交互式面向对象的Prolog程序设计

Jarmo T. Alander等

摘 要

本文给出PC Prolog的规范模块CONSULT及其相应的数据结构，可用作简单交互式面向对象Prolog系统的内核。该系统最初是为CAD/CAM的应用而设计的，但也可用于许多与复杂层次数据结构有关的领域。

1. 引 言

1.1 背景

赫尔辛基技术大学工业自动化研究所开发了制造过程的计算机自动化，早期的工作主要涉及机器人建模和组装。这项工作的目的是试图开发一些用于计算机集成制造(CIM)和建模的计算工具和数据结构，特别注意解决用面向对象和基于规则的程序设计控制机器人高级组装所产生的问题，该领域基本的问题在于：1)产品和生产设计；2)产品和生产建模；3)生产和过程控制。

我们的目的是创造一个灵活而开放的系统，利用原则上能表示一个制造系统的所有重要计算方面和建模方面的通用而有力的数据库结构，支持解决上述三个基本问题。

1.2对象

将数据和计算结构化意味着使用面向对象的方法。一个对象是指一组数据及相应的处理方法，它用class形式来创建。class说明对象的属性及其类型是什么(即继承)，也说明对该对象允许什么操作(即抽象数据类型)

在Simula、C++ 和Smalltalk-80中可以找到使用对象的漂亮例子，这些系统大都具有"内嵌"型的类体系，将它们用于动态易变的样品生产线环境既困难又不实用，因为样品生产线环境中经常出现中断和修改，我们必须能够在线地存贮、修改、恢复系统状态。

1.3 增量式开发

在设计CONSULT过程中，特别考虑，使工具在尽可能保持通用和功能强大的同时又尽可能简单。简单的基本数据结构和解释性能可以使CONSULT成为增量式容错系统开发工具。它的数据对象在处理时不必绝对完整。Prolog数据库的相关特性有可能利用部分信息来存取数据。当我们要从突然而不可预料的软、硬件故障中恢复系统时，这一特点尤其使人高兴。

增量式系统开发的可能性还在于：通过在原数据库中定义新的属性和类，我们可以交互地随意改善它们的性能。特别是在样品生产系统中，当发现一个未预料到的新情况需要立即操作时，这尤为方便。

对我们来说，Simula和C++是两种最熟悉的面向对象型语言。在CAD/CAM方面的编程中使用面向对象的方法使人感到既合理又有效，但在研究和样品实验室环境中，这些语言的类层次太严格而不够灵活。根据这一事实以及Prolog数据库的灵活性和Prolog处理问题的能力，所以我们将Prolog选作样品系统开发语言。

2. 定 义

本节给出理解CONSULT基本结构所需

的一些定义。

2.1 表示法

程序文本以打字机字体表示。符号以小写字母开头，而变量以大写字母开头。另外，符号和名字也可由大、小写字母，数字和下划线组成。

2.2 原子

这里原子代表数据项自身，即数字，符号和字符串。

原子由字符组成。原子在Prolog的内部表示要么是符号，要么是字符串。在外部文件中可以将数字作为原子类型，但它们的内部表示仍是字符串。

例1 (原子) 以下是CONSULT的一些原子：

-a, b, long-atom, code12345;

-0, 1, 2, 10000000000000000000;

-"a very long atomic string"。

2.3 关系

原子是由两个称为d-关系和e-关系的Prolog谓词进行分组。

2.3.1 d-关系 d-关系给出原子的符号性质。结构d(obj, attr, val)表示符号obj具有称为attr的性质或属性，同时具有值val。一个原子可以有0个，1个或多个属性，而每个属性可以有1个以上的值。确切地说，d-关系表示由三个单词组成的一个句子，说明在本模型中的某个真理：d(bus, is, car)≡"bus is a car是真的"

2.3.2 e-关系 不同的原子由e关系分组或联接在一起。结构e(sym1,sym2)将sym1和sym2联接成一个偶对。每个原子可以与0或多个原子(包括其自身)相联接。e-关系解释哪些原子具有某种相邻关系：

e(house,garden)
e(garden,tulip) ≡house中有一garden，而garden里有tulip

也可以用d-关系来表示这种情况，但是，用e-关系表示对象间的数据更为自然，而对象内数据的表示则可留给d-关系。

3. 对象和类

3.1 片段

所有以相同原子作为首元素的d-关系组称为片段(fragment)。一个片段可以由1个或多个d-关系组成。

例2 (片段)符号p1有属性x=10, y=20和z=30，这可表示一个座标点(10, 20, 30)片段：d(p1,x,10) d(p1,y,20)d(p1,z, 30)

选择术语片段是用以指明片段中所包含的信息相互之间没有明确和系统化的类层次*)。

3.2 表示

使用d-关系和e-关系的内部数据表示也可用于外部数据表示。但我们可以用表而不用关系来节省一些内存。关系d(obj, attr, val)可以用表obj(attr(val))表示。例如，上例2中片段的外部表示为：外部表示，p1(x(10)y(20)z(30)。上述表示将节约30%字符。e-关系的外部代码是操作符：e(atom1,atom2)等价于atom1；atom2。如果合并两个具有相同符号甚至更长链的e-关系，那么将节约更多的空间。

3.3 对象

在实际应用中片段可以表示多或少的对象。使用片段所存在的问题是，它们原则上是唯一的并且不存在任何明显的通用方法使不同的片段相关。每个用户都可随意定义他自己的片段系统及其模型数据库。为了得到片段系统的一些结构，我们在它们中间定义一个类层次结构。

一个d(obj, is, class)形式的d-关系称为片段obj的句柄(handle)，片段obj被看作类class的一个对象。class可以是原子、片段或其最完整的形式，即对象。

这种对象-类结构的语义是，该类告诉我们这类的所有对象都具有什么属性。这由

*)片段本身也可能是良结构的有用数据，而类不如此。

形如d(Object, attr, Attribute)的d-关系实现。其中符号attr可用符号has代替，有助于记忆。

类A本身是一个对象，这意指存在另一个类AA，A是AA的一个对象。这个类-对象链可以一直传递下去，直至到达一些非常基本的类。这样，就得到一个类-对象层次结构，表明对象间的"家族联系"。

例3 (类层次)让我们定义一个矩形盒图box，盒图的四边都平行于座标轴。再假设盒图由三个座标区间(x, y, z)来表示。对此模型我们需要box-类和interval-类的定义：

```
box类
d (box,has,xi)
d (box,has,yi)
d (box,has,zi)
d (box,is,class)
```

```
interval类
d (interval,has,inf)
d (interval,has,sup)
d (interval,is,class)
```

盒图box对象b是：

```
对象b:
d (b,xi,i1)
d (b,yi,i2)
d (b,is,box)
```

其中对象i1,i2,i3是：

```
interval对象
d (i1,inf,1)
d (i1,sup,2)
d (i1,is,interval)
```

```
interval对象
d (i2,inf,5)
d (i2,sup,12)
d (i2,is,interval)
```

```
interval对象
d (i3,inf,3)
d (i3,sup,6)
d (d3,is,interval)
```

Prolog本身并未限制对每个对象只能定义一个句柄，我们可以随意定义任意多的句柄。比如可以定义一个对象属于两个或更多个类。这意味着该对象能够继承来自不同祖先链的性质，这叫多重继承。这种对象称为"混血儿(hybrids)"。

在大多数具有类结构的语言中，对象或实例是由类描述来创建的。在我们的系统中，可以按正、反两种次序进行定义，这在我们不清楚隐含的类层次但知道所包含的对象时非常方便，因为我们可以先定义对象的集合，然后归类成适当的类和超类，弄清这一系统。

例4 (隐喻)作为我们灵活分类模式之实用性的例子，我们来看看当从一种状态转到另一种状态时会发生什么根本变化。假设有一egg类，表示蝴蝶或飞蛾的卵。它的属性指明所有卵的特性。对象creature(动物)具有上述所有属性，还具有某些与特定卵有关的信息属性，比如繁殖地点和日期。当卵变成毛虫后一切将十分清楚。这时egg类中的数据便毫无意义了。我们现在看到的动物是由另一类叫做caterpillar(毛虫)来说明的。这时，我们是否应再构造另一对象来表示这个看来是新的动物呢？似是而非。对象creature中的数据并未完全失去它的意义，因为它代表的这一特殊标本的历史仍然存在，即使其性质和外表都已大大改变。在我们的系统中，只要将句柄d(creature, is, egg)改为d(creature, is, caterpillar), d(creature, was,egg)后，这个问题就很容易解决了。

这个例子来自自然界，但在工业生产过程中也可以找出大量类似的现象。比如原材料经多次加工后成为有用的产品，产品的外观自然与原有原材料很不相同，然而仔细观

实产品，它仍具有原材料的许多性质和属性。

3.4 属性

类层次通过继承机制来说明每一对象O的属性A(O)：$A(O)=\bigcup_{i\in pre(O)} A_i$

其中pre(O)是O的所有祖先的集合。除了属性定义外，一个类还可以有属性值定义。这些属性值是缺省的。如果某一属性有多于一个的属性值定义，那么将使用其中最低层次的属性值，即是给定对象最近祖先的属性值或者就是对象本身的属性值。

例5 （缺省值）比如定义一个car（汽车）类和对象tinycar（微型汽车）

```
car类
car (轮子(4)  门(2))
```

```
tinycar对象
tinycar(轮子(3)  门(1))
```

该tinycar只有3只轮子和1扇门，虽然典型的car有4只轮子和2扇门。一个属性值是一个原子，它可以指向任何片段或对象，也就是说，一个属性值其实可以有复杂的属性结构。在类结构中处理属性求值的Prolog例程objAttrVal显得非常简单：

```
objAttrVal谓词
objAttrVal(Obj,Attr,Val):-
        d (Obj,Attr,Val).
ObjAttrVal(Obj,Attr,Val):-
        d (Obj,is,Class).
Obj AttrVal(Obj,Attr,Val).
```

3.4.1 **属性类型** 属性类型不存在任何限制，我们可以以多种方式自由使用。比如可以通过分类来进行属性类型检查。很明显，此时应使属性及其值属于同一类。

另一方面，一个对象有很多值，这些值依赖于该对象符号在何处被作为属性使用。

例6 （盒图及interval）在"类层次"的例子中未给属性赋任何类型。它们应是：

```
box实的类型
xi (is(interval))
yi (is(interval))
zi (is(interval))
```

```
interval类的类型
inf (is(integer))
sup (is(integer))
```

类型的引入带来了对象的等价问题。在缺省的情况下，等价的对象（类型和值）属于同一类或就是同一对象。这叫强等价性。

类型和值之间的等价性有如下几类：

- 强等价：完全相同
- 较强等价：属同一类或类树
- 弱等价：具有相同的超类
- 不等价：无任何共同的祖先

3.5 例程和操作

各种计算机语言中都有数据和相应的处理例程。在面向对象系统中，数据和例程组合成对象，既具有数据的特性又有例程的特性。本系统中，例程或操作由routine类的对象表示，以表明所有例程的共性，即参数的个数和代码引用（即Prolog谓词）：

```
routine类
routine (is(class)
has (pred)   谓词
has (np)     参数个数
……           其他属性
 )
```

```
card routine
card (is(routine)
pred (object Card)
np (0)
 )
```

以上定义了称为Card的routine例程来计算基数，即某对象属性的个数。它不带任何参数。card的工作实际上由例程objectCard完成。

象其它类一样，routines可以形成复杂

的属性层次结构，它们可以有缺省值和其它继承的属性等。

4. CONSULT

处理对象的例程routines和类定义一起组成一个CONSULT包。

4.1 object-consult (File, Class, Obj—文件，类，对象)

CONSULT的主程序是oc，即object-consult，它从文件中将对象读入主存中的数据库。但是，与通常的Prolog中的Consult例程不同，oc不轻易失败，而是给用户打开一个交互式对话窗口，试图从大多数错误中恢复过来。事实上，我们可以只用oc例程便可建造模型！这种oc模型示于表1中。

对话使用菜单，调用TOOLBOX中的菜单处理程序。除了属性修改功能，object-consult还包括诸如自动分类等高级操作的例程。

4.2 eval (Obj, Routine, Paramenters)

类routine中的对象用eval例程求值。与object-consult一样，eval也不易失败，而是起动与用户的交互式对话过程。执行的Prolog例程实际由Obj和Routine两者的层次所引用的例程来决定。

4.3 undo(撤消)

在交互式设计和编程环境中能够撤消错误的操作是很有用的。通常这种操作称为undo。在本系统中，基于Prolog数据库，很容易实现一个简单而通用的undo操作，使工作数据库恢复到经过的一组数据库操作状态。这主要是assert和retract操作。undo操作假定有一个很长的文件，按出现的先后记录每个已被正确执行的数据库操作。undo(n)以相反的次序扫描该文件，并且相应执行n个相反的数据库操作：

assert(d(a, b, c)←→ retract (d(a, b, c))

4.4 数据隐蔽

建立在d-关系和e-关系之上的类结构是开放的，这意味着所有数据都可以被使用d-关系和e-关系的用户和PC Prolog例程所存取。数据隐蔽或保护有几种方法。方法之一是将关系分成两组或者更多组，其中一组作为d-e偶对为用户使用，而其它组为系统使用并保护起来。这种方法有一个固有的缺陷，使我们丧失了只有两个关系的数据库的简单性。

在具有良构造的大型系统中数据隐蔽是很重要的。保护通常给我们优先提供了所有的数据结构及其存取权限。以这种方式可以避免许多复杂的错误。但在构造灵活的原型时，严密的保护可能需要额外的修改工作和将来一些问题，特别是如果不能准确地知道所有的数据结构和所需的过程时更是如此。

4.5 效率

我们面向对象的系统建筑在Prolog数据库中的d-关系上，因而存在某些固有的效率问题。首先，空间开销较大，因为每个数据项都指向它所属的对象。其次，必须搜索每个对象的d-关系，很花时间。

如果使用外部表的表示方法来代替内部的d-关系表示法，则时间和空间的低效都可大大改进。然而，这样做将失去系统相关的灵活特性，这正是我们原型构造中系统的关键之处。

如果在生产过程中，效率是最关键的问题，那么可以将consult系统翻译成只使用外部表示的系统，但这将不可避免地限制某些灵活性。

表1 对象Consult过程的操作方式

文件	类	对象	对象Consult的操作(文件，类，对象)
自由[a]	自由	自由	与用户对话
自由	自由	固定	与用户对话；显示属性
自由	固定	自由	与用户对话；显示属性；显示分类的属性；
自由	固定	固定	与用户对话；显示属性；显示分类的、未分类的属性
固定	自由	自由	从文件中读对象[b].
固定	自由	固定	从文件中读对象；检测对象；
固定	固定	自由	从文件中读对象；检测对象；检测类；
固定	固定	固定	从文件中读对象；检测对象；检测类，检测类和对象的一致；

标准PASCAL的面向对象程序设计方法

Jonathan P. Jacky & J. Kalet

摘　要

本文讨论了用非面向对象程序设计语言进行面向对象的程序设计方法，并已成功地用于一个大型医用模拟程序。作者认为，面向对象的设计并不一定真正需要语言的支持，可以看作为一种设计方法学。这一观点不论对从事面向对象研究的人还是对从事应用程序设计的人都能有所启发和裨益，因为面向对象的设计方法已经越来越受到重视和欢迎，人们正在逐步采用这种方法进行软件开发。

近几年来，面向对象的程序设计已引起人们极大的热情。在这种程序设计中，程序基于的对象是类似于记录的数据结构。每个对象的类型，即类，与一系列特殊的操作有关。这些操作称为方法，类似于过程。当用消息调用对象时，方法即被实现。面向对象的程序设计的好处是由于对象与被模拟或计算的项的特性非常一致，因而程序设计较为容易；由于对象支持模块化和封装，因而程序错误较少；由于添加新的对象方便，因而程序易于扩展。

面向对象的程序设计通常与特定的程序设计语言有关。第一个这样的语言是Simula[1]，最有名的是Smalltalk[4]。最近，面向对象程序设计的支持已加入Lisp[10]，C[3,9]和Pascal的新的方言中。然而采用新的程序设计语言并不总是实用的，因为面向对象的方法并不真正需要语言的支持，它可视为一种设计方法学。基本原则是必须把程序中的每个数据项看作为某些对象的属性，并且只通过调用为该对象的类而定义的方法之一来访问。专门的语言能实现之，但在其它语言中也可以。有几种用于CLU和Pascal[8]，Ada[2]，甚至Fortran[5]语言的面向对象的方法学。我们的注意力在Pascal方法的开发，它已经成功地应用于一个癌症辐射治疗处理的大的(30000行)图形模拟程序[6]。

通常，这些系统表示患者的解剖模型和辐射源的排列，并且计算患者体内辐射剂量的分布。医务人员用这些系统寻求传递高辐射剂量只对肿瘤而不伤害健康组织的辐射源排列。典型地，用户用菜单选择射源，并用显示光标的移动把它们置于适当的

---

a) 从类中导出的每个对象可以备目录和文件引用，它们在I/O对象中的使用优先于File(文件)参数。

b)如果文件输入出错，则开始与用户对话。

5. 结　论

面向对象的方法是许多应用领域的一种极其自然的编程方法。特别是在CAD/CAM中更是如此，因为其中很自然地存在许多对象和分类。

本工作中，我们试图构造一个面向对象的系统，它十分灵活，足以很好地用于原型应用领域，这样的工作与日俱增，但并不十分要求效率。

Consult的优点有：1)灵活的类型；2)灵活而方便的数据存取；3)实时交互式数据库；4)开放型系统。

主要的缺点是：1)要求比表系统(LISP)多30%～40%的内存空间；2)运行时间长。

consult很容易引入现存的Prolog程序中，从而给出数据库的结构，同时得到简单而有力的用户界面。

参考文献(略)

〔彭　敏　黄素红译自《International Symposium for Young Computer Professionals》Aug. 21-23, 1989, Beijing, China〕