计算机科学 1994Vol. 21№. 6

复杂对象及嵌套关系的语言完备性问题

聂培尧
（山东财政学院信息管理系　济南 250014）

摘　要　最近，嵌套关系模型语言及复杂对象语言的研究已引起人们的广泛重视。这些语言有些是代数结构的，有些是基于演算的，还有的是面向逻辑程序设计的。本文将介绍这些语言的发展概况、最新研究结果以及语言的表达能力，并就其完备性问题进行了讨论。

1. 引言

关于对具有自然层次结构的数据处理，到目前为止已经提出了各种数据模型。这些模型的语言大都为代数的、基于演算的、或面向逻辑程序设计的。最近，很多研究人员试图以关系模型为基准开发出完备性准则来评测这些语言的表达能力。本文的目的是评述这些研究情况，然后对复杂对象及嵌套式关系语言的表达能力做一些比较和讨论。

七十年代后期，Makinouchi[1]提出了在关系模型中抛弃第一范式约束的假设，这就是所谓非第一范式(→1NF)关系模型的开始[2-5]。在此之后，又有人提出了更一般的模型：格式模型(Format Model)[6]、逻辑数据模型[7-8]和复杂对象模型[9]。大多数这些模型都提出了数据的层次组织，它与传统的层次模型的 DML 语言不同之处是模型的 DML 语言为代数或演算。文[10]是最先引入层次模型的这种类演算语言的。最近，文[11,12]中又分别独立地提出了多种基于规则的层次对象操纵语言。除了前面讲的这些语言外，文[13-15]又提出了用于复杂对象操纵的各种语言，这些语言均为对 SQL 或 QBE 语言的扩充。如：

- IBM Heidelberg 提出的对系、元组及列表结构中对象进行操纵的类 SQL 语言[13]；
- 对数据操纵语言 QBE 和 SQL 进行扩充以用于结构数据库操纵的 GQBE[14]和 GSQL[15]语言；
- 对 SQL 语言进行扩充以用于嵌套式关系模型操纵的 SQL/NF[16]语言。

由于层次数据库结构语言的激增，导致了关系数据库方面的数据库语言的功能评测问题。文[17-19]中提出了查询语言功能的形式化评测方法，而对于复杂对象操纵语言的评测，我们可以借助于文[17-19]中的形式化评测方法。关系数据库语言的评测方法一般有以下三种：

(1)追溯到 E. F. Codd 的原始文章[20]，Codd 提出了关系代数和演算并使用演算作为功能的测度，Codd 指出，一个数据库查询语言称为"完备的"，是指该语言具有演算的功能。然而文[17]则指出，一些非常自然的查询并不能表示成为演算，这也正暴露出了 Codd[20]提出的测度方法的缺陷，即：这样一种基于特殊语言的评测方法当考虑将其用于更强的语言时是行不通的。一般我们称这种最早由 E. F. Codd 提出的完备性概念为演算完备性。文[7,21]中对复杂对象的演算完备性进行了研究。

(2)Bancilhon 和 Paredaens 于 1978 年也分别对关系模型中的语言独立完备性提出了相同的评测准则，即后来称为"BP-完备性"的准则。一般来说，一数据库语言称为 BP 完备的，是指给定具有某些结构性质的两个关系，存在从一种关系到另一个关系的查询变换。由于这种完备性评测方法是基于测定一个给定关系实例到另一关系的映象能力的，而非出于查询上的考虑，因此这种方法也存在某些缺点。

(3)第三种方法是由 Chandra 和 Harel[18]提出的，通常称为 CH-完备性。一语言称为 CH 完备的，是指对任意可计算的映射、即部分递归和保持数据库的自同构性，存在一个使用该语言所定义的查询。文[19]也提出了一种类似的处理语言完备性的方法。然而该方法存在的问题是允许在完备性语言中描述非常模糊的查询。在某些意义下，文[19]中提出的评测方法似乎太强。要指出的是，把 CH 完备性的概念推广到复杂对象查询中虽然十分简单，但到目前为止在这方面的研究仍不多见。Dalhaus 和 Makowski[35]已把文[18]中提出的 QL 语言推广到

聂培尧　副教授、副系主任，主要从事数据库系统、MIS 等领域的研究。

一类受限的复杂对象语言上，因此这种扩充的语言[35]是 CH 完备的。

嵌套式关系中提出的大多数语言均为代数的，然而这种代数结构很难进行比较，因为嵌套代数几乎全都含有对某些关系代数运算符的扩充。这里要指出的是，到目前为止在对传统的关系代数到底应如何进行扩充，应增加何种新的运算符以构成有效的嵌套代数的问题上仍没有统一的看法（虽然 NEST 和 UNNEST 运算符对于嵌套关系代数来说已很普遍，但这两种运算并不能适应于更丰富的数据模型）。对于基于演算的语言来说这种情况更是如此。通常，演算是基于简化的 Jacob 数据库逻辑的，这就说明了通常所使用的代数不是演算完备的。因此，文[22]中证明了要使其完备，必须增加附加运算符，即幂集。但是考虑到幂集的计算代价，显然这种附加运算符在复杂对象代数中不能被考虑。文[21]中提出了不使用幂集也能描述其代数表达能力的一种受限演算形式。这里要说明的是，Van Gucht 曾证明了一特殊的代数是 BP 完备的。这就说明了这样一个事实，即从结构观点来看，Van Gucht 所考虑的这种代数虽然不是演算完备的；但对 BP 完备性来说是足够强的。这也说明了 BP 完备性与演算完备性是两个不同的概念，并且 BP 完备性相对较弱。

在复杂对象的操作方面目前已开发了基于逻辑的语言[11,12]。文[22]中证明了这类基于逻辑的语言可具有基本的演算能力，这是与关系模型的一个最主要的区别。在关系模型中，不动点运算的引入增强了演算或代数的能力。在复杂对象情况下，假设代数中具有幂运算，则不动点运算不会影响语言的能力。换句话说，在复杂对象演算中有可能在集合变量上加以量词进行限定，这就使得在纯关系模型中使用不动点运算的查询表达式成为可能。

2. 复杂对象及嵌套关系

2.1 类型和对象

在关系模型中，实例值是元组的集合，基本的构造符为集合构造符和元组构造符。在进行类型（即关系模式）构造时，每种构造符都将使用一次，即先使用元组构造符，再用集合构造符，两种构造符可以重复使用。

假设存在一域名 $D_1, D_2, \cdots\cdots, D_m$ 的集合，这里的每个 $D_i (i=1,2,\cdots,m)$ 均有一值集合。一个域 D 中的元素称为原子值。进一步，我们假设存在一名字的无限集合，该集合也称为属性。

下面我们来定义复杂对象和复杂对象类型，这里的类型定义为结构定义，而复杂对象则为定义的实例值。

(1)若 D 为一域名，则 D 为一类型。对于每一与域有关的 a，a 为该类型的对象；

(2)若 T 为一类型，而 A 为该类型中不使用的名字，则 A：T 为一命名的类型，其名为 A。T 的任何实例值为 A：T 的实例值；

(3)若 T 是一类型，而 A 为该类型中不使用的属性，则{A：T}是一集合类型。若 O 为类型 T 的对象集合，则 O 为该类型的一个对象；

(4)若 $T_1, \cdots, T_n$ 是类型，且 $A_1, \cdots, A_n$ 为不在 $T_1, \cdots, T_n$ 任何一个类型中使用的属性，则$[A_1 : T_1, \cdots, A_n : T_n]$是一元组类型。若 $O_1, \cdots, O_n$ 分别是类型 $T_1, \cdots, T_n$ 的对象，则$[A_1 : O_1, \cdots, A_n : O_n]$是类型的一对象。不失一般性，我们引入 $T_{[\,]}$记号，作为一种类型，这里[ ]表示空元组，指该类型唯一的对象为空元组。

这样，我们可以对命名类型使用相应的构造运算符来创建元组和集合类型。

这里我们主要考虑结构定义的名命。形式地，一个类型为一结构定义，或者为一个名字。同一结构可有若干个名字与之相联系，表示不同的命名类型，但却共享同一结构定义。

为了说明以上的定义，我们来考虑一个供应商的例子，其对应的关系模式可由图 1 所示的类型表示。

```
database :
[  supplier :
   {  [name : [last : string, first : string], sno :
      int, city-name : string]}
   part :
   {  [pno : int, name : string, price : int]}
   supply :
   {  [sno : int, pno : int, quantity : int]}]
```

图 1 供应商-零件(Supplier-Part)数据库

2.2 数据库及查询

一数据库类型是一集合类型的元组 DB：$[R_1 : T_1, \cdots, R_n : T_n]$，而一数据库实例值则是一含有对域名已赋值的域和数据库类型的复杂对象的一种结构。这里把域作为定义的一部分是十分重要的，即使我们的主要兴趣是在不依赖于域的具体赋值查询问题上也是如此。显然，我们可以使用任意类型，即我们可以选择任何的复杂对象进行讨论。但在本文的讨论中我们仍将使用标准关系数据库，此时数据库可视为是元组的集合，这样可方便其语言的比较。在

下面的讨论中，我们将遵循关系数据库的习惯约定，使用 R_i 作为数据库 DB 的第 i 层构成部分的名字，R_i 表示相应实例值中的对象。这里每一 R_i 均被称为一数据库关系，其中在任何嵌套层的关系及元素均被视为是复杂对象。

嵌套关系（也称非第一范式关系—1NF 或 NF^2 关系）是复杂对象的特例。在嵌套关系中需要有集合和元组的构造符，因此应遵循以下的递归定义：若 $T_1,\cdots,T_n$ 是未命名的原子或嵌套类型，$A_1,\cdots,A_n$ 为不在任何 $T_1,\cdots,T_n$ 中使用的属性，$T=\{A:[A_1:T_1,\cdots,A_n:T_n]\}$ 是一嵌套类型。类型 T 的复杂对象称为嵌套关系。在 V 关系模型中[2]同样需要在每一层上至少存在一原子对象（即至少要求 T_i 为一原子）。进一步，在 V 关系中还需要使用具有原子值的属性集合来作为 V 关系的关键字。文[21]中也曾提出同样的要求。

使用命名的元组类型同样可以构造诸如象 Prolog、Datalog 或 LDL[11,17,23,24] 这样的逻辑程序设计语言中的项。在这些语言中，对任意给定的一函数符号 f 均可应用到项中，如 f(a,b)中。因为函数符号是不可解释的，这样 $f:[a,b]$ 可满足这种构造。

查询是从一数据库类型 DB 的实例值到一类型集合 T 的函数。假设 T 的域全部出现在 DB 中。对于嵌套关系，T 必须是一嵌套类型。要注意的是，对于一给定的数据库实例值的查询，其结果可能依赖于域，而不仅是依赖于关系。一般认为这种查询是不可能满足需要的。如果一查询的值（结果）不依赖于域且为相同的，则称为是域独立的。查询语言是一种表达查询的机制，若一查询仅能表达域独立的查询，则称该查询语言为域独立的。

3. 关于完备性准则

下面我简单介绍三种完备性的概念。

3.1 演算完备性

如引言中所述，复杂对象语言是演算完备的，如果该语言能对复杂对象演算中所表达的查询进行准确的定义。这里主要是指域独立演算，嵌套关系的演算完备性的定义也相类似。

复杂对象和嵌套关系的演算完备性已在很多文献中进行了研究[2,7,21,22]。在文[2]中，演算完备性的研究是在 V 关系模型中进行的。嵌套关系的演算语言和演算完备性在文[21]中有所涉及，文[7]中研究了更一般模型的演算完备性，在这种模型中其类型不是树，而是循环图。本文的讨论则主要基于文[22]中所做的研究工作。

3.2 BP 完备性

Bancilhon 和 Paredaens 最早试图找到一种适合于任何语言的完备性定义，为此他们注意到了以下“自然”查询的性质：

性质：令 Q 为从模式 R 上的实例值到另一模式 S 上实例值的映射，称 Q 是一般的，是指对每一 R 上的实例值 R 和域上的双重内射 ρ，有 $\rho(Q(R))=Q(\rho(R))$。

这一性质实质上说明了数据库查询是把数据值作为未解释的对象进行处理的。值得注意的是，若在数据值间引入序关系时则将违反这一性质。

现在我们来定义 BP 完备性。一语言 A 是 BP 完备的，当且仅当：(a)A 中的每一查询均定义了一个一般映射；(b)对于每一 R 和 S，若域上的每一双重内射 ρ 都使得 $\rho(R)=R$ 或 $\rho(R)=S$，则 A 中存在一查询 Q 使得 $S=Q(R)$。

3.3 CH 完备性

前面所介绍的完备性概念对数据库来说其范围太狭窄、太具体。在作语言的完备性研究中，应考虑到用于计算理论中的完备性概念是十分自然和必要的。一语言是完备的，是指它能精确地表示一类可计算函数。文[19]中注意到了这种完备性的概念对于数据库函数来说似乎太通用了，因为我们希望把数据库的元素作为未解释的和把关系作为无序的来处理，但是一个由图灵机所计算的函数可对不同的符号进行区分，当然也可在数据库的输入表示中使用序。文[18]中所提出的完备性定义则考虑了具有一般性约束的普通可计算性概念。一语言 A 是 CH 完备的，当且仅当：

(α)A 中的每一查询均定义了一可计算的、一般的映射；

(β)每一可计算的、一般的映射都对应一 A 中的查询。

注意，(α)蕴涵了(a)而(b)蕴涵了(β)。

这一概念概括了数据库中大多数映射。显然，由演算所定义的映射是可计算的和一般的。这样，一个具有 CH 完备的语言至少具有与演算相同的能力。对于关系模型而言，文[19]中已经证明了二元关系的传递闭包不能在演算中表达，因而演算不是 CH 完备的。CH 完备语言是由文[18,19]所提出的，对于完备性来说，文[18,19]中提出的两种语言需要具有重复结构，如 While 等。

4. 复杂对象语言及演算完备性

4.1 关于演算

这里我们使用了文[22]中的复杂对象演算概念。这是一种多分类一阶谓词语言，其项和公式的构造为：使用域名 $D_1,D_2,\cdots$，和属性来构造类型，而项包括了常数、谓词名和变量。这里假设每一常量、谓词名以及变量都与一类型相关联。另外，我们使用选择符"·"来对元组的分量进行选择。若 t 是元组类型的项，且 A 为该类型的分量名，则 t.A 为 t 类型中对应于 A 的项。对于所有的 s，$T=\{s\}$，我们有两个特殊的谓词 $\in_{s,T}$（属于谓词）和 $=_{st}$（等价谓词），当其实例值的类型明确时我们可简记为 $\in$ 和 $=$。在项中使用谓词可以产生原子公式，特别地，$t\in R$（简记 R(t)），$t\in s.A$ 和 $t\in s$ 均为公式。应用等价谓词构成公式的方法也是类似的，这里就不详述了。假设原子公式中项的类型与谓词中所使用的项的类型一致，我们可以使用连接词 $\wedge$、$\vee$、$\neg$ 和限定词 $\forall$、$\exists$ 构造其非原子公式。

令 $DB:[R_1:T_1,\cdots,R_n:T_n]$ 为一数据库模式，我们把该 DB 与仅使用域名 $D_1,\cdots,D_k$ 和谓词名 $R_1,\cdots,R_n$ 的子语言相联系，而 R_i 为 T_i 类型且在该语言中其常量和变量的类型仅涉及到模式的域名（换句话说即仅与模式域名有关）。一解释 DB 是一映象，该映象为对域赋域名 $D_1,\cdots,D_k$，对 $R_1,R_2,\cdots,R_n$ 赋 $R_1,R_2,\cdots,R_n$。这样，一解释可简单地认为是一数据库实例值。给定一解释，则对仅使用相应模式的每一域名均可确定其域值。这样，使用一解释可对限定符赋其意义，并且公式的真值也可按标准方法定义之。

一演算（查询）是一表达式 $q=\langle A:t|\varphi\rangle$，对于每一公式 $\varphi(t)$ 使得 t 是唯一的自由变量；偶对 A：t 称为查询的目标列表。令 t 的类型为 T，则查询 q 定义了一查询，即一个从 DB 的实例值到 $\{A:T\}$ 实例值的映象：

$q(DB)=\{v|v$ 的类型为 $A:t$ 且 $\varphi(v)$ 在 DB 中为真$\}$。该定义可推广到 n 个变量的公式中。

本文中所使用的语言[22]和关系演算是有区别的。因为我们现在把集合作为对象分量，所以我们不仅需要元组变量，而且也需要集合变量。考虑到谓词 $\in$ 不仅在 $t\in R$ 的原子公式中使用；而且还在 $s\in t$ 的原子公式中使用，这样，关系演算表达能力的差别可直接由数据模型间的差别导出，我们可以很容易地把演算推广到更为严格的模型中，例如推广到 $\neg$1NF 模型中。

例 4.1　考虑模式 $[R:\{[A,A']\},S:\{[C,D:\{E\}]\}]$。查询为：

(1)从 S 的元组中选择出第一个分量为第二个分量成员的元组：$s|S(s)\wedge s.C\in s.D$。

(2)R 和 S 的向量积：$t|\exists r,s(R(r)\wedge S(s)\wedge t.[A,A']=r.[A,A']\wedge t.[C,D]=s.[C,D])$。

(3)R 和 S 在 A=C 上的联结：我们使用选择将该查询表示成为向量积的组合。记向量积公式为 φ_2。φ_2 的输出变量为 t，类型为 $T_t:[A,A',C,C']$。我们先写一 C 查询来表示输入变量 R_t 的一选择，类型为 $\{T_t\}$。

$t'(R_t(t)\wedge t.A=t.C)$。

现在我们在该查询中用 $\varphi_2(t)$ 替换 $R_t(t)$ 得：

$r|\exists r,s(R(t)\wedge S(s)\wedge t.[A,A']=r.[A,A']\wedge t.[C,D]=s.[C,D])\wedge t.A=t.C$

(4)关系 R 的幂集：$t|\forall u(u\in t\rightarrow R(u))$。

(5)S 元组的第二个分量的子集的集合，不包含值 2,4 或 5。

$t|\exists s(S(s)\wedge\forall w(w\in t\rightarrow(w\in s.D\wedge w\notin\{2,4,5\})))$

最后两个查询可以使用集合表达式和子集比较符更简明地表示如下：

(4′)$t|t\subseteq R$

(5′)$t|\exists s(S(s)\wedge t\subseteq\{w'|w\in s.D\wedge w\notin\{2,4,5\})$

(6)关系 S 把 C 分量值加入到 D 分量的值集中，然后消去 C 分量，其结果为类型 $\{D:\{E\}\}$。这一查询示出了构造查询的另一种方法。给定一个定义在类型 T 上的对象函数，我们要把函数推广到一给定类型集合 $\{T\}$ 的每一元素上，这样一种函数称为过滤器。已经证明对一给定的由公式表示的适当函数，我们可以很容易地构造一个定义所需查询的公式。令 R 为类型[C,D]的谓词名，下面的公式表示了类型[C,D]元素上所求的函数：

$\forall w(w\in t\leftrightarrow w\in R.D\vee w=R.C)$，或者简记 $t=R.D\cup\{R.C\}$。

现在我们可很容易地得到所需的查询：

$t|\exists s(S(s)\wedge t=s.D\cup\{s.C\})$。

要指出的是，如(6)中所示，一个过滤函数由公式表示与一查询表示稍有不同。例如由例中所示，当一个查询是由下而上构造时，应当对其差异有所了解。

如果允许使用子集比较符 $\subseteq$ 和传统数学概念的集合定义 $\{x|\varphi\}$，则上面的(4)和(5)的查询表示可更为简洁。

我们知道，二元关系的传递闭包不能在关系演算中表示，但可在复杂对象演算中表示。如上所见，

我们可以对一给定关系表示其幂集。给定一个二元关系，我们在其两个属性上进行投影，然后进行并运算，这样就可得到全部元素的集合。我们取集合自身的向量积，即可得到一个含有传递闭包的二元关系。进一步，我们求出该关系的所有的子集，使用幂集构造符和值域为幂集元素上的变量，我们可以描述传递闭包的性质，即为包含了给定关系的最小集且是传递闭包的。

4.2 域独立和安全性

我们知道，域名仅用于类型定义之中，但是一个公式的真不仅仅依赖于 R_i 的对象，还将依赖于 D_i 的域，因为域决定了变量的取值范围。我们称一个查询为域独立的，是指该查询在任意两个具有不同域的结构中其查询结果值相同。在下面的讨论中我们将使用"域独立演算"这一术语。该术语是指限于域独立公式和查询上的演算。

域独立的定义是基于语义的。即使是传统的关系演算，其域独立性也是不可判定的[25]。在传统处理上一般是使用句法限制来确保域独立。文[24]中给出了一个确保域独立的句法规则约束公式，该公式在文[22]中称为是安全的。文[22]中给出了以下的值域约束和安全性定义：

基础：

(1)$t=R,R(t)$，

(2)$t=c,t\in c$，(c 为一常数)

这里(1)和(2)均为 t 的值域公式。

构造：若 $\varphi,\varphi_1,\cdots,\varphi_k$ 为变量 $s,s_1,\cdots,s_k$ 的值域公式，则以下为 t 的值域公式：

(3)$\exists\ s(\varphi(s)\wedge t=s.A)$

(4)$\exists\ s(\varphi(s)\wedge t\in s)$

(5)$\exists\ s_1,\cdots\exists\ s_k(\varphi_1(s_1)\wedge\cdots\wedge\varphi_k(s_k)\wedge t.A_1=s_1\wedge\cdots\wedge t.A_k=s_k)$

(6)$t=\{s\mid\Psi(s)\}$，这里的 Ψ 为一安全公式，定义如下：

(7)$\exists\ s(\varphi(s)\wedge t\subseteq s)$

闭包：

(8)若 φ_1,φ_2 为 t 的值域公式，则 $\varphi_1\vee\varphi_2$ 也为 t 的值域公式。我们称一变量 t 在一公式 α 中为值域受限的，是指其类型为空元组类型，或者以下条件成立：

(i)若 t 为 α 中的自由变量，则 α 的形式为 $\varphi(t)\wedge\Psi$。其中 φ 为 t 的值域公式；

(ii)若 t 为 α 中的存在限定词，则包含 t 的实例值的 α 的子公式的形式为 $\exists\ t(\varphi(t)\wedge\Psi)$，其中 φ 为 t 的值域公式；

(iii)若 t 为 α 中的全称限定词，则包含 t 的实例值的 α 的子公式的形式为 $\forall\ t(\varphi(t)\rightarrow\Psi)$，其中 φ 为 t 的值域公式。

一公式是安全的，是指其中的全部变量均是值域受限的。以上公式中的限定词通常也称为边界限定词，公式也可分别记为 $(\exists\ t(\varphi(t))\Psi$ 和 $(\forall\ t\varphi(t))\Psi$。

4.3 递归查询

下面讨论一种允许递归的基于简单 Horn 子句的语言。数据库中使用的谓词我们称为基谓词，一文字为一原子公式或负原子公式，一规则则为以下形式的表达式：

$$p(t)\ :-Q_1(t_1),\cdots,Q_n(t_n),$$

其中 p 为导出谓词，Q_i 为一文字(即正或负原子公式)，t 为常量和变量的向量。一个规则可以解释为以下的形式：

$\forall\ x_1\cdots\forall\ x_m(Q_1\wedge\cdots\wedge Q_n\rightarrow p)$，其中 $x_1,\cdots,x_m$ 为出现在公式中的全部变量。一程序为规则的有限集合。规则头部为导出谓词的定义，每一导出谓词均依赖于规则体中定义的谓词。如果这种依赖关系含有循环圈，我们称该程序为递归的。一个(递归)查询是一偶对(prog,Q)，其中 prog 为一(递归)程序，Q 为一正文字，其中的谓词则是一导出关系。这种查询的回答结果是一集合，该集合中的元素是由程序从给定的数据库中所导出的那些查询谓词的元组所构成。

规则、程序和查询也需要是域独立的[26]。我们说一个规则是安全的，是指每一出现在规则头部或出现在负文字中的变量在规则中为值域受限的。我们知道，当不动点程序被视为事实集合上的运算符时，这种不动点程度即为程序的最小模型，可以认为是一公式。另外，否定也是较为困难的，一般来说，对一具有否定的程序赋其意义并非总是可行的，文[28,29]中提出了一种分层的解决方法。一谓词 p 是由谓词 q 导出的，若 q 出现在以 p 为其头的规则体中。分层意指在涉及负文字的关系中导不出循环圈。分层的方法在谓词的划分上强加上了偏序关系。文[12,28]中证明了不动点语义可扩充到分层的程序上。我们可以根据偏序关系对谓词的集合自底开始计算其不动点。

4.4 代数

代数包括传统的运算符：集合运算中的并、交、差；向量积；改名运算(用于对子对象改名)；选择运算和投影运算。这里增加的一个新的、非平凡的运算

为幂集运算，用该运算可以产生一个集合的全部子集的集合。然而要指出的是，以上这些运算符还是不够的。为了实现重构，还需要一种以规定的方法改变集合中所有元素的能力，即使用过滤器。选择和投影是重构的特例，元组缩灭运算也为特例，该运算可把集合中的每一元组缩灭成一个"平面"关系元组，集合缩灭运算则可把集合集中的一集合缩灭成一个集合（这两种运算在复杂对象的一般模型中是很有用的，但在¬1NF 关系中并不需要）。其它运算符的例子还有如¬1NF 关系中的嵌套（*nest*）和非嵌套（unnest）运算。但是，由于这里我们需要对定义的对象进行递归处理，所以我们需要递归定义重构的能力。这种能力已在不同的代数中以不同的方式得到满足[22]。

文[22]采用的方法是在代数中使用了一种叫做替代（*replace*）的运算，其一般格式为 *replace*（*G*）（R），其中 G 为一替代说明。说明是由谓词、常量和属性名经过元组及集合构造符、条件构造符等构成的一代数表达式。特别地，一说明其本身包含了一替代子表达式，因而其定义也是递归的。已经证明，投影、选择、元组缩灭是替代的特殊情况。十分显然，给定一类似于替代的函数形式（即高阶运算），可定义任意深度的重构，但是其表达能力仍将依赖于有效的基本运算。文[30]中采用了类似的方法。在文[30]中虽然使用了投影和选择运算，但这两个运算既作为运算，又可作为函数形式。值得注意的是，因为具有了一丰富的、带有递归的数据结构的全域，我们可以定义函数语言模式 FP[35]的（弱）实例值代数语言。

除了不使用"."外，代数语言包含了演算。另外，我们还有元组和集合构造符"[,]"和"{,}"以及替代运算中使用的过滤运算符"(,)"。全部的运算产生集合类型的输出，并且，除了替代运算以外，全部的运算仅接收集合变量。

例 4.2 对例 4.1 中的模式[R：{[A,A′],S：{B：[C,D：{E}]}，查询的代数表达式为：

(1)replace(ifC∈D then B)(S)

使用传统的选择运算，上式可简化为：

select(C∈D)(S).

(2)tuple-collapse(cross(R,S))

(3)σ(A=C)(tuple-collapse(cross(R,S)))；

(4)powerset(R)

(5)set-collapse(replace(powerset(D−{2,4,5}))(S)

(6)replace(D∪{C})(S)。

反之，任何演算的公式均不能保证其域独立性。代数则是一种域独立的语言。

4.5 基本等价结论

定理 1[22] 代数、安全演算以及域独立演算是相互等价的。

证明的思想为使用文[24]中的传统证明方法，从演算到代数归纳的第一步为对公式中的每一变量构造一表达数据库中变量值域的代数查询。一旦构造出这些查询，继续对公式的结构施以归纳，这一步需要弱运算。现在，对于第一步，假设变量 t 的类型为"原子集合"，我们可对数据库施以投影、然后联结，并对出现在数据库中的原子集合构造一代数表达式，则 t 的域即为该表达式所描述的集合的幂集，也即为代数中所需的幂集。在对（严格的）安全演算归纳时，因为变量的域是由值域公式所限定的，第一步则直接构造了值域公式代数。特别地，对于严格的安全演算来说是不需要幂集的。

定理 2[22] 不含有幂集的代数与严格的安全演算相等价。

证明见文[22]。

定理 3[22] 一个作为分层递归查询的查询是可表达的，当且仅当该查询在安全演算（或幂集代数）中是可表达的。

证明见文[22]。

以上这些定理说明了代数、安全演算以及域独立演算是演算完备的。若把这些结论与关系模型中演算和代数相比较，可以看出在关系演算或代数中不能表达传递闭包。但不带否定的递归查询可表示这种查询。然而对于某些关系的补传递闭包仍需要使用否定和分层来表达。在本文讨论的模型中，演算和幂集代数均可表达由分层递归程序所表达的查询。正如文[32]中证明的，这些语言具有二阶关系演算的能力。生成一关系的幂集所需的时间是指数的，显然，无限制地使用幂集很费时。

5. BP 完备性

下面研究代数的 BP 完备性。BP 完备性的概念是由 F. Bancilhon 和 J. Paredaens 在文[18]中提出的。BP 完备性中的一个重要概念是关系的共组（cogroup）。一个关系的共组 CG(R) 包含了使关系不变的、出现在关系域值上的全部置换。使用共组的概念，一查询语言 L 的 BP 完备性可描述如下：

对于所有的关系 R_1 和 R_2，以下的语句是等价的：(1)在 L 中存在 Q 使得 $Q(R_1)=R_2$；(2)R_1 中所出现的 R_2 的域值以及 $CG(R_1)$ 均为 $CG(R_2)$ 的子集。

J. Paredaens 使用关系代数表达式来计算一关系的共组，并从共组中重构其原始关系。这些表达式

完全依赖于给定关系的实例值，但这也正是我们所期望的，因为共组关系模式也正是如此。

Van Gucht[29]注意到了 BP 完备性和共组的定义也可应用于嵌套关系，并且证明了包含改名、投影、选择、并、差、笛卡尔积，嵌套和非嵌套运算的嵌套代数也满足 BP 完备性准则。

定理 4[29] 嵌套代数（不带幂集）是 BP 完备的。

证明见文[29]。

嵌套代数的 BP 完备性蕴涵了所有其它用于嵌套关系操作的代数的 BP 完备性，具有更强的表达能力。要指出的是，这一结论对于幂集代数和其它所有具有相同表达能力的代数都是成立的。

6. CH 完备性

CH 完备性的概念是在文[18]中引入的，文[19]也对其进行了研究，然而这些研究考虑的是纯关系模型。文[35]中则考虑了结构对象的完备性问题。本文介绍了两种用于一般类型的复杂对象语言的 CH 完备性。第一种是对文[18]中所提出的 QL 语言的扩充；第二种是对文[19]中的 detTL 语言的扩充。

6.1 CO-QL 语言

我们首先非形式地介绍一种用于处理复杂对象的扩充 QL 语言 CO-QL[16]。CO-QL 语言使用了对象变量 $y_1, y_2, \cdots y_n$。另外，CO-QL 中的基本项或为一个变量，或为一数据库对象。复杂对象的代数表达式是使用基本项构成的，如 $\Pi_{AB}(y_3 \cup R_2)$。CO-QL 语言中的程序集合归纳定义如下：

(1)若 t 是一个项且 y_i 是一个变量，则 $y_i \leftarrow t$ 是一程序；

(2)若 p、p′ 是程序，则(p;p′)亦为程序；

(3)若 p 是一程序，且 y_i 是一变量，则"while $y_i \neq$null do p"也是一程序。

查询是由一偶对(program,i)定义的，当程序终止时，查询的结果为 y_i 的内容。要说明的是，查询是一个部分函数，因为程序可能不终止，CO-QL 可以对所有可计算的一般映射进行计算，因此 CO-QL 满足 CH 完备性的性质(β)。注意，一个变量可取任意类型的对象，但这将需要具有图灵机的计算能力（因为程序中的变量是具有类型的，因此程序在计算时所具有的状态数是受限于输入数据库中的总的递归函数的。这毕其计算能力显然受到了限制）。若在语言的结果变量中加入某些类型限制，则语言仍满足 CH 完备性定义中的性质(α)，且也是 CH 完备的。

6.2 CO-detTL 语言

文[19]中对另一种纯关系模型满足 CH 完备性的过程语言 detTL 进行了研究。该过程语言是基于元组插入、删除、组合(t,;t)、while 构造(while 〈condition〉do t done)、和 with new 构造(with new z do t done)的。该语言中还使用了暂时关系，这种关系起到了文[18]中变量的作用，其基本性质为：暂时关系是类型定义的，即具有固定元，语句"with new z do t done"可解释为对 z 赋一新原子值并由 t 中新值替代 z 而得到的程序。使用 with new 创建的值提供了语言的计算能力。对于纯关系而言语言 detTL 是 CH 完备的。

现在我们来考虑对 detTL 语言进行扩充以适用于复杂对象操作的 CO-detTL 语言。这种扩充后的 CO-detTL 语言具有对象插入、删除、组合、while 和 with new 构造功能。该语言与 detTL 语言的唯一区别在于：(1)CO-detTL 语言可对复杂对象进行操作；(2)在 while 条件中可以出现 $x \in y$ 的表达式（这里 x,y 为兼容类型的类型变量）。

CO-detTL 语言是 CH 完备的。

文[36]中给出了对于 Datalog 语言的一种扩充，并证明了这种扩充 Datalog 语言对平面关系是完备的。显然，这种语言也可以扩充到复杂对象的操作上，从而获得一种复杂对象操作的完备语言。

综上所述，对一完备关系语言进行扩充以得到一完备的复杂对象语言是可行的。这种扩充后的语言应当基本具备对复杂对象进行编码(coding)和解码(decoding)的能力。扩充后语言的完备性证明的核心应包括：

(1)能在纯关系中对复杂对象进行编码；

(2)能使用关系上下文中的完备性结果；

(3)能对结果进行解码。

文[35]中的完备性证明是按以上思路进行的。

7. 结论

本文探讨了复杂对象语言的三种完备性评测方法。在三种评测方法中，有两种是语言独立的，即 BP 完备性和 CH 完备性。第三种方法，即演算完备性，则是定义在具体语言上的。文[18]中指出了 BP 完备性作为数据库语言的完备性准则是不合适的。而 CH 完备性又似乎太强，因为它可以表达所有可计算的查询，但我们不可能对如此大类的查询语言找出一合适的优化策略。文[32]中的结果则指出，即使是演算完备性语言也是很强的，因为在这种情形中演算允许表达二阶查询。根据本文中的讨论我们可以看出，关于语言的特性看来还需要作进一步的研究。文[22]中所提出的严格安全演算或许是下一步研究的方向。（参考文献共 37 篇略）