计算机科学1991№.2

基于法则的面向对象系统模型

章远阳　　（北京大学计算机系）

提　要

本文从模块的受控存取入手，描述了基于法则系统（Law-Governed System, LGS）的概念和模型，其中的法则（Law）用来控制大型系统的功能，结构及其演变。作为LGS的一个实例，我们给出了刻画面向对象程序设计（OOP）方法特性的一组法则，按OOP的不同语义层次，构成了一个基于法则的面向对象系统（Object-Oriented LGS, OOLGS），并给出了OOLGS中多重继承的一种实现方法，最后提出了LGS实现和应用中的有关问题。

目前，软件工程技术及其应用的发展已经促使人们将注意力集中到大型系统的设计上，实践表明，大型系统设计的关键在于模块的组装技术[5]，即如何描述模块间的界面控制，以保证模块间的存取访问是受控的。这不仅保证了大型程序的正确性，而且也是降低其开发成本的技术保证，为此，人们在程序设计语言的基础上建立了一些用于描述模块连接的子语言，即MIL (Module Interconnection Language)，如Mesa, Pebble[5]等。这种方式的明显特点是：由于MIL和基语言的耦合过于紧密，导致了由MIL描述的控制法则也密切依赖于语言本身，这样的法则说明性差，并且不易修改，显然不利于反映大型系统持续演变（evolving）的特征[3]。

要使控制方式能灵活地随系统的演变而变化，就必须将描述控制方式的法则(易变化部分)和系统本身（相对稳定部分）分离，使得系统本身的功能、结构受到法则的制约，由于法则部分可说明性强，并可随外界变化修改，因而支持了系统的演变，按这一思想构造的系统称为基于法则的系统（LGS）。

一、基于法则的系统

1. LGS概念

LGS的基本概念或特征可以概括为：

1）LGS=System+Law，System中的所有活动必须受控于Law，Law刻画了System的构造、使用中必须遵守的一切约束（dispine）。

2）LGS不仅控制系统的功能、结构和使用，还控制（反映）系统的演变，甚至支持Law本身的功能、结构、使用和演变。

3）Law的制约作用是显式的，严格的，强制的。

Law不仅支持模块间的受控访问，如封装机制，模块间的受控互连机制，不同类型的模块保护机制等，还支持各种模块组织方式，其中的一个典型就是本文着重讨论的模块继承，同时由于Law中可以显式地说明在何条件下，何人能对系统的哪一部分进行何种操作，因此它能控制（regulate）、传播对模块的任何修改，进而支持系统的演变。

2. LGS实例：分层模块系统

LGS的目的是支持大型的、持续演变系统的构造，这里我们以分层（layered/hierarchical）模块系统为例进行说明。

根据分层模块系统（设为S1）的定义[2]，系统S1应满足下列法则的要求，或者说应受下列法则的制约：

R11：将S1中的模块分为若干层次，顶层为第

i层，位于第n层的模块只能调用第n层或第n+1层的模块中所定义的过程（procedure）。

R12：本组法则不可改变。

L1：分层模块系统的法则

从L1我们可以看到：

1）L1是非形式化的。在后面介绍了LGS的形式模型之后，我们可以给出L1的形式化表示，以便机器实现。

2）R11描述了系统S1中模块的组织结构和受控访问形式，使得受控于L1的任何系统都能准确地反映分层模块系统的含义。

3）R12说明了L1本身应遵守的法则，即L1的自描述法则，它表明：无论S1如何变化，都"万变不离其宗"，总是严格地服从R11的制约，这种LGS称为静态（fixed）LGS，其中的Law在系统的整个生命期内是不变的；如果Law在系统的生命期内可以有条件地修正、改变，则相应LGS称为动态（Variable）LGS。

3. LGS模型

可以将LGS定义为一个二元组，LGS＝（System，Law），其中System是系统中相互通讯的模块集合，Law是制约模块间通讯的法则集合，以下为方便起见，对System中模块属性的引用表用ADT中的点记法（如M.level表明引用模块M中的属性变量level），对Law中的法则采用*Proglog*的*Horn*子句形式表示。

模块通讯采用消息发送的形式，即(m，t)，其中m表示消息正文，t表示发送的目标模块。由于我们在此讨论的LGS对System的制约作用是面向消息发送者的（sender-sensitive），因此需要一函数send将(m，t)转换成带有消息发送者S的询问信息send（s，m，t）交给Law处理，Law将其处理成对Law中规则集的询问?send（s，m，t），即根据Law中的法则来判定s能否向t发送消息m，结果可能是：

1）能，此时直接将消息m交给模块t，即deliver（m，t）。

2）不能，此时询问?send（s，m，t）和Law中的法则合一失败，返回fail，表明消息m被阻塞（blocked）。

3）不能直接向t发送消息m，必须将消息m修改为m′或/和将目标模块转换成（rerouting）t′，将m′转交给t′，即形成新消息（m′，t′）来完成原来的功能。

将上述过程描述成函数形式（设反映Law功能的函数为law），有：

send：（m，t）→send（s，m，t）

law：send（s，m，t）→{deliver（m′，t′），fail}

（对deliver（m′，t′），如（m=m′）∧(t′=t)，则为上述第一种情况，否则为上述第三种情况）deliver：deliver（m，t）→response

因此通过Law，将消息传送过程分为：消息发送—法则检查—递交处理三个阶段，Law对消息起到了"过滤器"的作用，因此，LGS可以看成是上述三个函数的合成（设反映LGS功能的函数为LGS）：

LGS：（m，t）→{response，fail}

LGS=deliver.law.send

Law是控制模块间消息发送的一组规则集，其中的第一条规则R0总是：

send（S，M，T）←canPass（S，M，T）& deliver（M，T）

谓词*canPass*表示函数law的过滤作用，因此R0反映了上述函数的合成关系。当规则退化为：send（S，M，T）←deliver（M，T）时表明：任何模块发送的任何消息都可以直接递交处理，因为此时谓词canPass恒为真，这是LGS的特例，实质上退化成一个不受任何法则制约的系统。

下面我们给出前述分层模块系统S1必须遵守的法则L1的形式表示L1′，设模块中说明该模块所在层次的属性变量是level，则可将L1′表示成：

R11′，R0

R12′：canPass（S，M，T）←S.level=T.level

R13′：canPass (S, M, T) ←S.level=T.level+1

L1′：L1的形式表示

二、基于法则的面向对象系统

1. OOP与LGS

OOP方法因其支持大型系统的开发等特点正日益受到人们的青睐，可以看到：OOP和LGS都是从不同的角度支持大型系统设计的，前者强调方法学概念，即将OOP的有关思想通过OOPL贯穿于开发过程中，是隐式控制形式；后者强调在开发和运行过程中系统必须遵守的法则，是显式控制形式。显然，如果将两者结合起来，将OOP的方法学思想显式地体现在法则中，将使OOP控制策略易于说明，并可随系统的演变而不断修改，这就是基于法则的面向对象系统（OOLGS）。OOLGS可以看成是LGS在大型系统开发方面支持OOP策略的典型实例，当然还可以支持其它策略。

OOLGS=System+OOPLaw，这反映了人们对大型复杂、易变系统的处理态度：

1.将系统看成一个实体集合，并在各实体间根据其属性和功能进行分类抽象。

2.实体间通过消息发送实现系统功能。

3.实体间的消息发送必须受到法则的制约。

4.制约实体间消息发送的法则本身应能随系统的演变而变化。

前两点体现了ERA模型及OOP思想，后两点体现了LGS的思想，这说明了人们对客观世界认识的深化：一个具有良好秩序的系统是由受到法则制约的一组个体组成的。

正如P.Wegner[4]认为的那样，一个面向对象系统应包含三个要素：对象、类如继承。相应地将系统分成按OOP语义逐步增强的三个层次：基于对象（object-based）系统，基于类（class-based）系统，面向对象系统。

我们将分别按这三个层次，给出支持相应语义强度的法则。为说明方便起见，将实体发送的消息分成三类：

1.原始（Primitive）消息：其功能由LGS系统本身定义，消息前缀以“#”号，共四条：

（1）（#set（A，V），t）表明将对象t中的属性A置成V值。

（2）（#get（A，V），t）表明引用对象t中的属性A，并赋给变量V。

（3）（#kill，t）表明撤消对象t。

（4）（#new（Image），t）表明根据Image创建一个新对象，Image中罗列了新对象经初始化的所有属性，包括对象标识符等。

2.直接（direct）消息：其功能由对象中的方法定义，是可以人为改变的，此类消息发出后，由LGS直接转交给目标对象（相当于law函数中的第1种情况），消息前缀以“@”号。

3.间接（indirect）消息：此类消息前缀以“^”号，它发出后，由LGS转换成直接消息，再转交给另一对象执行（相当于函数law中的第3种情况）。如果向某类发送消息，当此类中无法执行相应方法时，就需将消息转交给其父类执行，这类消息就是间接消息。

2. 基于对象的LGS

根据[4]中的定义，对象（object）是一个二元组（state，operation），state是一“属性-值”集合，operation是一方法集合，反映了对对象属性集合的操作，state记忆operation的操作结果，operation反映了对象的主动特性。支持对象特性的LGS称为基于对象的LGS，记为系统S2（设对象的属性之一是对象标识符id，用大写字符表示）。

相应地，支持S2的LawL2如下所示：

R21：R0

R22：canPass (S, #M, T) ←S=T

R23：canPass (S, @M, T)

L2：基于对象的LGS法则

其中，R22表明只有对象自身才能直接访问自己的属性，R23表明访问其它对象的属性可以通过向其发送直接消息@M，并由目标对象来执行。这显然支持了封装概念。

3. 基于类的LGS

根据[4]中的定义，类提供了一种模板结构（template），即同类对象的属性框架（frame）。附以相应初始属性值的框架称为映象（image），可以根据映象，通过new操作来创建新对象，同类对象的模板结构相同，即具有相同属性类型的操作，这些对象称为相应类对象的实例对象，它们的结构如下所示：

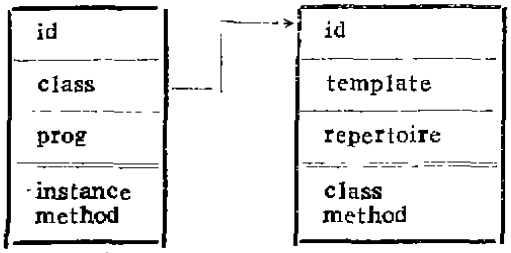

图1 实例对象和类对象

其中，class表明实例对象所属的类，prog表明实例方法的列表。repertoire表明类方法的列表，类方法在Objective-C中称为生成方法，template反映了相应实例对象属性的结构，类似于Objective-C生成对象中的type变量[6]。

基于类的LGS S3是建立在基于对象LGS S2基础上的，在L2中增加若干反映"类-实例"关系的法则R32——R34后，组成了支持基于类LGS的法则L3：

R31：R0

R32：canPass(S, #M, T) ←—isClass (S) &instanceOf(T,S) &(M=set(A, V) |M=get (A,V) & inTemplate(A,S))

R33：canPass(S, #new(Image),T) ←—isClass(T) &okStructure(Image,T)

R34：canPass(S, #Kill,T) ←—isClass(S) &instanceOf (T,S)

R35：R23

L3：基于类的LGS法则

R32——R34控制原始消息的发送，R35控制直接消息的发送，R32表明类可以访问其实例的属性，R33表明可以向类对象发送消息new产生相应类的实例，okStructure (Image, T) 表明Image符合类T中要求的template模式，R34表明类可以撤销其实例对象。在L3及以后的法则中，对功能简单且表意明显的谓词，如isClass等，我们都不做解释，并假定已由系统提供。

4. 面向对象的LGS(OOLGS)

如果基于类的LGS再进一步支持类继承，则形成OOLGS，即系统S4。相应地，支持OOLGS的Law为L4，这里继承的含义同Smalltalk[1]，因此，OOLGS实质上起到了"真正的"OOPL的作用。

R41—R45：同R31—R35，即L3。

R46：send(S, ^M,T) ←—transform(M, M1,T) &inheritsFrom(M, T, T1) &deliver(@M1, T1)

R47：transform(M, M1, T) ←—M=msg(args) &M1=msg(T, args)

R48：inheritsFrom (M,T, T1) ←instanceOf(T, C) &findClass(M, C, T1)

R49：findClass(M ,C,C) ←—inRepertoire(M,C)

R410：find Class (M, C,T1) ←—not inRepertoire (M, C) &parentOf (C1, C) &findClass(M, C1, T1)

L4：OOLGS的法则

其中，R42——R44控制原始消息的发送；R46——R410控制间接消息的发送；R41——R45支持系统中"实例-类"的抽象，R46——R410支持系统中"子类-类"的抽象，即支持类继承。在L3的基础上，加上控制间接消息发送（可能是实例对象将消息转送给相应类对象，也可能是子类对象将消息转送

给其父类对象)的法则R46——R410，组成了支持OOLGS的法则L4。

R46表明对间接消息的处理分成三个步骤：

1.谓词transform将消息M转换成M_1，转换方式由R47控制，即在原消息正文前缀上消息发送的原始目标对象，使消息的真正处理者得知这一信息。

2.谓词inheritsFrom将目标T转换成T_1，即寻找真正处理消息的对象T_1。T_1可能是实例对象的相应类对象，也可能是某类对象的父类对象，直至找到能真正处理消息的类对象T_1为止(R49)。R48—R410描述了沿着继承链上溯确定T_1的算法过程，即刻画了继承语义，R48—R410表示的搜索策略可按要求进行修改。

3.将消息M_1直接发送给T_1，即R46中的谓词deliver(@M_1, T_1)。

5. 多重继承

多重继承问题是OOP中的一个重要研究领域，上述L4中的R48—R410支持OOP的单重继承，我们只要对其加以修改，就可以使上述的OOLGS支持多重继承。从类型树的角度看，单重继承的类组成一棵有向树，具有多重继承关系的类组成一个有向无循环图(DAG)，如下图所示：(结点表示类，箭头表示继承关系。)

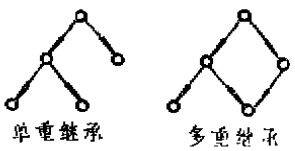

图2 单重继承和多重继承

因此，多重继承中对消息最终接收（处理）对象的搜索，由单重继承的有向树搜索变成对DAG的搜索。以下假定父类中的方法名不发生重名，我们给出OOLGS中支持多重继承部分的法则L5，以此来替换L4中的法则R48—R410。

R51: multipleInheritsFrom(←instanceOf(T,C) &findClass(M,C,T_1)&Q=enqueue(□, C)

R52: findClass(M, C, C) ←inRepertoire(M, C)

R53: findClass(M, C, T_1) ←not inRepertoire(M, C) &parentsOf(C_1, C) &Q=enqueue(tail(Q),C_1) & C_2=dequeue(Q) &findClass(M,C_2,T_1)

L5：支持OOLGS多重继承部分的法则

其中，Q=enqueue(x,y)表示将元素y加入队列x，形成新队列Q的入队操作，dequeue(Q)表示出队操作，tail(Q)表示除队列Q首元素外余下的队列部分，parentsOf(C_1, C)表示类C的父结点是一个类对象的集合C_1，R53反映了DAG的宽度优先搜索策略。

我们还可以用其它法则显式地替换R48—R410，来反映不同方式的继承机制，这正是LGS的最大特点，而在传统的OOPL中是难以改变语言固有的继承方法的。

三、讨　论

LGS的实质是将系统易变的控制部分和系统本身相分离，反映在大型系统开发上，就是将控制系统开发的策略(方法学)以及系统内模块访问的控制机制和程序语言本身分离，前者构成的法则可以显式地制约系统的功能结构，并且可以通过改变法则部分来随时响应系统发生的演变。OOLGS将OOP思想和LGS相结合，其中反映OOP概念的法则可以显式修改，以灵活支持OOP中的不同机制(如多重继承)。OOLGS是LGS在OOP方面的一个应用实例。

关于LGS的实现及应用方面，还有以下问题需进一步研究：

1.由于法则在LGS中起到了对消息的"过滤"作用，因此法则对系统的制约作用可以通过将消息截获(intercept)的方式，将它和法则中的规则进行匹配解释实现，这种实现方式的效率有待进一步提高。

2.原则上，支持LGS的程序设计环境可

以用任何语言实现，如Darwin程序设计环境[2]完全采用PROLOG实现，能否用OOPL来实现OOLGS，使有关OOP的基本功能依然由语言本身支持，法则部分只是在其上再追加OOP的其它机制(如多重继承，不完全继承等)，这样可以在保持LGS优点的基础上改善其环境效率。

3.LGS和规则系统(rule-based system)，约束系统(constraint-satisfaction system)既有联系又有区别，联系是它们都涉及到用规则(rule)描述问题，而区别在于后两者的目的是要取代(replace)整个系统，而LGS只是希望其中的法则能制约(regulate)系统的结构、功能及其演变，便于人们对大型系统的开发和管理。

4.LGS除描述OOP语义外，还可以控制原型系统(prototypical system)中的消息授权(delegation)处理[8]；或用一组控制模块间访问法则来构成MIL[2]；或利用LGS来描述资源的存取保护机制等，这些都是LGS中需要进一步考虑的课题。

参考文献

〔1〕 Goldberg,A. and Robson, D.,Smalltalk-80: The Language and its Implementation, Addison-Wesley, 1983.

〔2〕 Minsky, N. H. & Rozenstein, D., A Software Development Environment for Law-Governed Systems, In Proceedings of ACM/SIGSOFT Symposium on Software Development Environments, pp65-75, Nov.1988.

〔3〕 Naftaly H. Minsky & Rozenshtein,D., Controllable Delegation: An Exercise in Law-Governed Systems. In Proceedings of the OOPSLA's 89 Conference, pp 371-380, Oct. 1989.

〔4〕 Peter Wegner, Dimensions of Object-based Language Design, In Proceedings of the OOPSLA'87 Conference, pp 168-181, Oct. 1987.

〔5〕 陈火旺等，程序设计方法学基础，湖南科学技术出版社，1987.

〔6〕 Brad, J. Cox, Object-Oriented Computing: An Evolutional Approach Addsion-Wesley, 1986.

〔7〕 Jay Almarode, Rule-Based Delegation for Prototypes, In Proceedings of the OOPSLA'89 Conference, pp 363-370, Oct. 1989.

（接第51页）

$$+\sum_{i=2}^{n} m_i)/T \cdot n \qquad (4)$$

方程(4)的最大值是一个背包问题，即是在最小的T时间时$\sum_{i=1}^{n} m_i$为最大，可用相应的算法求解出(4)的最大值，即总线的最大可利用率。从中可知，要想提高总线的吞吐率，须使总线的使用有选择性。

当考虑的情形为$z_7=0$时，即后一节点机在前一节点机刚释放总线时便可获得总线的使用权，即有

$$1 > \sum_{p=1}^{n} A_p \cdot i_p$$

这时总线的利用率提高了$T/(T-z_7)$。

对于多总线，设有b根，则(3)式可化为

$$b > = \sum_{p=1}^{n} (A_p + z_7) \cdot i_p$$

随着b的增大($b=<n$)，i_p便接近于最大允许值$1/(S_p+A_p)$。

对于令牌总线的分析可循此法进行。

3. 结 论

本文利用时间Petri网作为工具讨论了局域网中的两种总线控制方式及相应的系统性能。从中可看到，Petri网不但可以作为一种论证工具，同时也是一种较优的评价工具。用时间Petri网作为性能评价工具，可很方便地和计算机工具结合起来，为设计和评价一个系统提供了一种方法，所以说时间Petri网是一个值得注意的研究领域。(参考文献略)