计算机科学1998Vol.25 №.5

扩展演绎数据库技术

The technique for extending deductive database

王修伦 孙永强

(上海交通大学计算机系 上海200030)

摘 要 The key techniques for extending deductive database is analyzed in the paper which include complex types, object conception, update, transaction and dynamics. It's shown that the studies for deductive database makes the deductive database understand deeply and go toward mature. Finally we point out some problems to be resolved.

关键词 Deductive database, Complex object, Object-oriented, Update, Transaction, Dynamics

传统的演绎数据库是指将关系和逻辑程序语言相组合而形成的演绎数据库(简称演绎数据库)。针对演绎数据库的建模能力之不足,下面我们从三个方面分析如何增强其功能:

(1)传统演绎数据库基于关系数据库之上,由于关系数据库不支持复杂值,所以很难自然地表达复杂应用的语义,须扩充复杂数据结构。

(2)面向对象技术为复杂应用提供一个有效的方法,如何将对象技术引入到演绎数据库中仍需要进一步研究。

(3)动态性在演绎数据库中的表示。

1 支持复杂值的演绎数据库

显然,Datalog 的底层基于关系数据库,因此不能有效地支持新的应用,如:工程设计、图形数据库、CAD/CAM 等。这些应用要求能正确表示和操作嵌套结构的复杂结构化值。Prolog 可以借助函子来间接表示某些元组结构的复杂值,下面给出一个例子:

```
car(vehicle(ny-6556,chevy-vega,1997),
    owner(name(arthur,Keller),gotFrom(deater)))
car(vehicle(ny-6236,wolkswagen,1997),
    owner(name(arthur,Keller),gotFrom(deater)))
```
(例1)

其中 vehicle,name,owner,getFrom 为函子,并不象+,*等具有特殊的解释。上面的例子说明通过函子可以间接地描述嵌套元组。

但是 Prolog 并没有提供对集合的强大支持。为了支持集合的描述能力和语义,美国 MCC 公司设计与实现一个功能强大的演绎数据库语言,称为逻辑数据语言 LDL(Logic Data Language)[1]。在 LDL 中,集合可以有两种使用形式:枚举和聚合。变量可以在聚合上取值,聚合可以嵌套。这使得 LDL 具有很强的集合表示能力。下面我们给出两个例子:

```
book_deal({X,Y,Z}):-
    book(X,Px),book(Y,Py),book(Z,Pz),
Px+Py+Pz<100                                    (例2)
parentof(X,<Y>):-fatherof(X,Y).
parentof(X,<Y>):-matherof(X,Y).
fatherof(bob,pam),matherof(bob,tom).            (例3)
```

例2的语义是显然的,例3中引入了一个新的构造子()(集合组合 group),<Y>表示 Y 为集合的元素。因此例3的查询 parentof(bob,(Y))的结果为 parentof(bob,{pam,tom})。

可是在 LDL 中,集合表示仍然存在两个问题:(i)集合组合被限制在单一规则中,使得查询的语义是很复杂的,与逻辑程序的描述语义相违背。(ii)集合组合要求程序可层次化,使得一些具有清晰的描述语义的程序在 LDL 中是不可能的。如下面的程序:

ancestorof(X,<Y>):-parentsof(X,<Z>),anceatorsof(Z,S),Y∈S。

这个程序的目的是要递归定义 ancestorof,但是 LDL 中组合机制要求规则体中的两个关系 parentof,ancestorof 必须是已求解的,显然这样的语义在 LDL 中难以表达,

从程序的语义上看,LDL 的解释与模型间的比

王修伦 博士生,研究方向为对象数据库,孙永强 教授,博士导师,主要研究方向为计算理论,并行和分布处理。

较是基于新的序关系，称为 d_优先序关系。不幸的是 d_优先关系并不是偏序关系，这样其极小模型(Least model)概念是病态的。

COL[2]是 Datalog 语言的扩展、支持复杂对象。不象 LDL 那样用函子来间接表示元组，在 COL 中直接支持元组和元组构造子，用函子来表示集合，即把函子解释为定义域到集合的映射，这样可以用函子表示集合的聚合与枚举。例3可以用 COL 定义如下：

```
Y∈par(X):-fatherof(X,Y)
Y∈par(X):-motherof(X,Y)
parentsof(X,par(Y))                    (例4)
```

这个例子用函子 par 来构造 X 的所有双亲为一个集合 par(X)，聚合可以用几个规则实现，同样也可以用递归规则来实现集合语义，如：

```
Y∈anc(X):-Z∈par(X),Y∈anc(Z)
ancestors(X,anc(X))                    (例5)
```

这样 COL 在表达集合语义方面比 LDL 具有优越性，但是这两种语言在表达嵌套结构查询上仍然具有复杂性且不灵活。

COL 缺乏模型论语义，在一个在解释和模型间不存在正确的序关系，文[1]已说明了模型间的子集关系是不够精确的。对于一个可层次化的 COL 程序，可能有多个或无穷多个不可比较的极小模型，基于这种子集关系，必须要采取一个机制选择一个模型作为程序的指定的语义。从纯粹的模型论角度而言，这个选择出来的模型并非是程序的合理的极小模型，为解决这个问题，COL 通过证明论方法来验证所选择出来的模型是否为所需要的模型，即这个模型可以通过自底向上计算而得，这种方法看起来有点偏离了演绎数据库描述性的本质。

Relationlog[3]也是扩充 Datalog 来支持复杂值，它直接支持元组和集合构造子。在 Relationlog 中，通过引入部分集概念来描述集合的语义，完全集与 LDL 中的集合枚举类似，而部分集则通用化 LDL 中的集合组合概念，与 LDL 不同的是部分集可以在构造的头和体中同时出现。当在体中出现时，表示的是集合的部分信息，在规则头出现时，表示集合组合。如：

```
parentsof(X,{Y}):-fatherof(X,Y)
parentsof(X,{Y}):-motherof(X,Y)        (例6)
```

使用部分集可以对集合进行递归定义，如：

```
ancestorsof(X,{Y}):-parentsof(X,{Z}),ancestorsof(Z,
{Y})                                   (例7)
```

Relationlog 的程序语义基于分层概念，解释与模型间的序关系基于一种新的序关系：优先序关系。由于优先序关系是一种偏序关系，因而程序的模型是良定义的。

2 演绎对象数据库

我们首先看看面向对象与复杂对象的区别：在面向对象系统中，每一对象都有唯一的对象标识、对象按类进行分类，一个对象可以通过另一个对象的标识来引用它，通过对象标识获得信息共享。而复杂对象在语义描述方面显然不如面向对象强。下面给出一个例子：

```
r1. book("Prolog",author(name("Bob","Su"),address
("121222"))).
r2. book("Database",author(name("Bob","Su"),address
("121222"))).                          (例8)
```

当我们修改 r1为 book("Prolog",author(name("Bob","Su"),address("PS121222")))时，按语义一致性要求，应该对 r2进行相应的修改。在复杂值演绎数据库中，这样的一致性维护是比较困难的。

一种解决方法是为每一个对象都引入一个非逻辑标识，上面的规则可修改如下：

```
book("Prolog","m1").
Book("Database","m1")
author ("m1", name ("Bob","Su"), address
("121222")).
```

这与关系的正则化类似，系统为此引入一些无意义的标识，然而在关系数据库中无原则地引入这些标识必然带来许多问题。

在对象系统中统一以对象标识表示。在对象数据库系统中，一个很自然的问题是：什么是对象?。在 O_2[4]中，一个对象为对象标识和值序对。对象标识是不变的，对象可由其标识唯一地引用，与对象相关联的值可以为整数、字符、对象标识、元组和集合，是可变化的部分。值部分为对象的状态。对象一般共享公共属性，在面向对象系统中，将这些相似对象组织成类，而类又可以按子类关系划分为类层次，子类中的实例通过继承可以共享父类中的性质。性质继承使得我们能够在维护信息完全性前提下减低描述的冗余性、同时也为系统演化提供一种方法。

集合在对象数据模型中占据重要的地位，但集合在对象数据库中并没有得到足够的重视。强有力的集合表示机制在面向值的演绎数据库中受到不同程度的支持，我们在上一节已经说明了这一点。

演绎对象库语言如：F-logic[5]等也支持集合机制，但是它们对集合的处理也是很受限的。例如，利用单元素集合{ann}，我们可以描述 ann 是 tom 的双亲之一，但是我们不能够直接描述 ann 是 tom 的

唯一的双亲，tom 的双亲属性依赖于程序中所描述的信息。同样我们也不能用双亲属性的空集来描述 tom 是一个孤儿。在这些语言中，我们只能用集合来表示部分信息，而不能说明集合值属性的完全信息。这种特殊的集合处理机制有其好处：在 LDL 和 COL 中不能层次化的程序在这些语言中是可以层次化的。

ROL[6]是一个演绎对象库语言，它集成了 Relationlog 对集合的处理机制。在 ROL 中，完全集所表示的信息是完全的，部分集表示完全信息的部分信息，这使得集合表示能力得到很大提高。

3 更新

在逻辑环境下研究更新机制是近来的一个热点，当前已有几种描述更新活动的理论框架和实现机制。在 Prolog 中，基本的更新原语是 assert 和 restract，assert 用来向数据库中插入一个简单子句，restract 则标识从数据库中删除一个简单子句。不幸的是更新语义并不是良定义的，更新对数据库的影响很难精确描述，往往依赖于过程语义。不同的计算规则将导致不同的计算结果。

Datalog/UT[7]扩展 Datalog 支持对外部谓词的更新。在 Datalog/UT 中，每一谓词都附有一个时序参数，称为时序点。更新序列可以由这些时序参数隐式表示。使用时序信息有几个好处：1）时间点是一个线性序关系；2）更新活动本质上与时间有关；3）更重要的是这样的语言与传统的 Datalog 在描述型语义上是一致的。这种机制实际上是一个历史数据库，可以很自然地支持对过去信息的查询。主要优点是可以利用 Datalog 描述型语义和自底向上的求值技术。

事务是现代数据库系统中一个关键性的概念，因为数据库应用中数据一致性是主要的约束条件。U_Datalog[8]是一个集成描述性查询和更新的逻辑语言，集成的基础是考虑了事务的行为特征。因此，事务处理分为两个阶段：①更新操作的收集，但更新动作并不执行；②将更新集作为一个批事务进行处理，如果有更新操作失败，则整个事务全部取消。

Obj_U_Datalog[9]可以看作是 U_Datalog 在理论粒度上扩充了对象概念，最主要的特点是提供了更新和事务语义，将 Obj_U_Datalog 转化为 U_Datalog。Obj_U_Datalog 的研究的目的是模拟协同对象集的协同工作，目前也未见有进一步扩展类和继承的概念。

关系数据库中广泛使用事务概念，通过将小的活动组合成大的活动可以降低数据库状态的数目，解决应用语义的一致性和原子性，一个事务可由两个集合描述：要删除的事实集 D 和要插入的事实集 I。新的数据库状态 New_db 可由旧的数据库状态 Old_db，D 和 I 定义：

New_db=Old_db-D+I

这个定义对应于先删除操作后插入操作。只有当事务提交时，才能执行删除和插入操作。Bonner[10]首先将事务概念引入到演绎数据库系统中，提出一个将更新和查询合一的逻辑框架 TR(Transaction Logic)。TR 从语法和语义两个方面对一阶逻辑进行了扩展，具有直观的模型论语义和合理、完全的证明理论。TR 提供两个操作子 del 和 ins，用来删除和插入事实。用一个新的逻辑连接词（串行合取连接词⊗）来描述子事务的执行序关系，下面的例子是用 TR 描述的：

raise(X):-(employee(E,S) ∧ Y=X+1)⊗del. employee(E,S)⊗ins. employee(E,S*Y). （例9）

在 TR 中，一个规则相当于一个命名的事务，这种命名机制可以用来描述嵌套事务，TR 的一个创新之处是在更高层次上为更新提供一个描述型语义。

Palopoli[11]讨论了对象数据库基于规则的更新语言，对象标识对于用户是不可见的，因此用户只能通过对象的属性来指定对象。由对象标识的本质决定，不同的对象可能有相同的状态。基于对象数据库的这种特性，作者定义几种更新操作子来表达可能的各种更新语义。例如：

$\leftarrow DEL_1$(person(X,[name:Tom,birthdate:[dd:1,mm:6,yy:58],sons:ϕ])).

其中 DEL_1 表示只删除一个满足值为[name:Tom,birthdate:[dd:1,mm:6,yy:58],sons:ϕ]的对象 person，因为在数据库中可能有多个 person 对象的值为[name:Tom,birthdate:[dd:1,mm:6,yy:58],sons:ϕ]，通过引入这些更新操作子，可以增强系统的更新活动的表达能力。但是该文并没有讨论对象的其它特征对更新活动的影响，也没有给出语言的模型论语义。

另一方面，将动态性引入到数据库中也受到研究者的重视。版本化是一个被普遍采用的方法，版本化粒度可以是整个数据库、关系或对象等。在关系数据库中，通过为每一关系增加一个额外的参数，对应于关系的版本，将状态引入到数据库中。

Krame[12]介绍了一个基于对象版本概念的利用

规则更新对象的方法，版本标识是一种特殊的对象标识，由表示更新类型的函数符号构造而成。通过版本化对象标识，我们可以回溯每一对象上的更新历史，因此对象版本具有时序特征：每一对象版本对应于一定时间段上的对象所处的状态。三个函数符号分别表示插入，删除和修改，这三个函数符号作用到对象表达式上构成版本化机制。例如：表达式＋(henry)．salary→50中的＋(henry)表示对象 henry 的一个版本。对象版本保存对象变迁的历史，这使得程序具有不动点语义。Lausen[13]为 M．Kramer 所设计的语言提供一个可能世界语义，这个语义是基于克里普克(Kripke)模型语义。与 Datalog/UT 的时序参数方法相比，版本化机制比较灵活，但有时并不很直观。

Ludascher[14]在 Statelog 中引入显式的状态概念，每一原子 R(X)扩展一个状态项 S 形成[S]R(X)，这样 Statelog 等价于版本化整个数据库。由于支持复杂状态项，Statelog 不仅支持线性时间，而且支持分枝或层次化状态空间。

May[15]对动态性概念作更进一步推广，将状态作为一类成员看待。在这个模型中，状态被抽象为类、对象、方法。这样所带来的好处是状态作为可标识的东西，而动态性的表现则通过程序中规则推理来实现。这种合一处理方法与 F-logic 的合一处理方法是一致的，因而可以很直观地与其集成。其直接的意义在于当为具有时序变化的应用进行建模时，状态有时必须是显式概念。将状态与演绎数据库集成除了可以提供灵活和清晰的建模能力外，还可以为数据库行为推理提供一个模型理论。用程序来描述和实现演绎和动态行为，那么我们就可以使用形式化方法来描述和验证应用的正确性和活性，一个例子是定义、实现和验证工作流系统。

小结 本文针对传统的演绎数据库存在的问题讨论了当前所采取的解决办法，通过扩充复杂结构化值，形成支持复杂对象的演绎数据库；进一步将对象标识、继承等面向对象的主要特征以及动态性概念引入到演绎数据库中。可以看到，随着系统功能的增强，系统也变得很复杂；这些研究的成果一方面促进了对演绎数据库的深刻理解，另一方面也使得演绎数据库表达能力越来越强。随着研究的深入和应用的需求，人们期望集对象模型的强大建模能力、演绎数据库的推理能力以及动态性和主动性为一体的数据库系统的出现。

参 考 文 献

[1] C. Beeri, et al., Set construction in a logic database language, J. Logic Programming, 10(3,4)1991

[2] S. Abiteboul and S. Grumbach, COL: A logic based language for complex objects, ACM TODS, 16(1) 1991

[3] M Liu, Relationlog: A typed extension to datalog with sets and tuples (extended abstract), In Proc. Intl. Logic Programming Symp., Portland, Oregon, U S. A, Dec 1995, MIT Press

[4] C. Lecluse and P. Rechard, O_2, database programming language, In Proc. Intl. Conf. On VLDB, Amsterdam, The Netherlands, 1989

[5] M. Kifer et al., Logic foundations of object oriented and frame based language, J. of ACM, 42, 1995

[6] M. Liu, ROL: A Deductive Object Base Language, Information System, 6(4)

[7] M. Liu and J. Cleary, Declarative updates in deductive databases, J. of Computing and Information, 1(1) 1994

[8] E. Bertino et al., Modelling Database Updates with Constraint Logic Programming, In 4th Intl. Workshop on Foundation of models and Languages for Data and Objects, 1992

[9] E. Bertino et al., Deductive Object Databases, In 8th European Conf. On Object-Oriented Programming, LNCS Vol 821, 1994

[10] A. J. Bonner and M. Kifer, An Overview of Transaction Logic, Theoretical Computer Science, 133(2) 1994

[11] Palopoli and R. Torlone, A rule-based update language for complex objects with identity, Data & Knowledge Engineering, 13, 1994

[12] M. Baudinet et al., Temporal Deductive Databases, In Book: Temporal Databases edited by A. U. Tansel et al. 1993

[13] G. Lausen, and G. Saacke., A Possible World Semantics for Updates by Versioning

[14] B. Ludascher et al., Nested Transactions in a Logical Language for Active Rules, In Proc. Intl. Workshop on Logic in Databases, San Miniato, Italy, 1996

[15] W. May et al., Integrating Dynamic Aspects into Deductive Object-Oriented Databases, In 3rd Intl Workshop on Rules in Databases Systems, Sweden, 1997