一种改进的主动规则的系统执行模型*

李庆忠, 王海洋, 姜跃平, 马绍汉, 董继润

(山东大学 计算机科学系,山东 济南 250100)
E-mail: lqz@sdu.edu.cn
http://www.sdu.edu.cn

摘要: 主动规则及规则处理机制一直是主动数据库研究的薄弱环节,主要表现为:(1) 语义不够丰富;(2) 缺乏基础语义;(3) 缺乏对规则处理的层次化和结构化的描述.为弥补上述不足,对规则处理机制赋予了丰富的规则延伸语义,根据语义的要求,用规则处理树层次化和结构化地描述规则处理过程对系统状态的影响.新的系统执行模型和扩展后的事务使我们能够提出一种支持丰富语义的规则处理算法.与其他规则处理算法相比,该算法实现了新语义,并利用了递归技术,能更好地吻合规则处理过程.

关 键 词: 主动数据库;ECA-规则;天折力度;规则处理
中图法分类号: TP311 文献标识码: A

主动数据库系统一直是一个热门课题,出现了大量的研究成果[1~4].大多数系统都使用了主动规则的概念.主动规则建立在事件(event)-条件(condition)-动作(action)三要素之上,又称为 ECA 规则.规则的基本运行方式是,一旦系统检测到规则事件的发生,就在特定的时刻检查规则的条件,如果条件满足,则执行相应的动作.

文献[1]以 IBM Almaden 研究中心的实验系统 Starburst 为背景,其系统执行模型中包含了操作块、改变、改变效应等概念,并定义了复合操作,描述了规则执行对数据库状态的影响,给出了规则执行算法.但是,它没有考虑不同的 EC(event-condition)耦合方式对规则执行的影响,而且只是把规则动作的执行语义视同为传统的事务操作块.

文献[2]讨论了与规则处理有关的各种语义,它提出了一个一般性的框架,试图把 Starburst,Ode 和 Postgres 等实验系统的规则语义都包容进来.这使得它所提出的规则处理方法能支持许多不同的规则执行语义,具有一定的广泛性,但也使得其算法只能是相当地笼统.它没有具体讨论规则执行对系统状态的影响,也没有考虑引入嵌套事务的处理方法.

因此,以往的主动规则及规则处理机制主要的不足表现为:(1) 语义不够丰富;(2) 缺乏基础语义;(3) 缺乏对规则处理的层次化和结构化的描述.

在本文中,我们的目标是力图弥补上述不足.我们采取了如下措施:(1) 对规则处理机制赋予丰富的规则延伸语义;(2) 根据规则语义的要求,我们用一种新形式(规则处理树)层次化和结构化地描述规则处理过程对系统状态的影响,在一定程度上弥补了上述第 3 个环节的不足;(3) 我们给出了引入规则之后的系统执行模型,重新定义了系统状态、执行块、改变、效应等重要概念(原定义参见文献[1]).新的系统执行模型和扩展后的事务使我们能够提出一种支持丰富语义的规则处理算法.与其他规则处理算法[1~3]相比,这种算法实现了我们提出的新语义,并利用了递归技术,能更好地吻合规则处理过程.

* 收稿日期: 2000-02-25; 修改日期: 2000-07-26
基金项目: 国家自然科学基金(69473013);国家"九五"攻关项目(96-B03-04-03-12)

作者简介: 李庆忠(1965－),男,河北威县人,博士,教授,主要研究领域为数据库理论与应用,信息系统;王海洋(1965－),男,山东文登人,博士,教授,博士生导师,主要研究领域为数据库系统;姜跃平(1970－),男,山东烟台人,博士,讲师,主要研究领域为数据库系统;马绍汉(1938－),男,山东平度人,教授,博士生导师,主要研究领域为智能算法;董继润(1935－),男,山东高清人,教授,主要研究领域为数据库系统.

本文第 1 节描述了规则的语义并对夭折力度进行了扩展.第 2 节讨论了系统的执行模型.第 3 节讨论了规则处理及其算法.最后是总结及展望.

1 规则语义及对事务模型的改进

ECA 规则的基本语义是,规则在其事件发生时被触发,而后在特定的时刻系统检查规则条件,如果条件满足,系统将执行规则的动作.下面先给出几个先导概念的定义.

定义 **1.1(规则实例(rule instance))**. 规则实例是一个三元组$\langle r,tid,ts\rangle$,其中 r 是实例所对应的规则,tid 是实例所属的事务标识,ts 为实例生成时的系统时间,称作触发时间戳.

每当一条规则被事务触发时,系统就产生一个相应的规则实例作为本次触发的系统记录和规则处理对象.

定义 **1.2(原始实例、级联实例和父实例)**. 规则实例可以由用户事务直接触发产生,也可以由其他规则实例的执行来触发产生.前者称为原始实例,后者称为级联实例.触发某级联实例的规则称作该级联实例的父实例.

定义 **1.3(触发事务)**. 若事务 T 的某一步操作触发了规则 r 并产生了相应的规则实例 i_r,则 T 称为 i_r 的触发事务.

一个规则实例与其触发事务之间有着紧密的联系.首先,规则执行一般(分离式规则除外)作为触发事务的一部分;其次,在发生严重错误时,规则执行可能要求整个触发事务夭折与回滚.

在规则处理过程中,如果存在多个等待立即处理的规则实例,规则处理就必须从中选择一个实例最先进行处理.规则选择策略如下:

规则选择策略.根据应用的业务要求,指定存在业务关联的规则之间的优先级.优先级高者优先,在优先级相同的实例中,先根据触发时间戳进行先触发者优先的选择,如果仍无法区分,则随机选择.

在规则处理过程中,并非对系统的所有规则都进行规则选择.对于开始于事务提交点的延迟式规则实例的规则处理来说,规则选择的范围是所有尚未得到执行的,由本事务及其级联实例所触发的延迟式规则实例.对于面向立即式规则实例的规则处理来说,规则选择的范围仅限于最近触发的立即式规则实例.

迄今为止,几乎所有的工作都把被执行的主动规则的动作部分完全视同为传统的事务操作块.但是,这种语义不能满足主动数据库的应用要求.一个突出的问题是规则动作中的夭折命令的作用.下面给出给出 3 种夭折.

定义 **1.4(夭折力度)**. 夭折力度即主动数据库中的夭折类型,共有 3 种:一种是传统的夭折语义,称为绝对夭折;第 2 种是局部夭折;第 3 种是级联夭折.

定义 **1.5(局部夭折)**. 只当满足某个条件时,规则动作部分失败,而不是全部失败,这称为局部夭折.

定义 **1.6(级联夭折)**. 为了表示主动规则之间的相互依赖关系,在某些特定条件下,用户需要取消规则动作以及触发它的逻辑操作块——级联实例中其父实例的动作,这称为级联夭折.

引入局部夭折和级联夭折对传统事务进行改进,能使规则处理机制支持丰富的规则语义.从表达能力上来看,这可以弥补传统事务的许多不足.

2 系统执行模型

为了更好地描述主动规则语义和规则处理机制,我们为系统执行模型重新定义了系统状态、执行块、改变及效应等基本概念,描述了效应的具体作用.

定义 **2.1(系统状态)**. 主动数据库的系统状态是一个四元组$\langle D,R,I_R,D_R\rangle$,其中 D 为数据库状态;R 为规则库状态,即当前存在的主动规则的集合;I_R 为等待执行的立即式规则实例集合;D_R 是等待执行的延迟式规则实例集合.

系统状态这一定义在传统的数据库状态之外扩展了规则库状态和规则实例.

定义 **2.2(原始状态、入口状态和中间状态)**. 触发事务开始时的系统状态称为原始状态;规则执行开始时的系统状态称为入口状态;规则执行过程中某一时刻的系统状态称为中间状态.

对于一个触发事务来说,其原始状态在整个事务周期保持不变,入口状态在规则执行期间保持不变,而中间

状态可能被规则或其级联实例的动作执行块变更.

定义 **2.3(执行块)**. 执行块由一系列特定的数据操作组成,并且这一操作序列的执行不可分割(可视为原子操作).

执行块总是以提交、夭折或者触发产生了立即式规则实例的操作为结束.

定义 **2.4(改变(transition))**. 一个执行块的执行称作一个改变.改变的粒度介于原子操作和事务之间.改变是由原子操作组成的一个逻辑整体,而对事务来说,它又具有原子性.改变使系统从一个状态迁移到另一个状态,可表示为 $S \xrightarrow{T} S'$.

定义 **2.5(效应(effect))**. 改变对系统状态的影响称作效应,可表示为一个七元组$\langle I_D,D_D,I_{IR},D_{IR},I_{DR},D_{DR},T_a\rangle$,其中$I_D$是改变向数据库中插入的元组的集合;$D_D$是改变从数据库中删除的元组的集合;$I_{IR}$是改变触发所产生的立即式规则实例;$D_{IR}$是改变所删除的立即式规则实例;$I_{DR}$是改变触发产生的延迟式规则实例;$D_{DR}$是改变所删除的延迟式规则实例;而$T_a$的取值分别为0,1,2,3,分别表示无夭折、局部夭折、有限级联夭折和绝对夭折.改变所更新的每个元组可以替代表示为一个删除元组(旧值)和一个插入元组.在有了效应这一定义以后,由于改变T而使系统状态发生的迁移就可以用T的效应ε_T表示: $S \xrightarrow{\varepsilon_T} S'$.

定义 **2.6(执行块的分层(stratum))**. 执行块b的分层记为$stra(b)$,规定如下:(1) 若b是触发事务的执行块,则$stra(b)=0$;(2) 若b是延迟式规则实例的执行块,则$stra(b)=1$;(3) 若b是延迟式规则实例r的执行块,且是由分层为i的执行块所触发的实例,则$stra(b)=i+1$.

执行块的分层将决定其效应的夭折类型对系统状态的影响.设执行块b所属事务的原始状态为$S0$.块b的执行所对应的改变记为b_t,因而改变b_t对应的效应为ε_{b_t},对任意状态S施加效应ε_{b_t}将产生一个新状态S'.

· 若$stra(b)=0$,则有:

(1) 若$\varepsilon_{b_t}.T_a=0$,则 $S'.D=S.D\cup\varepsilon_{b_t}.I_D-\varepsilon_{b_t}.D_D$;

$S'.IR=S.IR\cup\varepsilon_{b_t}.I_{IR}-\varepsilon_{b_t}.D_{IR}$;$S'.DR=S.DR\cup\varepsilon_{b_t}.I_{DR}-\varepsilon_{b_t}.D_{DR}$;

(2) 若$\varepsilon_{b_t}.T_a>0$,则 $S'.D=S0.D$;$S'.IR=S0.IR$;$S'.DR=S0.DR$.

· 若$stra(b)>0$,设b为规则实例r的执行块,并设r的入口状态为Sr,则有:

(1) 若$\varepsilon_{b_t}.T_a=0$,则 $S'.D=S.D\cup\varepsilon_{b_t}.I_D-\varepsilon_{b_t}.D_D$;

$S'.IR=S.IR\cup\varepsilon_{b_t}.I_{IR}-\varepsilon_{b_t}.D_{IR}$;$S'.DR=S.DR\cup\varepsilon_{b_t}.I_{DR}-\varepsilon_{b_t}.D_{DR}$.

(2) 若$\varepsilon_{b_t}.T_a=3$,则 $S'.D=S0.D$;$S'.IR=S0.IR$;$S'.DR=S0.DR$.

(3) 若$\varepsilon_{b_t}.T_a=1$,则 $S'.D=Sr.D$;

(3a) 若$r\in Sr.IR$,则 $S'.IR=Sr.IR-\{r\}$;$S'.DR=Sr.DR$.

(3b) 若$r\in Sr.DR$,则 $S'.IR=Sr.IR$;$S'.DR=Sr.DR-\{r\}$.

(4) 若$\varepsilon_{b_t}.T_a=2$,则有:

(4a) 若$stra(b)=1$,则 $S'.D=S0.D$;$S'.IR=S0.IR$;$S'.DR=S0.DR$.

(4b) 若$stra(b)>1$,设r是规则实例rp的级联实例,并设rp的入口状态为Srp,则有 $S'.D=Srp.D$;

若$rp\in Srp.IR$,则 $S'.IR=Srp.IR-\{r\}$;$S'.DR=Srp.DR$.

若$rp\in Srp.DR$,则 $S'.IR=Srp.IR$;$S'.DR=Srp.DR-\{r\}$.

· 在以上各种情况中,都有$S'.R=S0.R$成立,因为操作块中并不允许包含规则的模式修改命令.

效应的上述作用语义是规则处理算法的基础.

3 规则处理及算法

3.1 多规则处理

我们已经扩展了传统的系统状态概念,使之包含了规则库和规则实例集合的状态.现在我们要考虑规则处理过程对系统状态的影响.在本文中,我们假定规则动作里不含有“绝对夭折”命令,因为如果发生绝对夭折,规则处理将终止,并回到原始状态,这一语义显而易见.

单一规则处理的执行语义比较简单、直观,本文不再详细描述.文中主要讨论系统中存在多条 ECA 规则时的规则处理.与单一规则处理类似,触发事务的执行块或事务提交命令启动一个规则处理过程;不同的是,可能有多个规则实例在等待处理,需要做规则选择.从物理效果上来看,系统状态仍然是随着事务或规则执行块的效应而"线性"地迁移;但从逻辑语义上说,规则执行块与触发执行块的关系类似于嵌套事务的子事务与父事务.尤其是对于级联实例而言,父实例的执行块启动它的处理过程相当于事务启动一个子事务,当这种子事务执行完之后,父事务才能够继续执行.

多规则实例的执行块的效应依次作用在当前系统状态上,其执行顺序可以用多叉树结构直观地加以表示,我们称这种树为规则处理树.

图 1 中给出了一棵规则处理树的例子.图中以"*r*"开始的节点为实例节点,代表一个规则实例的执行;以"*B*"开始的节点为块节点,代表一个执行块;根节点 *T* 是事务节点,代表触发事务.每个实例节点左侧的兄弟节点是触发该实例的执行块的节点.该图表示了在某一规则处理点对等待执行的两个规则实例 *r*1 和 *r*2 的处理过程.图 1 中的 *r*1 节点的实例的动作由 *Br*11 和 *Br*12 这两个执行块组成,且 *Br*11 触发产生 *r*11 和 *r*12 这两个立即式实例.

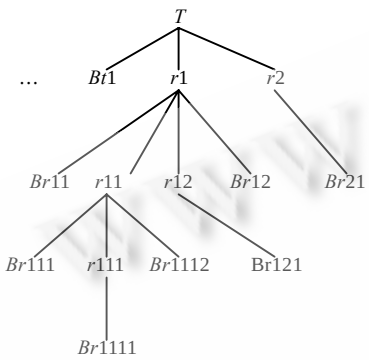

Fig.1 Rule processing tree

图 1 规则处理树

如果某一执行块触发产生了延迟式实例 *rd*,那么在中间或级联处理点,*rd* 将不出现在规则处理树上;而在终结处理点,*rd* 将作为最高层实例节点的兄弟出现在图中,其位置则由规则选择策略动态决定.如果执行块触发产生了分离式实例,则其执行将自成一棵规则处理树.需要注意的是,树中的叶节点都是块节点,而中间节点都是实例节点.

对规则处理树的节点进行后序扫描(即先自左至右扫描各个兄弟节点,在扫描兄弟节点时,又首先扫描该节点的子女块结点,当兄弟节点扫描完毕以后才扫描父节点),所得到的扫描序列中的块节点即构成了规则处理的执行块序列.在规则处理过程中,这些执行块就是按照这一顺序改变系统状态的.对如图 1 所示的情况,后序扫描的到的块节点序列为

$$Bt1 \rightarrow Br11 \rightarrow Br111 \rightarrow Br1111 \rightarrow Br1112 \rightarrow Br121 \rightarrow Br12 \rightarrow Br21,$$

而扫描序列中的实例节点则是各规则实例的执行序列.

规则处理过程中产生的系统状态迁移与执行块序列完全对应.对于图 1 的情况,系统状态的迁移可表示为

$$S0 \ldots \xrightarrow{EBt1} S1 \xrightarrow{EBr11} S2 \xrightarrow{EBr1111} S3 \ldots \xrightarrow{EBr21} Se \longrightarrow \ldots$$

3.2 规则处理算法

本节给出 ECA 规则的基本处理算法.该算法能够支持前面所述丰富的规则语义.为了能够使描述更加形式化和过程化,这里将避免过深地涉及实现细节.该算法具有以下特点:

(1) 串行执行.每一条规则的执行效应均作用于当前状态,所产生的新状态作为下一条规则执行的入口状态.

(2) 当有多条可执行规则时,遵循第 1 节的规则选择策略进行选择,有潜在的随机性.

(3) 每一规则执行的入口状态都是当前系统状态.规则条件总是在当前系统状态下进行检查.

(4) 所有规则的 CA(condtion-action)耦合方式都假定是立即式的.

(5) 支持 3 种不同的夭折方式.

(6) 规则的执行是可中断的.

算法 3.1. 基本规则处理算法

```
(1.0)  exec_rules(当前应处理规则集合,夭折标志,原始实例标志)
(1.1)  begin
(1.2)      while 当前应处理规则集合<>ϕ do
```

(1.3) 从给定的规则实例集合中选择一条规则 *r*,在进行选择时,将执行顺序组织成规则处理树,对规则处理树的节点进行后续扫描,按这种顺序进行规则选择.

```
(1.4)        exec_rule(r,夭折标志);   /* 执行选出的规则 */
(1.5)          if 发生级联夭折或绝对夭折 then
(1.6)            if 是原始实例 then
(1.7)              需要恢复成原始状态
(1.8)            end if
(1.9)            return;
(1.10)         end if
               /* r 不再在规则选择范围内 */
(1.11)     从当前应处理规则集合除去规则 r
(1.12)     重新计算当前应处理规则集合的值
(1.13)   end while
(1.14) end.

(2.0)  exec_rule(r,夭折标志)
(2.1)  begin
(2.2)    将夭折标志赋值为 0
(2.3)    将 r 从等待执行的规则实例集合中去除
(2.4)      if 在当前状态下规则 r 的条件不为真 then
(2.5)        return
(2.6)      end if
(2.7)    保存入口状态
(2.8)      while 从给定规则实例 r 的动作中析出下一个执行块,若成功 do
(2.9)        计算执行块的效应
(2.10)       将执行块效应的夭折标志赋值给夭折标志
(2.11)     if 夭折标志=0 then   /* 无夭折产生 */
(2.12)       把给定的无夭折效应作用于当前状态,将返回所得到的新状态置为当前状态
(2.13)         if 块效应的立即式规则集合<>ϕ  then   /* 触发了立即式规则 */
                 /* 处理立即式规则 */
(2.14)           exec_rules(块效应的立即式规则集合,夭折标志,false);
(2.15)         end if
(2.16)       end if
(2.17)       if 执行块要求做局部夭折或者级联实例要求做有限级联夭折 then
(2.18)         恢复当前状态为入口状态
(2.19)       end if
               /* 执行块或级联实例要求做绝对夭折 */
(2.20)       if 夭折标志为绝对夭折 then
(2.21)         终止规则处理并返回绝对夭折标志
(2.22)       end if
(2.23)   end while
(2.24) end
```

定理 3.1. 算法 3.1 能够使所有的规则实例得以执行,并且该递归算法是可以结束的.

证明:首先要证明所有的规则实例能够得以执行.由于规则的执行可能触发其他规则的执行,这样形成级联规则.

当级联的深度为 0 时,规则的集合为语句 1.0exec_rules 中的参数:当前应处理规则集合.语句(1.2)对所有未处理的规则进行循环,语句(1.3)在规则处理树中进行后续扫描,遍历选择出一个规则.而在语句(1.4)调用 exec_rule 时没有触发新的规则,此时,算法能够处理所有的规则.

假设级联的深度为 n+1,算法能够处理级联深度小于或等于 n 的所有深度中的所有规则.下面要证明算法 3.1 也能够处理深度为 n+1 的所有的规则.由假设可知,对于深度为 n 的情况,可以处理所有规则,因此,当深度为 n+1 时必定执行语句(2.14)调用过程 exec_rules 来执行深度为 n 的级联规则所触发的规则.因此,语句 1.1exec_rule 过程的参数〈当前应处理规则集合〉是某个深度为 n 的规则触发的深度为 n+1 的规则的集合.语句(1.2)对所有未处理的规则进行循环,语句(1.3)在规则处理树中进行后续扫描,遍历选择出一个规则.而在语句(1.4)调用 exec_rule 时没有触发新的规则,此时,算法能够处理所有的规则.

因而所有的规则实例能够得以执行.

然后证明算法 3.1 是可以结束的.当级联的深度为 0 时,规则的集合为语句 1.0exec_rules 中的参数:当前应处理规则集合,并且这些规则实例都没有触发新的规则,因此,语句(1.4)调用 exec_rule 以后首先检查条件是否满足,如果不满足,则返回,所以算法的运行可以结束;如果满足,则进入语句(2.8)循环,级联的深度为 0 说明不再触发新的规则,因此语句(2.13)条件中的块效应的立即式规则集合为空,因此将不调用(2.14)中的过程,将正常返回到过程 exec_rules,因此 exec_rules 将结束执行.

假设级联的深度为 n+1,在级联深度等于 n 时,算法能够结束.下面要证明当级联深度为 n+1 时,算法能够返回到深度为 n 的递归层次之中,因而也就证明了算法可以结束.深度为 n 执行到(2.13)的 if 语句,其条件中的块效应的立即式规则集合不为空,因此要调用 exec_rules,在执行 exec_rules 时要调用(1.4)执行 exec_rule,由于当前层次为 n+1,因此当它再调用 exec_rule 时,(2.13)中的块效应的立即式规则集合肯定为空,因此可以返回到 exec_rules.当 exec_rules 执行完毕并返回到深度为 n 的语句(2.14)之后,即回到了深度为 n 的递归层次之中,因而算法可以结束. □

4 总 结

规则语义,即规则处理机制,一直是主动数据库研究的薄弱环节.本文力图对传统的语义进行扩充,使我们能够采用一种新形式——规则处理树——层次化和结构化地描述规则处理过程对系统状态的影响,重新定义了系统状态、执行块、改变和效应等重要概念.我们提出的规则处理算法实现了我们提出的新语义,并利用了递归技术,以更好地吻合规则处理过程.本文提出的规则处理算法已经在山东大学自行设计的主动面向对象数据库实验系统 SDAOODB(active object-oriented database system of Shandong University)[5]中得以实现.

规则并发语义是规则处理中另一个以解决规则选择冲突为目的的重要课题.今后我们将以本文的研究工作为基础,进一步研究并发执行语义和相应的算法.

References:

[1] Widom, J., Finkelestein, S.J. Set-Oriented production rules in relational database system. In: Hector, Garcia-Molina, ed. Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data. ACM Press, 1990. 254~265.
[2] Fraernali, P., Tanca, L. A structural approach for the definition of semantics of active database. ACM TODS, 1995,20(4):414~471.
[3] Ludascher, B., May, W., Lausen, G. Nested transactions in a logical language for active rules. Technical Report, Report-80, Institute of Informatik, University of Freiburg, 1996.
[4] Ludascher, B., May, W., Lausen, G. Referential actions as logical rules. In: Mendelzon, A., Ozsoyoglu, Z.M., eds. Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. Tucson, Arizona: ACM Press, 1997. 217~224.
[5] Chen, Ning. The technique and implementation of object-oriented database SDAOODB. Computer Engineering, 1998,24(2):6~7 (in Chinese).

附中文参考文献:

[5] 陈宁.面向对象主动数据库 SDAOODB 的技术与实现.计算机工程,1998,24(2):6~7.

An Improved Active Rule System Executing Model*

LI Qing-zhong, WANG Hai-yang, JIANG Yue-ping, MA Shao-han, DONG Ji-run

(*Department of Computer Science, Shandong University, Ji'nan*, 250100, *China*)

E-mail: lqz@sdu.edu.cn

http://www.sdu.edu.cn

Abstract: Active rules and its processing mechanism have been the weaknesses in the research area of active database for a long time. The weaknesses were mainly represented: (1) The semantic of rules is not rich enough; (2) Lack of fundamental semantic; (3) Lack of hierarchical and structural description of rule processing. In this paper, in order to make up the above weakness, a rich extending semantic is given to the rule processing mechanism. A rule processing tree is represented to describe the influence of rule processing to the state of system structurally and hierarchically. Based on the new system executing model and the extended transaction model, a rule processing algorithm that supports rich rule semantic can be represented. Comparing with other similar algorithm, the algorithm implements the new semantic. The using of recursive technique makes the algorithm coincide with the rule processing.

Key words: active database; ECA-rule; abort type; rule processing

---

* Received February 25, 2000; accepted July 26, 2000

Supported by the National Natural Science Foundation of China under Grant No.69473013; the Key Science-Technology Project of the National 'Ninth Five-Year-Plan' of China under Grant No.96-B03-04-03-12