LPS程序的过程语义

赵春晓
(辽宁商业专科学校计算机系,锦州 121004)

李 磊
(吉林大学计算机系,长春 130023)

A PROCEDURAL SEMANTICS OF THE LPS PROGRAMS

Zhao Chunxiao
(*Liaoning Commercial College*, *Jinzhou* 121004)

Li Lei
(*Jilin University*, *Changchun* 130023)

Abstract To study the Non－1NF relational model, G. M. Kuper proposed the LPS language. Based the LPS language, a procedural semantics with mgu's are introduced, and we implamented the LPS language using meta－interpreter method.

摘要 为了研究Non－1NF关系模型G. M. ,Kuper[1]提出了LPS语言. 基于这种语言,本文给出了用最一般合一描述的程序的过程语义,并用解释方法实现了该语言.

§1. 引 言

在数据库理论的发展过程中,关系模型起着非常重要的作用. 但是,关系模型的表达能力和应用范围都是有限的. 为了扩展这种模型同时保留其优点,很多方法被提出,而嵌套关系(a nested relationship)(Non－1NF)模型是其中的一种. 在这种关系中,元组的成份可以是对象的集合,而不仅是简单的原子对象,作为查询语言的prolog不能很方便地描述这种复杂对象. 所以,必须使用一种针对Non－1NF关系的一种逻辑程序设计语言作为查询语言. 在[1]中,G. M. Kuper提出了一种含有集合对象的逻辑程序设计语言,称作LPS(Logic Programming with Sets)语言.

LPS语言是对逻辑程序设计语言的扩展. 在[1]中,G. M. Kuper给出了LPS程序的语法和语义,语义包括说明语义和过程语义. 但是,[1]中给出的过程语义是使用合一来描述归结过程的. 我们认为使用合一虽然能说明归结过程,但是合一不可能通过机器求得,只能人为地给出. 这样LPS语言就不可能在机器上实现,我们认为必须使用最一般合一才能实现归结过程.

§2. 语 法

我们先来介绍LPS语言的语法. LPS语言是基于两种逻辑的,相应于原子对象简记为a,相应于原子对象集合记为s. (注意:在LPS语言中不考虑集合的集合.)

定义1:一个LPS语言组成如下:

1. 谓词府号p_i^a,i=1,2,…,n_1. 还包括三个系统谓词符号$=^{aa}$,$=^{ss}$和$\in^{as}$.
2. 函数符号f_i,i=1,2,…,n_2.
3. 常量符号C_i^a,i=1,2,…,n_3.
4. 变量符号X_i^a,i=1,2,…,n_4.

本文1991年2月28日收到,1991年4月25日定稿. 本课题得到霍英东基金的部分资助.作者赵春晓,讲师,1991年硕士毕业于吉林大学,主要研究领域为微机应用. 李磊,教授,1991年博士后毕业于吉林大学,主要研究领域为数据库及逻辑程序设计.

© 中国科学院软件研究所 http://www.jos.org.cn

5. 逻辑联接词 ∨，∧，←.

6. 量词∀.

其中 q_i 是由 a 和 s 组成的串，它说明了谓词变元的种类，函数符号总是 a 到 a 的，r_i 是变量的种类，是 a 或是 s，常量都是 a 类的. 在实际使用中，函数和谓词的类型从上下文很容易看出，因而，文中不明确地指出其类型. 特别地，不区别两种相等谓词.

以下用 L 表示一个 LPS 语言，用小写字母 x,y,z 表示 a 类变量，用大写字母 X,Y,Z 表示 s 类变量.

定义 2：L 中的项 t 定义为：

1. 常量 C_i 是项，C_i 是 a 类的.

2. 变量 X_i 是项.

3. $f_i(t_1,\cdots,t_k)$是项，其中 f_i 是一个函数符号且每个 t_i 是 a 类的项，该项的类型是 a 类的.

4. 一个 a 类型项的有限集合$\{t_1,\cdots,t_n\}$是项.

定义 3：一个原子公式是一个形为 $P_i(t_i,\cdots,t_k)$的公式，其中每个 t_i 都是适当类的项.

定义 4：一个 LPS 语言 L 的解释 I 定义为：

1. 一个集合 D，另一个集合 $D^*\subseteq P(D)$. 其中 D^* 中元素的所有有限集合构成的.

2. 每个谓词符号$P_i^{q_i}$的一个指派 $I(P_i^{q_i})$. 如果 $q_i=a_1\cdots a_k$，那么 $I(P_i^{q_i})$一定是 $D_1X\cdots XD_k$ 的子集，其中每个 D_i 是 D 或者是 D^*，这取决于 q_i 是 a 还是 s.

3. 每个函数符号 f_i 的指派 $I(f_i)$，$I(f_i)$是一个从 $DX\cdots XD$ 到 D 的一个映射.

4. 每个常量 C_i 的一个指派 $I(C_i)\in D$.

5. 三个系统谓词的解释. 要求满足

(a) $I(=^{aa})$是 D 上相等关系.

(b) $I(=^{ss})$是 D^* 上相等关系.

(c) $I(\in)$是 D 中元素和 D^* 元素之间的属于关系.

定义 5：一个受限受称量词是一个形为 $(\forall x\in X)$的量词. 设 φ 是一公式，$(\forall x\in X)$的意义是$(\forall x)(x\in X\Rightarrow\varphi)$.

定义 6：一个扩展 Horn 子句是一个形为 $A\leftarrow(\forall x_1\in X_1)\cdots(\forall x_n\in X_n)(B_1\wedge\cdots\wedge B_m)$的公式，其中每个 B_i 都是原子公式，每个 x_i 是 a 类变量，每个 X_i 是 s 类变量.

定义 7：一个扩展 Horn 逻辑程序 P 是扩展 Horn 子句的一个有限集合.

定义 8：一个目标是一个形为$\leftarrow(\forall x_1\in X_1)\cdots(\forall x_n\in X_n)(B_1\wedge\cdots\wedge B_m)$的一个扩展 Horn 子句.

扩展 Horn 子句的意义非常类似于标准定义，如果 $y_1,\cdots,y_a$ 和 $x_1,\cdots,x_b$ 是在这样一个子句中的 a 类和 s 类变量. 通过对所有这些变量加上全称量词[2]，形成了子句的闭包，这非常类似于 prolog 语言中的程序子句，主要的不同就是子句体被一组受限全称量词所约束. 注意到，只要扩展 Horn 子句体的 $x_1,\cdots,x_n$ 是已知的话，那么子句体可能变成通常的 Horn 子句形式. 这样扩展的 Horn 子句，保留了 Horn 子句逻辑的大部分语义.

§ 3. 过程语义

文献[1]给出的 SLD 归结是用合一来描述的，并且认为不能用最一般合一. 我们认为使用合一虽然能描述归结过程，但不能在计算机上实现，而用最一般合一是完全可行的.

定义 9：一个替换 θ 是一个形为$\{x_1/t_1,\cdots,x_n/t_n\}$的集合—其中是互不相同的变量每个 t_i 是同类型的项.

定义 10：设 P 是一个程序，G_i 是目标 $\leftarrow(\forall x_1\in X_1)\cdots(\forall x_n\in X_n)(A_1\wedge\cdots\wedge A_m)$，并且 $R_{i+1}\in P$ 是子句 $A\leftarrow(\forall y_1\in Y_1)\cdots(\forall y_p\in Y_p)(B_1\wedge\cdots\wedge B_q)$. 如果

1. 存在某个 i 满足 $A_i\theta=A\theta$.

2. θ 是一个 mgu.

3. G_{i+1}是目标.

$\leftarrow[(\forall x_1\in X_1)\cdots(\forall x_n\in X_n)(\forall y_1\in Y_1)\cdots(\forall y_p\in Y_p)(A_1\wedge\cdots\wedge A_{i-1}\wedge B_1\wedge\cdots\wedge B_q\wedge A_{i+1}\wedge\cdots\wedge A_m)]\theta$，那么称 G_{i+1}是用 θ 由 G_i 和 R_{i+1}导出的.

注意到在执行完替换 θ 后，其中一些 X_i 和 Y_i 被集合变量所取代，另一些 X_i 和 Y_i 被形为 $\{t_1,\cdots,t_j\}$的项所取代.

© 中国科学院软件研究所 http://www.jos.org.cn

定义 11：设 P 是一个程序，G 是一个目标序列 $G_1,\cdots,G_n,\cdots$，子句序列 $R_1,\cdots,R_n,\cdots$，最一般合一序列 $\theta_1,\theta_2,\cdots,\theta_n,\cdots$组成. 其中每个 G_{i+1}是由 G_i 和 R_{i+1}使用 θ_{i+1}导出的.

2. PU{G} 的一个反驳是一个以空子句作为最终目标的有限推导.

3. 设 $\theta_1,\cdots,\theta_n$ 是 PU{G}反驳的一个 mgu 序列，θ 是乘积 $\theta_1,\cdots,\theta_n$ 中仅限于 G 中变量组成的替换，称 θ 是一个解替换.

4. PU{G} 的一个失败推导是以非空子句结束的推导.（注意：这里我们给出的是最一般合一，这不同于[1]的合一，而最一般合一可以通过合一算法求出.）

定理 3.1：设 P 是一个程序，如果 PU{←A}有反驳，解替换是 θ，那么 $A\theta$ 是 P 的一个逻辑结论. 这个定理就保证了我们给出的程序的过程语义是正确的. 我们用原程序设计的方法，在 AST－286 机上使用 Arity－PROLOG 语言设计实现了 LPS 的解释系统. LPS 的解释器包括三部分：1. 归结部分调用了一致化过程和转换过程以及系统谓词处理过程；2. 一致化部分调用一致化过程；3. 转换过程，将受限全称量词约束的扩展 Horn 子句转换通常的 Horn 子句. 图 1 和图 2 给出了 LPS 解释系统的构成.

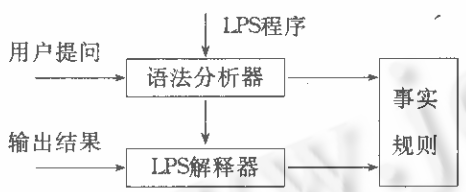

图1 LPS解释系统的构成

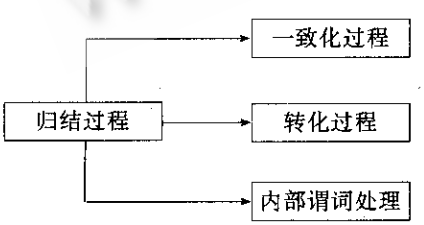

图2 LPS解释器

下面给出 LPS 解释程序的算法描述.

算法 3.1：

```
solve(true)
solve[(A,B)] : −solve(A),solve(B)
solve(A) : −system(A),call(A)
solve(A) : −clause(A,B),solve(B)
solve(A#B) : −translate(A,B),solve(B)
```

下面给出两个用 LPS 解释系统求解程序的例子.

例 4.1：计算 count(X,n)，程序及求解过程如下：

```
count(L,S) : − union−disj(L1,L2,L) ∧ count(L1,S1)
        ∧Count(L2,S2),
            S=S1+S2
count(L,S) : −singleton(L) ∧S∈L
singleton(L) : −(∀ x∈L)(∀ y∈L)(x=y)
union−disj(L1,L2,L) : −union(L1,L2,L) ∧disj(L1,L2)
disj(L1,L2) : −(∀ x∈L1)(∀ y∈L2)(x≠y)
? −count({1,2,3},S)
            S=6
            Yes
```

（注：这里我们将 union(L1,L2,L)作为 LPS 系统谓词）

例 4.2：下面程序系统及求解

```
likea(li,{swim,talk,wine})
likea(liu,talk)
likea(X,Y) : −likes(X,L) ∧Y∈L
? −likea(li,Y),likea(liu,Y)
    Y=talk
    Yes
? −likes(li,Y),likes(li,Y)
    No
```

结论：本文给出了用最一般合一描述的程序的过程语义，但这种方法是不完备的，当然程序员在使用该语言编程时，只需注意使子句右端的集合变量是已知的，该方法还是很实用的. 我们已经用解释方法实现了该语言，目前的系统还只是实验性的.

参考文献

[1] G. M. Kuper Logic Progrmming with Sets，A∈M Count on Principles Database Systemrp，Sun Diego，11－20，1987.

[2] J. Lloyd，Foundation of Logic Programming，Springer－Verlag，1984.

© 中国科学院软件研究所 http://www.jos.org.cn