事务逻辑对象库语言*

王修伦　孙永强

(上海交通大学计算机科学与工程系　上海　200030)

E-mail: wangxiulun@hotmail.com

摘要　对象封装了结构和行为,对象数据库为大规模复杂应用提供良好的建模方法和实现手段.对象与逻辑结合导致目前对演绎对象库的研究.然而,这些研究基本上针对对象的结构描述,而很少涉及到对象的动态行为的描述.该文重点研究对象的动态行为,分析对象特征:继承和重载对事务语义的影响,并设计了一个事务对象库语言 TOL(transaction object base language).首先分析 TOL 中的基本更新活动的特征,然后研究其事务的模型论语义.TOL 模型论语义基于通路结构.

关键词　事务逻辑,对象数据库,描述型语义.

中图法分类号　TP311

对象概念集成复杂结构和行为于一体,是复杂系统进行模块化设计的有力建模工具.演绎数据库具有很强的推理能力,对象和演绎数据库的集成形成演绎对象库.当前已有许多这样的集成语言和系统,如 F_logic[1], ROL[2], IQL[3]等.可是这些语言只是对对象的结构部分进行较全面的刻画,而对对象的行为特征则并未给予很大的重视.显然,这是很不符合实际应用要求的.另一方面,事务概念在关系数据库中占有极其重要的地位,可是在目前的演绎对象数据库中还很少见到这方面的报道.这主要是由于,当前对演绎对象库的研究都是针对复杂结构而进行的.因此,研究将对象结构、行为与演绎数据库集成具有重要的意义.

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,如条件演算、事件演算和事务逻辑.这些方法并没有提供一种描述复杂事务的能力.事务逻辑 TR[4]首先由 Bonner 提出,是一个将更新和查询合一的逻辑框架.TR 的一个很重要的语法扩充是引入一个新的逻辑连接词,利用它能够描述事务如何由原子活动组合而成.其另一个重要特征是具有一个描述型语义.

本文的工作主要是研究如何将事务、对象和逻辑集成在一个框架下,使得复杂结构建模、描述型事务说明被封装到对象内部,为用户提供一种更强的建模能力和系统实现方法.我们设计的这个语言称为 TOL(transaction object base language).在 TOL 中,我们也引入一个新的逻辑连接词⊗,用来描述复杂事务及其语义.TOL 中支持两种事务:一种称为方法事务,通过方法定义被封装在对象内;另一种称为全局事务,可以描述复杂事务以及对象间的动态交互关系.

继承和重载是对象模型的一个重要特征,文献[5,6]研究了继承和重载的语义.在 TOL 中,继承和重载为事务方法提供更灵活的实现机制,使得事务方法可以进行递增定义.这与 TR 中的事务概念是一个极其重要的区别.

TOL 具有一个描述型语义,它基于 Herbrand 语义模型.我们在 Herbrand 模型基础上建立一个通路结构作为 TOL 事务模型.

1　TOL 语法

1.1　TOL 语法介绍

在讨论 TOL 语法前,我们先给出一些符号的意义:(1) B:由系统定义的类符号集;(2) C:由用户定义的对象标识类集;(3) A:属性名集;(4) M:方法名集;(5) D:B 中符号所表示的域的并;(6) V:变量集;(7) O:对象标识符集;(8) S:由系统维护的一个表,包含数据库中对象被引用的信息;(9) P:全局事务谓词集.

---

* 作者王修伦,1970 年生,博士,主要研究领域为对象数据库.孙永强,1931 年生,教授,博导,主要研究领域为计算机软件理论,新型语言,并行处理.

本文通讯联系人:王修伦,上海 200030,上海交通大学计算机科学与工程系

本文 1997-06-09 收到原稿,1997-09-19 收到修改稿

定义 1.1. TOL 中的项集 Ts 为

(1) $D \cup O \subset Ts$;(2) $o_1, \ldots, o_n$ 为项,则 $\{o_1, \ldots, o_n\}$ 为集合项.

没有变量的项为基项.在 TOL 中,项的概念可以充当对象作用.

定义 1.2. 类型符号集合 T_Set 为

(1) $B \cup C \subset T\_Set$;(2) $c_1, \ldots, c_n$ 为类型符号,则 $\{c_1, \ldots, c_n\} \in T\_Set$ 为集合类型.

定义 1.3. TOL 中类描述为下述形式.

(1) 如果 c, c' 为类符号,则 c isa c' 为类格说明,我们称 c 为 c' 的子类,c' 为 c 的超类,并且关系 isa 在集合 C 上是一个偏序关系,isa * 表示 isa 的自反,传递闭包;

(2) 如果 a 为属性符号,c 为类符号,c_1 为类型符号,则 $c[a \Rightarrow c_1]$ 表示 c 的属性 a 以 c_1 作为其类型,并且对于 c 的任意子类 c',都有 $c'[a \Rightarrow c_1]$;

(3) p 为一个谓词符号,$c_1, \ldots, c_n$ 为类型符号,则 $p(c_1, \ldots, c_n)$ 为谓词说明;

(4) 如果 m 为一个方法符号,c 为类符号,$c_1, \ldots, c_n, c_{n+1}$ 为类型符号,则 $c[m@c_1, \ldots, c_n \Rightarrow c_{n+1}]$ 为方法说明,当 c_{n+1} 不为 Void 时,m 为函数方法(或演绎方法),如果 c 为 Void,则 m 为事务方法,并且上述形式可简写为 $c[m@c_1, \ldots, c_n]$;

所有类描述集构成数据库模式.由于继承关系,模式可能存在冲突,已有许多文献给出冲突消解方法,在下面的讨论中,我们假设模式不存在冲突.在 TOL 中,存在两种方法:一种为演绎方法,另一种为事务方法.为简单起见,本文只考虑事务方法,而不考虑演绎方法.

定义 1.4. TOL 中的对象表达式定义如下:

(1) o 为对象标识,c 为类符号,则 $o:c$ 为简单对象表达式,表示 o 是 c 的直接实例;$o::c$ 也是简单表达式,表示 o 是 c 的直接或间接实例.这里有一个限制:任意一个对象标识 o 只能有一个类 c,o 为 c 的直接实例,表示为 class(o);

(2) o 为对象标识,a 为属性,o_1 为一个对象,则 $o[a \rightarrow o_1]$ 为简单对象表达式;

(3) $o[a_1 \rightarrow o_1]; \ldots; o[a_n \rightarrow o_n]$ 为对象表达式,则 $o[a_1 \rightarrow o_1, \ldots, a_n \rightarrow o_n]$ 为复合对象表达式.不含变量的对象表达式称为基对象表达式.

TOL 中任何事务都是由基本更新单元或其他事务组成的,下面,我们先介绍 TOL 中的基本更新单元.

定义 1.5. TOL 中有两种基本活动操作子:*ins* 和 *del*.*ins* 表示向数据库中插入操作,*del* 表示从数据库中删除操作.$ins.o:c$ 表示对象创建;$del.o:c$($del.o::c$)则表示对象删除;$ins.o[a \rightarrow o_1]$ 表示设置对象 o 的属性 a 的值为 o_1;而 $del.o[a \rightarrow o_1]$ 表示删除对象 o 的属性 a 值.在下面的讨论中,我们假设活动操作子只用在简单对象表达式前面.

我们对这两种操作子的应用作一个限制:对象属性的更新由事务方法来实现.这种限制使得对象的状态可以通过属性访问,但是对对象状态的更新只能通过对象提供的事务方法调用实现,任何其他对象都不能直接改变其内部的状态.这样,查询可以具有灵活的表达方式,而又能保证对象状态变化语义的正确性,体现对象模型的封装性.

定义 1.6. TOL 中也引入一个新的逻辑连接词$\otimes$,通过它,我们可以更方便地描述应用语义.

$\beta \otimes \alpha$ 表示 α 紧接着 β 执行.如果 α 或 β 失败,则整个 $\beta \otimes \alpha$ 事务也失败,对数据库的更新活动需要执行卷回操作.$\beta \wedge \alpha$ 表示不确定事务表达式,其语义等价于 $\beta \otimes \alpha$ 或 $\alpha \otimes \beta$.

定义 1.7. 定义活动表达式:如果 α 为对象表达式,则 $ins.\alpha, del.\alpha$ 为简单活动表达式.如果 α 为 $o[m@o_1, \ldots, o_n]$,其中 o 为对象标识,m 为事务方法名,$o_1, \ldots, o_n$ 为对象,则 α 称为事务方法表达式,我们把 $c::o[m@o_1, \ldots, o_n]$ 称为类型化事务方法表达式.如果 α 为 $p(o_1, \ldots, o_n)$,p 为全局事务谓词,$o_1, \ldots, o_n$ 为对象,则 α 称为全局事务表达式.如果 β, α 为活动表达式或为对象表达式,则 $\beta \otimes \alpha$ 为复合活动表达式,则 $\beta \wedge \alpha$ 也是活动表达式,表示它们的执行序关系是任意的.不含变量的活动表达式称为基表达式.所有的对象表达式和活动表达式统称为表达式.

这里需要解释一下,引入活动表达式的目的是为了建立模型论语义.活动表达式可以看作特殊的对象表达式,活动表达式表现的是状态的变迁,其满足的概念需要用一种新的结构来解释.下面将讨论这个问题.

定义 1.8. 在 TOL 中,有两类事务规则,一类为定义在对象内部的事务方法规则,另一类为全局事务规则,这两类规则通过定义符号来区别.对于事务方法,规则头具有下列形式:

$$c::X[m@o_1, \ldots, o_n],$$

而全局事务规则的头为一个事务谓词.

1.2 例 子

为了更具体地讨论 TOL 语言的语义,我们先看一些例子.

例1：CLASS *person*;*student*;*department*. *student* isa *person*.

person[*name*⇒*STRING*], *person*[*age*⇒*INT*], *person*[*chaA*@*INT*,*INT*]

student[*dept*⇒*department*], *student*[*chaB*@*STRING*,*STRING*],

department[*courses*⇒*STRING*],

(*) *person*::*X*[*chaA*@*Y*,*newY*]←*del*. *X*[*age*→*Y*]⊗*ins*. *X*[*age*→*newY*].

(**) *student*::*X*[*chaB*@*Y*,*newY*]←*del*. *X*[*dept*→*Y*]⊗*ins*. *X*[*dept*→*newY*]

f←*ins*. *computer*:*department*⊗*ins*. *computer*[*course*→*Database*]⊗*ins*, *John*:*student*

⊗*ins*. *John*[*age*→20]⊗*ins*. *John*[*dept*→*computer*]⊗*ins*. *Simth*:*person*⊗*ins*. *Smith*[*age*→30].

在例1中，对象的创建是由 *ins* 来完成的. 一个对象必须在创建后才能被引用，因此，对象 *computer* 必须在被创建后才能被对象 *John* 引用，用⊗描述这种事务的执行的先后关系. 在这个例子中，为了更灵活地支持非更新事务，对象的静态性质可以通过其属性直接访问，比如可以直接查询 *John*[*dept*→*X*]，如果 *f* 执行后，则返回的结果 *X*=*computer*. 对象的所有静态属性值构成了对象的可见状态，另一个需要说明的是，继承机制为事务提供一个共享实现，如事务 *John*[*chaA*@*X*,*Y*]，由于在类 *student* 中并没有定义这个事务方法，其执行将继承其父类 *person* 中的同名事务规则.

下面的例子说明了事务方法重载为事务实现提供一种进化的方法.

例2：CLASS *empl*;*stu_emp*. *stu_emp* isa *employee*.

empl[*salary*⇒*INT*], *empl*[*age*⇒*INT*], *empl*[*Inc_sal*@*INT*].

empl::*X*[*Inc_sal*@*Y*]←*ins*. *X*[*salary*→*Z*]⊗*ins*. *X*[*salary*→*Z*+*Y*].

(*) *stu_emp*::*X*[*Inc_sal*@*Y*]←*ins*. *X*[*salary*→*Z*]⊗*ins*. *X*[*salary*→*Z*(1−10%)+*Y*].

通过继承，*stu_emp* 可以重新定义事务方法 *Inc_sal*，使得作为 *stu_emp* 对象调用 *Inc_sal* 时，与 *empl* 中的对象调用同名事务方法所产生的效果是不同的. 因此，事务方法在类中通过继承重载了其父类同名方法，这大大增强了系统建模能力. 重载机制保证不同对象在调用同名事务时，将激活相应的事务规则. 在这个例子中，事务方法重载是完全覆盖其父类的事务方法，可是，在逻辑环境下，仍然存在另一种事务重载情况，即动态事务方法重载. 例如，将例2中的规则(*)修改为下面的形式：

stu_empl::*X*[*Inc_sal*@*Y*]←*X*[*age*→*Q*]⊗(*Q*>35)⊗*del*. *X*[*salary*→*Z*]

⊗*ins*. *X*[*salary*→*Z*(1−10%)+*Y*),

则如果存在两个对象 *st*1，*st*2 都为 *stu_emp* 的实例，并且 *st*1[*age*→20]，*st*2[*age*→40]，显然，对象 *st*1 调用事务方法 *Inc_sal* 时，在类 *stu_emp* 定义的事务规则并不能成功，我们希望继承能为事务的实现提供一个缺省实现机制，即在调用失败的情况下，其父类的同名事务方法的定义规则为其提供一个实现. 这样的语义是很清晰的. 支持这样的语义将使系统具有更大的灵活性. 当然，在动态重载存在时，属于同一类的对象在执行同名事务时将激活不同的事务规则.

1.3 约束子事务

在 TOL 中，为了保证对象删除语义的正确性，由系统建立和维护一个表 $\mathcal{S}$，S 中包含所有对象的引用信息. 对象创建将使系统自动在 S 中建立初始化信息，对象的删除则依赖于 S 表，S 表对用户是不可见的，对 S 表的操作也由系统自动完成. S 表对 TOL 的一致性维护具有很重要的作用.

我们已经定义了 TOL 中的基本活动表达式，这些活动表达式对于用户而言是原子操作，可是对于系统而言这些活动表达式的意义并不是简单的插入和删除活动. 为了维护语义的一致性，系统为每一个基本活动表达式建立一个约束子事务，显然，这些约束子事务对用户而言是不可见的. 因此，在 TOL 中，事务具有两种粒度，一个是面向系统的约束子事务，另一个是用户描述的事务. 前者的正确性由系统维护，后者的正确性需要用户自己维护. 我们将在第2节具体定义这些约束子事务，这里仅讨论对象删除问题.

在 TOL 中，支持显式删除对象. 由于对象可能被引用，这使得对象删除所需要的一致性维护比较复杂. 当用户提交一个删除对象事务时，系统将隐式地执行一个完整性维护子事务. 如 $del.o;c$，则系统将执行一个如下的子事务：查询 $S(o,X)$，如果 $X=0$，表明没有其他对象引用，删除操作可以执行，删除 o 的所有状态，即 $del.o[a_1\rightarrow o_1]\otimes\ldots\otimes del.o[a_n\rightarrow o_n]$，其中 $a_i(i=1,\ldots,n)$ 为 o 的所有属性. 否则，表明该对象正被其他对象所引用，这个事务将不会对当前数据库进行任何更新操作. 当删除一个对象的状态时，如果这个对象的状态是对其他对象的引用，则需要执行一个约束子事务. 如 $del.o[a\rightarrow o_1]$，如果 o_1 为另一对象标识，则约束子事务为：$del.S(o,X)\otimes ins.S(o,X-1)$，表明对对象 o 的引用数目减少一个. 从上面的讨论可以看到，基本活动表达式的约束子事务可能又隐含其他约束子事务，我们可以证明这

种嵌套约束子事务嵌套深度是有限的,即不会形成框架问题.在下面的讨论中,除非特别说明,事务概念指的都是用户事务.

我们已经介绍了TOL语言的语法,并且非形式地描述了它的语义.在TOL中,事务在两个方向上进行描述:可以由对象创建、单元更新、串行逻辑连接词⊗和不确定操作子∧来直接描述,或者通过继承机制,间接地继承已有的事务定义.一个很重要的概念是引入了一个系统表S,对于每一基本活动表达式,系统通过表S,自动执行一个约束子事务,保证对象删除语义一致性,而不会导致删除异常.

2 TOL模型论语义

在上一节中,我们介绍了TOL的语法,通过一些例子,非形式地描述了TOL的语义,并简单地描述了基本活动表达式的意义.本节研究TOL的事务模型语义.TOL事务模型基于Herbrand语义模型,然而与传统的Herbrand模型不同的是,事务的引入以及事务的执行将导致数据库产生负影响.为了支持事务的模型论语义,我们在Herbrand模型基础上引入一个通路概念,在通路概念基础上建立TOL的模型论语义.下面首先定义一些基本概念,然后引入通路概念和TOL模型.

定义 2.1. K为一个数据库模式,Herbrand域U_K为所有对象集合.

定义 2.2. K为一个数据库模式,Herbrand基B_K为所有基简单对象表达式集合.

定义 2.3. B_K的一个子集S_B是一致的,如果$o[a\rightarrow o]\in S_B$,则$\neg\exists\ o[a\rightarrow o']\in S_B$,使得$o\neq o'$.

定义 2.4. S_B为B_K的一个子集,类在S_B中的实例定义如下:

(1) 一个值为S_B中值类的实例,如果这个值在该值类所表示的域中.

(2) 一个对象标识符o为类c的实例,如果$o:c\in S_B$或$o::c\in S_B$.

定义 2.5. K为一个数据库模式.S_B为B_K的一个子集,φ为一个基简单对象表达式,则φ在S_B中是良类型化的,如果下列条件之一成立:

(1) $\varphi=o:c$(或$o::c$),且o为c在S_B中的实例.

(2) $\varphi=o[a\rightarrow o_1]$,则$\exists\ c[a\Rightarrow c_1]\in K$,使得$o,o_1$分别为$c,c_1$在$S_B$中的实例.

置换的概念与传统逻辑程序中的概念相同.下面我们定义对象表达式满足概念.

定义 2.6. I为B的一个一致子集,基对象表达式φ在I下满足的概念($\vdash$')可定义为:

(1) 如果$\varphi=o:c$,$I\vdash'\varphi$当且仅当$\varphi\in I$.对于任意c isa * c',$I\vdash'o::c'$.

(2) 如果$\varphi=o::c$,$I\vdash'\varphi$当且仅当存在c isa * c',$I\vdash'o::c'$(或$I\vdash'o:c'$).

(3) 如果$\varphi=o[a\rightarrow o']$,$I\vdash'\varphi$当且仅当$\varphi\in I$.

(4) 如果$\varphi=\neg\psi$,$I\vdash'\varphi$当且仅当$I\not\vdash\psi$.

定义 2.7. S表是形式为$S(o,n)$的一个集合.其中o为对象标识,n为一个非负整数.S表是有系统维护的,用户不能对其进行操作.S表是一致的,如果$S(o,n)\in S$,则$\neg\exists\ S(o,n')\in S$,使得$n\neq n'$.在下面的讨论中,我们假设$S$表是一致的.将$B_K,S$的并集称为扩展的Herbrand基$EH_K$.

定义 2.8. K为一个TOL程序模式,则对于K的一个数据库,D为B_K的子集I和S的子集S_t的并,满足下面的条件:

(1) I,S_t是一致的.

(2) 如果$o:c\in I$,则S中必然存在$S(o,n)\geqslant 0$.

(3) 如果$S(o,n)\in S$,则S_B中有且仅有n个对象表达式引用了对象o.

我们说对o'的引用可以用如下的形式说明:如果$o[a\rightarrow o']\in I$,o'为对象,则对象表达式引用o,或$o[m@o_1,\ldots,o_n]\in I$,如果$\exists\ o_i$为对象标识($1\leqslant i\leqslant n$),$o_i=o$.

定义 2.9. D为一个数据库,它所对应的状态为一个扩展的Herbrand模型集合SD,并且满足$\forall\ sd\in SD, sd\vdash'D$.

在第1节中,我们已经非形式地介绍了约束子事务概念.对于用户而言,约束子事务保证了基本活动表达式的执行使得当前数据库在维护语义正确性的前提下转变为新的数据库.基于此,我们形式地定义基本单元活动表达式的意义:用$\alpha\equiv d_1\rightarrow d_2$表示在当前数据库为$d_1$时,基本活动表达式$\alpha$的执行将导致数据库转变为数据库$d_2$.$I_1,I_2,s_1,s_2$分别为$d_1,d_2$对应的对象表达式集合和$S$表.则

(1) $ins.o[a\rightarrow o_1]\equiv d_1\rightarrow d_2$,如果$\neg\exists\ o[a\rightarrow o'_1]\in I_1$,则$I_2=I_1\cup\{o[a\rightarrow o_1]\}$,若$o_1$为对象标识,则$s_2=s_1-\{S(o_1,n)\}\cup\{S(o_1,n-1)\}$.

(2) $del.o[a\rightarrow o_1]\equiv d_1\rightarrow d_2$,如果 $o[a\rightarrow o_1]\in I_1$,$I_2=I_1-\{o[a\rightarrow o_1]\}$. 若 o_1 为对象标识,则 $s_2=s_1-\{S(o_1,n)\}\cup\{S(o_1,n-1)\}$.

(3) $ins.o:c\equiv d_1\rightarrow d_2$,如果 $\neg\exists\ S(o,n)\in s$,则 $I_2=I_1\cup\{o:c\}$,$s_2=s_1\cup\{S(o_1,0)\}$.

(4) $del.o:c\equiv d_1\rightarrow d_2$,如果 $\exists\ S(o,n)\in s$,$n>0$,$d_2=d_1$,否则设对象 o 的当前状态 $o[a_1\rightarrow o_1],\ldots,o[a_n\rightarrow o_n]$,并且存在数据库序列 $d'_1,\ldots,d'_n$,使得 $del.o[a_1\rightarrow o_1]\equiv d_1\rightarrow d'_1$,$del.o[a_2\rightarrow o_2]\equiv d'_1\rightarrow d'_2,\ldots,del.o[a_n\rightarrow o_n]\equiv d'_{n-1}\rightarrow d'_n$,为 $I_2=I'_n-\{o:c\}$,$s_2=s'_n-\{S(o_1,0)\}$.

(5) $\mathrm{del}.o::c\equiv d_1\rightarrow d_2$,当且仅当 $\mathrm{del}.o:\mathrm{class}(o)\equiv d_1\rightarrow d_2$.

(6) $\mathrm{ins}.o::c\equiv d_1\rightarrow d_2$,当且仅当 $\mathrm{ins}.o:c\equiv d_1\rightarrow d_2$.

我们定义了基本活动表达式的意义,可以看到,基本活动对应两个数据库间的关系. 一个事务可能包括多个基本活动,因此一个事务可能使数据库产生一系列变化,这就促使有下面的定义:

定义 2.10. 一个通路是一个状态序列. PS 为所有状态集合,则 $\langle ps_1,\ldots,ps_n\rangle$ 为一个长为 n 的通路 π,如果 $ps_i\in PS\ (I=1,\ldots,n)$. 我们定义 $\mathrm{first}(\pi)=\langle ps_1\rangle$,$\mathrm{last}(\pi)=\langle ps_n\rangle$. 基于 PS 的所有通路集合表示为 $\mathrm{Path}(PS)$. 一个通路 π 可以被划分为任意两个子通路 π_1,π_2,$\pi_1=\langle ps_1,\ldots,ps_i\rangle$,$\pi_2=\langle ps_i,\ldots,ps_n\rangle$. 我们用 $\pi=\pi_1\cdot\pi_2$ 表示.

定义 2.11. 活动集合 $Acts$ 为所有基活动表达式集合,S_Acts 是 $Acts$ 的一个一致子集,如果满足下列条件:

(1) $\mathrm{ins}.o[a\rightarrow o']\in S\_Acts$,则 $\neg\exists\ \mathrm{ins}.o[a\rightarrow o'']\in S\_Acts$,使得 $o'\neq o''$,

(2) $\mathrm{del}.o[a\rightarrow o']\in S\_Acts$,则 $\neg\exists\ \mathrm{del}.o[a\rightarrow o'']\in S\_Acts$,使得 $o'\neq o''$.

所有 $Acts$ 的一致子集的集合表示为 $Con(Acts)$.

定义 2.12. 一个通路结构 PS 为三元组 $\langle U,PS,I_{\mathrm{path}}\rangle$,$U$ 为 Herbrand 域,PS 为所有状态集合,I_{path} 为一个映射:$\mathrm{Path}(PS)\rightarrow Con(Acts)\cup 2^{E_H^H}$,并且满足:

(1) $I_{\mathrm{path}}(\langle ps\rangle)\in ps$,

(2) 如果 α 为一个基本活动表达式,且 $\alpha\equiv d_1\rightarrow d_2$. ps_1,ps_2 为 d_1,d_2 对应的状态,则 $\alpha\in I_{\mathrm{path}}(\langle ps_1,ps_2\rangle)$.

下面我们给出基于通路结构的满足概念,在讨论满足概念之前,需要分析一下事务方法规则重载定义.

定义 2.13. 重载. 若有两个事务规则,其形式如下:

$$r_1=c_1::X[m@o_1,\ldots,o_n]\leftarrow \mathrm{Body}_1,$$

$$r_2=c_2::X[m@o'_1,\ldots,o'_n]\leftarrow \mathrm{Body}_2.$$

r_1 由类 c_1 定义,r_2 由类 c_2 定义,且 $c_1\ \mathrm{isa}+c_2$. 如果存在两个基 θ_1,θ_2 置换,使得 $\theta_1X=\theta_2X$,$\theta_1o_i=\theta_2o'_i\ (i=1,\ldots,n)$ 成立,则 θ_1r_1 可能重载 θ_2r_2.

定义 2.14. 设 M 为一个通路结构 $\langle U,PS,I_{\mathrm{path}}\rangle$,$\pi\in\mathrm{Path}(PS)$,$\theta$ 为一个基置换,任一表达式在 M 和通路 π 下满足($\vdash$)概念和不满足($\nvdash$)概念定义为:

(1) $M,\pi\vdash_\theta\beta$ 当且仅当 β 为任意对象表达式,$I_{\mathrm{path}}(\pi)\vdash\theta'\beta$,如果 $\beta=o:c$(或 $o::c$),$c\ \mathrm{isa}+c'$,则 $I_{\mathrm{path}}(\pi)\vdash\theta' o::c'$,

(2) $M,\pi\vdash_\theta\beta$ 当且仅当 β 为任意活动表达式,并且 $\beta\in I_{\mathrm{path}}(\pi)$.

(3) $M,\pi\vdash_\theta\beta\otimes\Phi$ 当且仅当存在 π 的一个划分,$\pi=\pi_1\cdot\pi_2$,使得 $M,\pi_1\vdash_\theta\beta$,$M,\pi_2\vdash_\theta\Phi$,

(4) $M,\pi\vdash_\theta\beta\wedge\Phi$ 当且仅当或者 $M,\pi\vdash_\theta\beta\otimes\Phi$,或者 $M,\pi\vdash_\theta\Phi\otimes\beta$.

(5) $M,\pi\vdash_\theta\beta\rightarrow\beta$ 当且仅当 $M,\pi\nvdash_\theta\beta$.

(6) $M,\pi\vdash_\theta r$,r 为一个事务方法规则 $\varphi\leftarrow\delta$,$M,\pi\vdash_\theta\delta$ 蕴含 $M,\pi\vdash_\theta\varphi$ 或者存在另一规则 r',基置换 θ' 和通路 $\pi'\in\mathrm{Path}(PS)$,$\mathrm{first}(\pi)=\mathrm{first}(\pi')$,$\theta_1r_1$ 可能重载 θ_2r_2,使得 $M,\pi'\vdash_{\theta'}r'$,并且 $M,\pi'\vdash_\theta\varphi$,$M,\pi\vdash_\theta\delta$.

定义 2.15. K 为一个数据库模式,P 为一个程序,M 为一个通路结构 $\langle U,PS,I_{\mathrm{path}}\rangle$,$r$ 为 P 中任一规则,任意基置换 θ,任意通路 $\pi\in\mathrm{Path}(PS)$,使得 $M,\pi\vdash_\theta r$,则 M 是 r 的一个模型. 若对于 P 中任意规则,M 都是其模型,则 M 为 P 的一个模型.

K 为一个数据库模式,P 为一个程序. P 的通路模型并不具有唯一性,它们的差别体现在对通路的解释上. 与经典逻辑程序语义一样,可定义通路模型的交和并.

定义 2.16. K 为一个数据库模式,P 为一个程序,$M_1=\langle U,PS,I_{\mathrm{path1}}\rangle$,$M_2=\langle U,PS,I_{\mathrm{path2}}\rangle$ 为 P 的两个通路结构,则它们的交和并定义如下:

$M_1\cap M_2=\langle U,PS,I'_{\mathrm{path}}\rangle$,$\forall\ \pi\in PS$,$I'_{\mathrm{path}}(\pi)=I_{\mathrm{path1}}(\pi)\cap I_{\mathrm{path2}}(\pi)$,

$M_1\cup M_2=\langle U,PS,I''_{\mathrm{path}}\rangle$,$\forall\ \pi\in PS$,$I''_{\mathrm{path}}(\pi)=I_{\mathrm{path1}}(\pi)\cup I_{\mathrm{path2}}(\pi)$.

引理 2.1. 两个通路结构的交和并仍然是通路结构.

证明:只需证明交和并满足通路结构的定义.由通路结构的定义和定义 2.16 可证明.

定理 2.1. K 为一个数据库模式,P 为一个程序,M_p 为所有 P 的通路结构,则$(M_p,\cup,\cap)$为一个完全格.

证明:由引理 1 证明. □

定义 2.17. 通路结构间的包含关系$\subseteq$.P 为一个程序,M_1,M_2 为两个通路结构,$M_1=\langle U,PS,I_{path1}\rangle$,$M_2=\langle U,PS,I_{path2}\rangle$,$M_1\subseteq M_2$,如果$\forall\ \pi, I_{path1}(\pi)\subseteq I_{path2}(\pi)$.

3 结　论

本文初步研究了事务概念与对象结合的方法,设计了一个事务对象库语言.将描述风范与具有过程性的事务概念集成到一个语言框架上,为系统建模和实现提供一种手段.与现有的研究相比,TOL 着重分析了事务与对象特征的相互影响,而这在许多演绎数据库中都未提及.

我们还拓广经典的模型语义,为事务建立一个基于通路结构的模型论语义,加深对事务执行的理解.

参考文献

1 Kifer M, Lausen G, Wu J. Logic foundations of object oriented and frame based language. Journal of ACM, 1995,42(4):741～843

2 Liu Meng-chi. ROL: the deductive object base programming language. Information System, 1996,21(5):431～457

3 Abiteboul S, Kanellakis P C. Object identify as a query language primitive. In: Proceedings of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1989. 157～179

4 Bonner A J, Kifer M. An overview of transaction logic. Theoretical Computer Science, October 1994,133(10):205～265

5 Abiteboul S, Lausen G, Uphoff H *et al*. Methods and rules. In: Proceedings of ACM SIGMOD International Conference on Management of Data. New York, 1993. 32～41

6 Dobbie G, Topor. On the declarative and procedural semantics of deductive object oriented systems. Journal of IIS, 1995,4(2): 193～219

Transaction Logic Object Base Language

WANG Xiu-lun　SUN Yong-qiang

(*Department of Computer Science and Engineering　Shanghai Jiaotong University　Shanghai* 200030)

Abstract　With the structure and behavior encapsulated in object, object database supplies large complex applications with better modeling capacity and implementation utilities. Deductive object database results from the integration of object and logic. However, deductive object database focuses more on object structure description and less on object behavior description. In the present paper, the authors study the object dynamic behavior analyze the interaction among inheritance, overriding and transaction, and gives a language called TOL (transaction object base language). The elementary actions in TOL are analyzed first, and then a model semantics is built based on path structure for transactions.

Key words　Transaction logic, object database, declarative semantics.