \in \mathcal{T}$ implies that $M(A_k\theta)(x) \in \mathcal{T}$. So $M(y\theta \subseteq C_{A_k(x)\theta}) \in \mathcal{T}$ by Definition 10. Then from the definition of the model of $P$, $M(A) \in \mathcal{T}$.

(3) Let $A$ be obtained by an existential rule $r : H(y) : -_E A_1(x_1, ..., x_k), ..., A_m(x_1, ..., x_k), A_{m+1}, ..., A_n$. Then in $P$, there exists a ground and legal substitution $\theta$, where $domain(\theta) = vars(r)$, such that $A = H(y)\theta$ and $P \vdash A_1(x_1, ..., x_k)\theta$, ..., $P \vdash A_m(x_1, ..., x_k)\theta$, $P \vdash A_{m+1}\theta$, ..., $P \vdash A_n\theta$; also from the conditions of the existential rule, $P \vdash (x_1 \in y)\theta$, ..., $P \vdash (x_k \in y)\theta$. By the induction hypothesis of the proof tree $M(A_1(x_1, ..., x_k)\theta) \in \mathcal{T}, ...M(A_m(x_1, ..., x_k)\theta) \in \mathcal{T}, M(A_{m+1}\theta) \in \mathcal{T}, ..., M(A_n\theta) \in \mathcal{T}$. Also it is obvious that $o(H(y)\theta) > o((x_i \in y)\theta)$, $o((A_j(x)\theta)\theta')$, for all $1 \leq i \leq k$ and $1 \leq j \leq m$.So by the induction hypothesis of the order, we have that $M(D(x_i)\theta) \in \mathcal{T}$, where $1 \leq i \leq k$ and $y\theta = C_{D(x)}$. So $M((x_1 \in y)\theta) \in \mathcal{T}$, ..., $M(((x_k \in y)\theta) \in \mathcal{T}$. From the definition of the model of $P$, $M(A) \in \mathcal{T}$.

So we have proved the claim $(*)$, i.e. for all $A \in infer(P)$, $M(A) \in \mathcal{T}$. Because of the arbitrary of $M$, we have proved that $infer(P) \subseteq cons(P)$

(b) $infer(P) \supseteq cons(P)$

Assume $M = \langle \mathcal{U}, \mathcal{F}, \mathcal{P}, \mathcal{T} \rangle$ is a model of $P$. Let $\mathcal{T}' = \{M(A) : A \in infer(P)\}$ and $M' = \langle \mathcal{U}, \mathcal{F}, \mathcal{P}, \mathcal{T}' \rangle$. Then it is a routine exercise to prove that $M'$ is also a model of $P$. So $cons(P) = \bigcap_M \{A : M = \langle \mathcal{U}, \mathcal{F}, \mathcal{P}, \mathcal{T} \rangle$ is a model of $P$ and $M(A) \in \mathcal{T}.\} \subset \{A : M'(A) \in \mathcal{T}'\} = infer(P)$.

So $infer(P) \supseteq cons(P)$.

The proof of the Theorem is completed.

■

## References


[1] Abiteboul, S., Grumbach, S., COL: A Logic-based Language for Complex Objects, ACM TODS, 16(1), *pp.*1-30, 1991.

[2] Ceri, S., Gottlob, G., Tanca, L., Logic Programming and Databases, Springer Verlag, 1990.

[3] Chimenti, D., Gamboa, R., Krishnamurthy, R., Naqvi, S., Tsur, S., Zaniolo, C., The LDL System Prototype, IEEE Transactions on Knowledge and Data Engineering, 2(1), *pp.*76-90, 1990.

[4] Dovier, A., Omodeo, E. G., Pontelli, E., Rossi, G., {log}: A Language for Programming in Logic with Finite Sets, J. of Logic Programming, 28(1), *pp.*1-44, 1996.

[5] Jana, D., Semantics of Subset-Logic Languages, Ph.D. Dissertation, Department of Computer Science, SUNY-Buffalo, 1994.

[6] Jayaraman, B., The SuRE Programming Framework, TR 91-011, Department of Computer Science, SUNY-Buffalo, August 1991.

[7] Jayaraman, B., Jana, D., Set Constructors, Finite Sets, and Logical Semantics, J. of Logic Programming, 38, *pp.*55-77, 1999.

[8] Kuper, G. M., Logic Programming with Sets, J. of Computer and System Science, 41(1), *pp.*44-64, 1990.

[9] Levy, A., Basic Set Theory, Springer Verlag, 1979.

[10] Liu, M., Relationlog: A Typed Extension to Datalog with Sets and Tuples, J. of Logic Programming, 36, *pp.*271-299, 1998.

[11] Lloyd, J. W., Foundations of Logic Programming, Springer Verlag, 1987.

[12] Moon, K., Implementation of Subset Logic Languages, Ph.D. Dissertation, Department of Computer Science, SUNY-Buffalo, February 1995.

[13] Osorio, M., Semantics of Logic Programs with Sets, Ph.D. Dissertation, Department of Computer Science, SUNY-Buffalo, 1995.

[14] Shoenfield, J. R., Mathematical Logic, Addison Wesley, 1967.

[15] Zhou, Q., A Paradox in Hilog, Chinese J. of Computers, 19(10), *pp.*780-782, 1996(in Chinese).