面向对象数据库的推理查询语言*

张成洪 施伯乐 胡运发

(复旦大学计算机科学系 上海 200433)

摘要 本文基于复旦大学开发的一个面向对象数据库系统FOOD,提出一种推理查询语言O—Datalog.该语言能方便地表达对面向对象数据的各种查询和推理要求;它可以转换成类Datalog形式,能运用各种高效计值算法,比其它一些基于非Horn子句逻辑的语言更易于实现.O—Datalog在形式上是一种Datalog的扩充,本文着重介绍其语法和语义.

关键词 面向对象数据库系统,推理查询,Datalog.

近年来,面向对象数据库和知识库已成为数据库研究的重点.而把面向对象数据库的丰富的表达能力和知识库的推理能力结合起来,开发演绎的面向对象数据库系统(DOOD),目前已成为一种趋势.[1]关于面向对象数据模型与逻辑语言集成的系统大多是从已有知识库出发,扩充面向对象的能力.[2~4]由于知识库的数据模型是基于关系的,这类系统一般可以在规则中表达面向对象数据,对具有面向对象数据的规则进行计值,但尚未顾及面向对象数据的存储和操纵策略、内外存管理、事务管理等面向对象数据库的基本要求.有的只具有部分面向对象的特征.

随着对面向对象数据库的逻辑描述的研究,在面向对象数据库中扩充推理能力更加方便,人们也开始从事这方面的工作.自1986年Maier提出O—logic[5]以来,相继又出现了C—logic[6],F—logic[7]等关于面向对象模型的形式化描述,这些工作促进了DOOD系统的研究和开发,例如XSQL[8]和PathLog[9]就是基于F—logic提出的规则语言.但由于它们都基于非Horn逻辑,其语义很难用bottom—up计值实现,无法使用知识库中许多成熟的算法[10],因此现在还没有一个效率比较满意的系统.

比较以上2种途径,在知识库中增加面向对象特征的方法较难完成一个完整的数据库管理系统,要实现功能完备的DOOD,最好从面向对象数据库出发;这又有2种作法:①把面向对象数据库的对象定义语言、对象操纵语言都变换成基于规则的形式,但这样实现的效率不高;②我们提出的一种方法,即在已有的面向对象数据库中扩充推理查询功能,数据定义和数据操纵的功能仍然由原来的面向对象数据库系统负责,要求推理查询的模型和原来

* 本文研究得到国家863高科技项目和国家自然科学基金资助.作者张成洪,1968年生,博士,主要研究领域为面向对象数据库,知识库.施伯乐,1935年生,教授,博士导师,主要研究领域为数据库,知识库,面向对象数据库.胡运发,1940年生,教授,博士导师,主要研究领域为知识库,知识工程,多媒体技术.

本文通讯联系人:张成洪,上海200433,复旦大学计算机科学系

本文1995—06—21收到修改稿

系统的面向对象模型完全一致,推理查询语言要能理解面向对象数据,并能对它计值.

本文结合我们开发面向对象数据库系统 FOOD[1]和知识库系统 KBASE+[4]的研究,从面向对象数据库系统 FOOD 出发,提出一种推理查询语言 O－Datalog,此语言能正确表示 FOOD 中的数据,并能正确实现对这些面向对象数据的查询计值. O－Datalog 在形式上是一种 Datalog 的扩充,具体表现在如下几个方面:①类名可当作一种 EDB 谓词,而且由于一个类往往有多个属性和方法,用户可能只关心其中的一部分,为此在谓词的每个参量前加上一个属性名或方法调用作为标签(label). ②参数是类型化的,它必须和它的标签的值域类型一致;如果属性或方法的值域是复杂数据类型(如数组或集合),那么参数的类型必须是相应的复杂数据类型. ③参数标签除了一个类的直接属性名和方法名之外,还可以是路径表达式,表示嵌套的属性名和方法名. ④参数可以是各种类型的数据,还可以是另一个对象的属性值或方法值,甚至另一个对象的嵌套属性值或方法值. ⑤类名可以作为一个参数,即一个谓词中可以嵌套使用另一个谓词. 这主要是为了方便说明嵌套对象的多个性质. ⑥由于 FOOD 中把类组织成类层次,O－Datalog 允许谓词后面带"*"号,以显式说明用户关心该类的逻辑实例.

1　数据模型

O－Datalog 的数据模型采用 FOOD 系统的数据模型,它是一个将基于对象和基于值的特征结合起来的模型,支持类、类层次、对象、对象标识、多继承、重载等具有强建模能力的面向对象概念. 在下面的讨论中,我们考虑以下一些不相交的集合:关系名集合 R,类名集合 C,属性名集合 A,方法名集合 M,常量集合 $D=INT\cup STRING\cup CHAR\cup FLOAT$,对象名集合 O.

1.1　型

定义 1. 所有型的集合记为 $TYPE$,包括

(1)所有的基本型 $BTYPE$,

$$BTYPE=\{INT,STRING,CHAR,FLOAT\}\cup\{Ref(c)\mid c\in C\}\cup\{Ref(c*)\mid c\in C\}$$

(2)若 $t\in TYPE$,则 $Set\langle t\rangle$为集合型,

(3)若 $t\in TYPE$,则 $Array\langle t\rangle[i]$(其中 i 为整数)为数组型.

1.2　模　式

对于关系名 $r\in R$,r 的模式 $scheme(r)$ 为 $\{(a_1:t_1),\cdots,(a_n:t_n)\}$,其中 $a_i\in A$,$t_i\in BTYPE$. 我们把 a_i 称为属性名,t_i 称为值域,各个属性没有顺序关系.

对于类名 $c\in C$,c 的模式 $scheme(c)$分为 2 个集合:属性集合和方法集合. 属性集合类似于关系名的情况:$\{(a_1:t_1),\cdots,(a_n:t_n)\}$,只不过 $t_i\in TYPE$.

c 的方法集合为$\{m_i:t_{i1},\cdots,t_{ik}\rightarrow t_i\}$,其中 $m_i\in M$,$t_{ij}\in TYPE$,$t_i\in BTYPE$. m_i 称为方法名,t_{ij}称为参数类型,t_i 为返回值类型,在方法集合中,方法名和参数类型完全相同是不允许的,但允许方法名相同,而参数类型不尽相同,这时就是重载或多态.

1.3　类层次和继承

类名集合 C 中包含一个系统定义的类名 $class$,并且在 C 上定义了一个偏序关系 ISA^*,

使得$\langle C, ISA^* \rangle$构成一个以 *class* 为根的有向无圈图. 对于 C 中 2 个不同元素 $c1, c2$, $c1\ ISA^*\ c2$ 表示 $c1$ 是 $c2$ 的子类, $c2$ 是 $c1$ 的超类.

根据类层次, 一个类可以从其超类继承属性和方法, 所以它对应的实际模式比直接定义在该类上的模式更大. 设 $scheme^*(c)$ 代表 c 对应的所有属性和方法, 则 $scheme^*(c) = \bigcup_{c\ ISA^*\ k} scheme(k)$, 即 c 对应的模式闭包是它和它的所有超类的模式的并集.

1.4 实 例

每个关系名 $r \in R$, r 的实例是元组, r 的实例集合为元组的集合; 每个类名 $c \in C$, c 的实例是对象, 即每个类都对应一个对象集合. 设 $Instance(c)$ 记为 c 的实例集合, 则 $Instance(c)$ 是 O 的子集. 由于类层次, 还有逻辑实例的概念, 即把一个类的所有子类的直接实例也看成它的逻辑实例, 类 c 的逻辑实例集合一般记为 $Instance^*(c)$.

在讨论每个对象的状态之前, 我们先定义值及其对应的型.

定义 2. 值

(1)特殊符号 *null* 是一个值, 称为空值.

(2)$v \in D$, v 称为值, 而且如果 $v \in INT$, 则称 v 的型为 INT, 同样, $v \in STRING$, v 的型为 $STRING$; $v \in CHAR$, v 的型为 $CHAR$, $v \in FLOAT$, v 的型为 $FLOAT$.

(3)$v \in O$, v 也称为一个值, 若 $v \in Instance(c)$, $c \in C$, 则 v 的型为 $Ref(c)$. 如果同时有 $v \in Instance^*(k)$, $k \in C$, v 的型也为 $Ref(k^*)$.

(4)若对于 $v_1, \cdots, v_n$, 其中 $v_i (1 \leqslant i \leqslant n)$ 均是型为 t 的值, 则 $\{v_1, \cdots, v_n\}$ 也是值, 它的型为 $Set\langle t \rangle$.

(5)若对于 $v_1, \cdots, v_n$, 其中 $v_i (1 \leqslant i \leqslant n)$ 均是型为 t 的值(或空值), 则 $[v_1, \cdots, v_n]$ 也是值, 其型为 $Array\langle t \rangle [n]$.

由于每个 $o \in O$, o 都属于一个类 c, $c \in C$, 设 c 的 $scheme^*(c)$ 中的属性集为 $\{(a_1:t_1), (a_2:t_2), \cdots, (a_n:t_n)\}$, 则 $\{(a_1:v_1), (a_2:v_2), \cdots, (a_n:v_n)\}$ 称为 o 的状态, 其中对于 $1 \leqslant i \leqslant n$, a_i 也称为对象的属性, v_i 是一个值, 称为对象的属性值, 其型为 t_i.

类似地, 设关系对应的模式为 $\{(a_1:t_1), (a_2:t_2), \cdots, (a_n:t_n)\}$, 对于关系实例集合中的每一个元组 $[v_1, v_2, \cdots, t_n]$, v_i 是值, 且类型为 t_i.

2 语 法

符号表包括在数据模型中出现的所有符号: 关系名集合 R, 类名集合 C, 属性名集合 A, 方法名集合 M, 常量集合 D, 对象名集合 O. A 中包含一个特殊元素 *lid*, 并令谓词名集合 $B = C \cup R$, 广义常量集合 $Z = D \cup O$, 再考虑变量集合 V, 数组变量集合 $ArrayV$.

2.1 性质和表达式

一个类的性质可以是它的一个属性, 一个方法; 由于一个类的属性或方法的值域可能为另一个类, 因此它的性质还允许是嵌套属性或方法; 更进一步, 这些属性或方法的值域可能是数组类型, 其数组元素也可以算作一个性质.

定义 3. 性质 property

$property ::= basic\_prop \mid path\_prop \mid array\_prop$

basic_prop ::=*attribute*|*method*

path_prop ::=*basic_prop*.*basic_prop*|*basic_prop*.*path_prop*

array_prop ::=*basic_prop*[*range*]|*path_prop*[*range*]|*array_prop*[*range*]

其中 *range* 为数组下标范围;*attribute* 是 A 中的属性,方法 *method* 定义如下:

method::=*m*()|*m*(*argument_list*)

argument_list::=*value*|*value*,*argument_list*

其中 $m\in M$,*value* 是一个值.

表达式一般包括常数、对象、对象的性质、变量,还包括集合表达式和数组表达式,其定义如下:

定义 4. 表达式 express

express ::=*simple_expr*|*array_v_expr*|{*expr_list*}|[*expr_list*]

simple_expr ::=*constant*|*object*|*object*.*property*|*variable*

array_v_expr::=*array_v*|*array_v_expr*[*range*]

expr_list ::=*express*|*express*,*expr_list*

其中 *constant* 是 D 中的一个常量,*object* 是 O 中的对象,*variable* 是 V 中的一个变量,*array_v* 是 *ArrayV* 中的一个数组变量. *range* 和 *property* 同定义 3.

2.2 原子、公式和规则

关系原子同 *Datalog* 中的原子类似,形式为 $r(t_1,t_2,\cdots,t_n)$,$r\in R$,每个 t_i 是一个简单表达式 *simple_expr*.

类名原子的形式为 $c(p_1:t_1,p_2:t_2,\cdots,p_n:t_n)$,其中 $c\in C$,称为谓词,每个 $p_i:t_i$ 称为一个项. p_i 是一个性质 *property*,它与谓词紧密相关,假设 c 的模式闭包为 $scheme^*(c)$,由于任何一个性质都是以某个基本性质(属性和方法)*begin_p* 开头,则 $begin\_p\in scheme^*(c)$. 如果这个属性的值域为另一个类,还可往下嵌套;若值域为数组,可以使用数组性质. t_i 是一个表达式 *express*,或者为另一个类名原子. 谓词还可以使用 c^* 形式,表示考虑类 c 的逻辑实例.

另外有一类"比较原子",其形式为 $T_1\Theta T_2$,其中 T_1,T_2 为简单表达式,$\Theta\in\{=,<,>,\leqslant,\geqslant,\neq\}$.

定义 5. 以上的类名原子、关系原子和比较原子都称为原子.

定义 6. 公式①若 L 是原子,则 L 是公式;②若 $L1$ 和 $L2$ 是公式,则 $L1$,$L2$ 也是公式.

定义 7. 规则 *Head*:−*Body*. 其中体 *Body* 为一个公式,头 *Head* 为一个关系原子.

事实是无体规则.

2.3 数据库、程序和查询

和 *Datalog* 相似,我们把出现在规则头(即通过规则定义)的谓词称为 *IDB* 谓词;对于只出现在规则体中的谓词,我们理解为存储在数据库中,称为 *EDB* 谓词. 关系谓词和 *Datalog* 一样,有 *EDB* 谓词和 *IDB* 谓词之分,而类名谓词不能出现在规则头,所以全是 *EDB* 谓词. 每个类的模式定义(包括属性和方法的定义)以及其实例对象的状态(属性值)都存放在数据库中,因此数据库包含了所有 *EDB* 谓词的定义,可以把数据库理解成基事实(不带变量的事实)库.

一个程序为规则的有限集合. 由于程序中的 *EDB* 谓词都存放在数据库中,因此程序和

数据库紧密相关,有时我们强调与程序 p 相关的特定的数据库 db,称 p 为 db 上的程序.

查询是一个无头规则,记为?:—$body$.

3 语 义

在下面的讨论中,设 U,V 为集合,则 $partial(U,V)$ 表示从 U 到 V 的部分函数所组成的集合,$P(U)$ 表示集合 U 的幂集;设 $U1,U2,\cdots$ 为集合,则 $\bigcup_{k=i}(U_k)$ 表示 $U_i,U_{i+1},\cdots$ 等集合的并.

3.1 解释结构

给定解释结构 $I=(U,I_S,I_B,I_A,I_M,I_\theta)$,其中 U 为论域,即所考虑的全部数值. 令定义 2 所定义的所有值组成的集合为 S,函数 $I_S:S\to U$ 把 S 中每个值对应到 U 中. I_B 又分为 I_C 和 I_R. 函数 $I_C:C\to P(U)$ 把每个类名对应到 U 中的对象集合. $I_R:R\to\bigcup_k=1P(U_k)$ 表示每个关系名对应 U 中元素组成的 k 元组的集合,其中 k 至少为 1. $I_A:A\to partial(U,U)$ 表示把和类名有关的属性解释成 U 中对象到值的部分函数,其中 lid 解释成一个对象到它本身的恒等函数. $I_M:M\to partial(\bigcup_k=0\bigcup_k\to partial(U,U))$ 把每个方法名解释成一个部分函数,这个部分函数根据参数不同,取值为另一个从 U 中对象到数值的部分函数. 如果各参数确定之后,方法和属性类似,当参数个数为 0 时,上式变为 $M\to partial(U,U)$. I_θ 为 $P(U\times U)$,其中 θ 为比较符.

3.2 性质 property 和表达式 express 对应的解释

由于 I_S 是把 S 中每个值对应到 U 中. 设 S 中元素 s 形为 $[e_1,e_2,\cdots,e_n]$,u 为 s 在 U 中对应的元素,即 $u=I_S(s)$,u_i 为 e_i 在 U 中对应的元素,即 $u_i=I_S(e_i)$,则 $u=[u_1,u_2,\cdots,u_n]$;若 $s=\{e_1,e_2,\cdots,e_n\}$,则 u_i 隐含地属于 u,记作 $u_i\infty u$. 一个变量赋值 v 是从变量集合 V 和 $ArrayV$ 到论域 U 中一个映射. 在 v 下的解释记为 Iv.

每个性质的解释都对应一个函数 $p:U\to U$,令性质 $p1$ 的解释 $Iv(p1)$ 为函数 f,性质 $p2$ 的解释 $Iv(p2)$ 为函数 g,则 $p1.p2$ 的解释 $Iv(p1.p2)$ 为复合函数 $f\cdot g$,如果针对 U 中某个元素 u,下面把函数值 $p(u)$ 记为 $u.p$,这时 $u.Iv(p1.p2)=Iv(p2)(Iv(p1)(u))=Iv(p1)(u).Iv(p2)=u.Iv(p1).Iv(p2)$.

简单表达式所对应的解释一般为 U 中的元素,而且

$$Iv(\{expr_1,expr_2,\cdots,expr_n\})=\{Iv(expr_1),Iv(expr_2),\cdots,Iv(expr_n)\}$$

$$Iv([expr_1,expr_2,\cdots,expr_n])=[Iv(expr_1),Iv(expr_2),\cdots,Iv(expr_n)]$$

所以 $Iv(express)$ 为 U 中元素或者集合或者数组.

定义 8. 设 u_expr 和 u_prop 是 U 中元素,u_expr 和 u_prop 一致,如果

①当 u_expr 是 U 中基本元素时,u_expr 与 u_prop 相等;

②当 u_expr 是集合,即 $\{u_1,u_2,\cdots,u_n\}$ 时,$u_i\infty u\_prop$;

③当 u_expr 是数组,即 $[u_1,u_2,\cdots,u_n]$ 时,u_prop 形为 $[u_1',u_2',\cdots,u_n']$,并且 u_i 与 u_i' 一致.

3.3 满 足

设 A 为公式,解释 I 在变量赋值 v 下满足 A 记作 $I\models_v A$.

(1)当 A 为类名原子时,如 $c(a_1:t_1,a_2:t_2,\cdots,a_n:t_n)$, $I\models_v A$ *iff* 在 U 中存在 $oid\in I_C(c)$,即 oid 为类名 c 对应的集合中的元素,对于每个 $a_i:t_i$ 和 oid 满足条件:

(Ⅰ)当 t_i 为一个表达式 *express* 时,$oid.Iv(a_i)$有定义,其中 a_i 为一个性质,$Iv(t_i)$有定义,且 $oid.Iv(a_i)$与 $Iv(t_i)$一致.

(Ⅱ)当 t_i 为另一个类名原子 $c'(a_1':t_1',\cdots,a_n':t_n')$时,$oid.Iv(a_i)$有定义,且令 $oid'=oid.Iv(a_i)$,使得每个 $a_i':t_i'$和 oid'满足条件(Ⅰ)(Ⅱ),这时(Ⅰ)(Ⅱ)中的 $a_i:t_i$ 及 oid,分别用 $a_i':t_i'$和 oid'替换.

如果谓词名为 c^*时,则考虑 $oid\in I_C(c^*)$,满足条件(Ⅰ)(Ⅱ).

(2)当 A 为关系原子时,类似于带函词的 *Datalog* 的情况.

(3)当 A 为比较原子 $T_1\Theta T_2$ 时,其中 T_1,T_2 为常量或变量,$I\models_v A$ *iff* $(Iv(T1),Iv(T2))\in I_\theta$.

(4)对于公式 $L1,L2$, $I\models_v L1,L2$ *iff* $I\models_v L1$ 并且 $I\models_v L2$.

3.4 模　型

设 I 是对 O−Datalog 语言的一个解释,对于规则 $Head:-Body$, $I\models_v Head:-Body$ *iff* "如果 $I\models_v Body$, 则 $I\models_v Head$".

对于程序 P,如果对于 P 中的任一规则 r,都有 $I\models_v r$, 则一般而言,可以把 I 称为 P 的模型.但由于数据库可以看成 P 的事实部分,所以称 I 为 P 的模型时,还要强调 I 满足数据库.对于数据库中每个对象,都对应一个类名原子公式,每个关系元组,都对应一个关系原子公式,I 满足数据库就是 I 满足所有的这些原子公式.

由于数据库中有模式的概念,因此 I 满足数据库,必然满足模式,可以进一步讨论 $I=(U,I_S,I_B,I_A,I_M,I_\theta)$.设类名 c 的模式为 $scheme(c)$,实例为 $Instance(c)$. $I_C(c)$就是 c 的实例的解释组成的集合,即 $I_C(c)=\{I_S(o)\mid o\in instance(c)\}$,若属性 A 的值域为 $Ref(c')$,则 $I_S(o).I_A(attribute)\in I_C(c')$.

例 1: $uncle\_of\_student(X,Y,Z):-student(name:X,mother.father.son[1]:$
$person(name:Y,activities:\{\text{"tell story"},\text{"hunt"}\},age(1995):Z))$

是合法的 O−Datalog 规则,含义为"如果某些学生的大舅既会讲故事又会打猎,那么把这些学生、其大舅以及其大舅在 1995 年的年龄组成一个新的关系 *uncle_of_student*".

4 结　语

有关面向对象数据模型和推理查询语言集成的文献相当多,它们一般都是在数据模型中支持对象标识、复杂对象、继承等面向对象概念,同时也提供基于规则的逻辑语言,用于查询面向对象数据和进行推理.有些文献从研究面向对象数据库的逻辑基础出发,提供了面向对象数据库的基于规则的形式化描述,这类工作以 F−logic[7]为代表.基于 F−logic 又提出了一些规则语言,PathLog[9]就是其中的一个例子.不过由于 PathLog 的谓词局限于方便表达一个路径,如果查询涉及到多条路径,PathLog 谓词就不能直观地表达,而 O−Datalog 语言完全可以表达.

另一类文献是从知识库出发,扩充面向对象的能力,它们大多基于 Horn 子句逻辑,有

的也和 O—Datalog 一样，是扩充 Datalog 语言的. COMPLEX[2]就是这方面的一个例子. 它的数据模型是在关系模型中加上对象标识、对象共享、继承等概念，不过由于没有支持复杂数据结构，也不支持嵌套属性和路径表达式，更不支持方法，它的表达能力不如 O—Datalog. Coral++[3]是从知识库出发的另一个代表，其数据模型基于 C++，其说明性语言是基于 Horn 子句逻辑的，并把 C++方法调用看作外部函数，Coral++中的规则避免创建、修改和删除对象. 但 Coral++规则中没有参数标签的概念，对象的每个属性和方法调用都要单独作为一个谓词. 因此其表达方式不如 O—Datalog 简炼和方便. 另外，由于 Coral++利用 C++操纵对象，但 C++本身不是一个数据库管理系统，不能有效管理面向对象数据的存储和更新，不如 O—Datalog 利用 FOOD 系统管理数据有效.

随着对面向对象数据库的逻辑基础的研究和标准化工作的进行，面向对象数据库日益成熟，这为从面向对象数据库出发研究 DOOD 提供了现实基础. 基于面向对象数据库的方法比基于知识库的方法更方便，可利用已有系统管理面向对象数据，不用考虑对象的内外存管理等繁琐的工作. O—Datalog 语言比其它一些系统的推理语言的表达能力更强；它在一定的解释下可转换成 Datalog 程序，这种转换包含了一种有效的计值方法，下一篇文章就讨论 O—Datalog 到 Datalog 的转换以及对 O—Datalog 程序进行计值.

参考文献

1 Barja M L *et al*. An effective deductive object—oriented database through language integration. In: Proc. 20th VLDB, 1994. 463～474.

2 Greco S, Leone N, Rullo P. Complex: an object—oriented logic programming system. IEEE Transactions on Knowledge and Data Engineering, 1992, 4(4): 344～359.

3 Srivastava D *et al*. Coral++: adding object—orientation to a logic database language. In: Proc. 19th VLDB, 1993. 158～170.

4 施伯乐，周傲英. 知识库系统 KBASE+的数据模型、语言及实现. 计算机学报，1994，17(6)：409～416.

5 Maier D. A logic for objects. In: Proc. of the Workshop on Foundations of Deductive Databases and Logic Programming, Aug. 1986. 6～26.

6 Chen W, Warren D S. C—logic for complex objects. In: Proc of ACM SIGMOD, 1989. 369～378.

7 Kifer M, Lausen G. F—logic: a higher—order language for reasoning about objects, inheritance and schema. In: Proc. of ACM SIGMOD, 1989. 134～146.

8 Kifer M, Kim W, Sagiv Y. Querying object—oriented databases. In: Proc. of ACM SIGMOD, 1992. 392～402.

9 Frohn J, Lausen G, Uphoff H. Access to objects by path expressions and rules. In: Proc. 20th VLDB. 1994. 273～284.

10 Ullman J D. Principles of database and knowledge—base systems, Vol I. Rockville: Computer Science Press, 1989.

11 施伯乐，张成洪，周傲英. FOOD：一个面向对象数据库系统. 计算机应用与软件，1994，11(6)：47～53.

A DEDUCTIVE QUERY LANGUAGE OF THE OBJECT ORIENTED DATABASE

Zhang Chenghong Shi Baile Hu Yunfa

(Department of Computer Science Fudan University Shanghai 200433)

Abstract　Based on the object—oriented database system FOOD which has been developed by Fudan University, a deductive query language O—Datalog is presented in this paper. O—Datalog language can easily express various requirements to query and deduce from the object—oriented database. It can be translated to a Datalog—like form, so the O—Datalog programs can be evaluated using the efficient algorithms for evaluating Datalog programs. It's easier to implement the language than to implement those based on non—Horn logics. O—Datalog is firmly an extension of Datalog in form. The syntax and the semantics of the O—Datalog language are discussed in this paper.

Key words　Object—oriented database system, deductive query, Datalog.