Gödel 语言对现代软件工程方法的支持 *

李松斌，赵致琢，李慧琪

(厦门大学计算机科学系，福建 厦门 361000)

摘 要：文章以一种新型的逻辑程序设计语言——Gödel 语言为例，对逻辑程序设计语言与现代软件工程主要思想和方法的结合情况进行了初步探讨。Gödel 语言通过引入模块系统、类型系统和延迟计算机制等不同于以往逻辑程序的新机制，能够支持现代软件设计的一些主流方法。模块系统提供了组织大规模程序的方法，并通过结合类型系统实现了对抽象数据类型程序设计的支持，从而能够支持面向对象程序设计；延迟计算则使利用该语言所编制的软件在可重用性方面获得提升。

关键词：Gödel 语言；软件开发方法；模块系统；类型系统；延迟计算

0 引言

现代软件工程的主要目标是让软件系统更容易开发、维护和理解，并让软件系统获得更高的可重用性、可靠性、可扩展性、安全性和灵活性，从根本上解决软件危机。基于上述思想，研究人员提出了多种现代软件开发方法，如面向对象软件设计、基于中间件的软件设计等。但是，对这些方法的运用多结合过程性语言展开，如 C、C++ 和 Java 等，与说明性逻辑程序设计语言相结合的例子则较为少见。实际上，逻辑程序提供了一种说明性的程序设计方法，说明性语言在与现代软件工程方法相结合方面应该说相比过程性语言更有潜力。这是因为，首先，逻辑程序建立在一阶谓词逻辑的基础之上，具有严格的数学背景，易于进行程序的变换、语义的廓清、正确性证明和验证，使程序具有较高程度的说明性和安全性。其次，逻辑程序设计用逻辑语言描述问题，利用归结推理技术来求得问题的答案，可以把逻辑程序非形式化地概括为：程序 = 逻辑说明 + 执行控制。理想的逻辑程序应该只说明程序的逻辑部分，而把控制部分即推理求解部分交给机器自己完成。这样，客观上可以降低程序员的负担，提高软件开发效率。最后，由于逻辑程序设计中严谨的数学表达规范和数学理论基础，程序员的工作基本上不涉及程序的控制，使用户对程序的理解变得简单，从而有利于程序的重用、维护和扩展。

Gödel 语言[1]是一种新型通用的逻辑程序设计语言，它充分吸收了逻辑程序设计研究领域的最新成果并借鉴了过程性语言作为软件开发工具的多种新思想，改变了 Prolog 用在大规模软件设计方面的缺陷。首先，它融入了面向对象程序设计思想，通过引入数据类型、支持抽象数据类型设计获得对信息掩蔽、操作封装和函数多态多类等机制的支持；其次，它引入了模块化程序设计和元程序设计机制，支持软件重用、维护和升级，能够方便用户进行大规模程序开发；第三，它引入了延迟计算和剪枝操作，能够较好地实现程序的高效执行。本文将以 Gödel 语言为实例，探讨新型逻辑程序设计语言对现代软件工程方法的支持。

1 Gödel 语言对现代软件工程方法的支持

随着应用的普及和应用领域复杂程度的不断加深，软件需求日趋复杂，为适应这些变化，各类现代软件开发方法应运而生。这些新的软件开发技术的主要作用是满足软件开发对于生产效率、可靠性、易维护性、易管理等方面的更高、更快、更强的迫切需求。针对这一目标现代软件开发的主流思想一般是采用面向对象的开发技术，结合模块化、抽象和重用的概念，进行集体协作级别的大规模的软件开发。这些变化反映在编程语言上——这些新思想的实现需要编程语言的支撑，使编程语言的抽象级别不断提高。很难想象用机器或汇编语言来实现面向对象编程。过程性的程序设计语言如 C++、Java 等的出现，使这些新的设计技术的实现成为可能。对于逻辑程序来说，作为一种说明性的程序设计方法，它的抽象级别更高，理想的状态是我们只须告诉机器做什么，而不需要告诉机器怎么去做，但是这很难做到。作为一种更为现实的策略，通过借鉴在过程性语言中已经实现的软件开发技术，来增强逻辑程序的设计能力，具有更为重要的意义。Gödel 语言的设计者们在这方面作了尝试。Gödel 语言通过引入新的语言成分和机制实现了逻辑程序设计对现代软件工程方法的支撑。这种支撑主要表现在将程序设计的一些新技术融入到逻辑程序设计语言中，使其成为能够支撑现代软件工程方法的新型逻辑程序设计语言。下面，结合 Gödel 语言中的新机制（如模块系统、类型系统和延迟计算），通过一些 Gödel 语言程序作为实例，对本节论题进行阐述。为了使读者对 Gödel 语言有一个更为感性的认识，我们先给出一个具体的 Gödel 语言程序：

例 1：

```
EXPORT      M5.                 % 模块 M5 的输出部分
BASE        Day, Person.        % 模块 M5 的基本类型声明
CONSTRUCTOR List/1.             % 模块 M5 的构造子声明
CONSTANT    Nil : List('a);     % 该常量是各种类型的空表
```

* 基金项目：厦门大学创新研究基金项目(Y07012)；福建省自然科学基金项目(A0310007)

```
    Monday, Tuesday, Wednesday, Thursday, Friday, Saturday,
      Sunday : Day;
    % 声明 Person 型常量
    Fred, Barney, Wilma, Betty : Person.
  FUNCTION        Cons: 'a*List('a)→List('a). %类型转换函数
    % 谓词声明
  PREDICATE       Append: List('a)*List('a)*List('a);
    Append3: List('a)*List('a)*List('a)*List('a).
  LOCAL           M5.     % 模块 M5 的本地部分
    % 谓词定义
    Append(Nil, x, x).
    Append(Cons(u, x), y, Cons(u, z)) ← Append(x, y, z).
    Append3(x, y, z, u) ← Append(x, y, w) & Append(w, z,u).
  MODULE          M6.     % 模块 M6 声明
  IMPORT          M5.     % 导入模块声明
  PREDICATE       Member2 : a * a * List(a).   % 谓词声明
    Member2(x, y, z) ← Append3(_, [x|_], [y|_], z). % 谓词定义
```

例 1 作为一个完整的程序由两个程序模块组成，其中模块 M6 是主模块，它导入了模块 M5。该程序首先在模块 M5 中定义了表连结操作谓词，其中 Append 谓词用于将其第一个参数表放在第二个参数表的前面构成一个更大的表，存放在第三个参数表中，Append3 在调用 Append 的基础上定义了三个表的连结操作；在模块 M6 定义的谓词 Member2 中调用了 M5 中 Append3，Member2 为真当且仅当 x 和 y 是列表 z 的元素且 x 在 y 的前面，如当询问：Menber2 (Friday, Monday, [Thursday, Friday, Monday])时该程序将返回真。关于该程序的详细说明将在模块系统中介绍。

1.1 Gödel 语言的模块系统

Gödel 语言的模块系统不仅为语言提供了组织规模程序的方法，而且通过引入类型系统实现了支持抽象数据类型程序设计，从而为支持面向对象程序设计奠定了基础。

1.1.1 对大规模程序设计的支持

在软件工程方法中模块化思想占有举足轻重的地位，模块化提供了组织大规模程序的一种方法。在实现细节上模块化可以避免程序的不同部分之间的名字冲突，使其不会互相影响，而且也提供了隐藏执行细节的方法。在设计方法上，模块化有利于降低系统的实现难度（分而治之），并提供了让一个团体有效地进行协同开发的工作方式。Gödel 语言的模块系统正是基于这样一种思想：一个程序是多个模块的集合，其中至少有一个是主模块，其他模块由主模块导入或间接导入。一个模块通常由两部分组成，即输出部分和本地部分。输出部分为程序的说明部分，一般包括模块声明、常量声明、谓词声明、延迟声明等。模块的本地部分主要用于给出输出部分定义谓词的实现细节，当然它也可以包含输出部分的所有成分。

如例 1 程序中，模块 M5 包含一个输出部分和本地部分，输出部分通过说明产生全部可用的符号。特别地，程序所声明的基类 Day 和 Person、常量 Monday 和 Fred 等以及谓词 Append 和 Append3 在其它模块中都是可用的，只要在这些模块中导入 M5。模块 M5 的本地部分定义了谓词的实现细节。与 M5 模块相比，M6 只包含了本地部分，因此它的首模块说明使用了关键字 MODULE 而不是 LOCAL，M6 导入了所有的被 M5 导出的符号，M6 包含了谓词 Member2 的定义。从上面的解说中我们不难看出一个 Gödel 语言程序由多个相对独立的模块构成，模块集$\{Mi\}_{i=0}^{n}(n\geq 0)$组成一个完整的程序，其中 M0 是一个最主要的模块，称为主模块，而子集$\{Mi\}_{i=1}^{n}$中的模块是这样的模块，它要么出现在 M0 模块的导入说明中，要么出现在另一模块（已经在以 M0 为头的导入说明"链"上）的导入说明中，对于上面的例子，{M6, M5}构成了该程序，其中 M6 是该程序的主模块。通过上述模块系统，Gödel 语言的程序员就可以像堆积木一样来构建他们的程序了。

1.1.2 对面向对象程序设计方法的支持

在模块系统的支持下，可以将任务进行抽象并封装在一个模块中，而且还可以把这个任务的实现细节隐藏在模块的本地部分（本地部分对其他模块是不可见的），然后经模块的输出部分为其他模块提供调用本模块符号的接口。借助上述机制，Gödel 语言实现了对抽象数据类型程序设计的支持。一个抽象数据类型由一个类型和该类型上的操作集合组成，类型表示方法的细节和操作执行细节，它对用户是不可见的。一个抽象数据类型可以定义在一个模块中，其中，类型定义可以放在模块的输出部分，而其实现细节可以存放在模块的本地部分，向用户提供定义文件而隐藏对应的实现文件。由此可见，通过支持抽象数据类型程序设计，Gödel 语言的设计实际上融入了面向对象的程序设计思想。

文献[2]讨论了逻辑程序设计语言支持对象式逻辑程序的一些基本要求：要求语言提供机制来支持对象、类、封装、隐藏、代码重用、继承以及对象之间进行消息传递等。下面，我们围绕这些特点来讨论在 Gödel 语言中如何支持实现面向对象程序设计。Gödel 语言程序的设计者可以把一个具体问题抽象为多个类的集合。一个类用一个模块来描述，模块的输出部分相当于类的头文件而本地部分则视为实现文件，这样就实现了信息的隐藏和操作封装。如模块 M5 中，输出部分定义了 Append 和 Append3 谓词，并将这两个谓词的实现细节隐藏在其本地部分，这样，模块M6 只要得到模块 M5 的输出部分（头文件）即可进行调用，无须知道其实现细节。进一步，模块 M5 可以自由地修改其实现部分，只要保持头文件不变即可。针对对象间的消息传递，Gödel 语言主要通过接口谓词来实现。接口谓词就是由模块的输出部分定义的谓词。Gödel 语言作为逻辑程序设计语言，程序的执行过程是一个反驳消解的过程，接口谓词作为本模块对其他模块提供的服务，实际上类似于一个接口，起到激活该谓词所在模块（类）的作用。程序在执行过程中一旦需要其他模块的谓词（处于该模块导入链中的某模块）来参于匹配过程，则推理支持系统立即生成该谓词所在模块在内存中的一个实例，从而使该谓词的推理得以进行。重要的是在这个调用接口谓词的过程中，接口谓词获得了由其他模块提供的执行参数，从而实现了消息传递。关于 Gödel 语言对多态程序设计的

支持将在类型系统部分进行说明。

1.2 Gödel 语言的多态多类系统

长期以来，逻辑程序设计研究者一直在不断地改进逻辑程序设计语言，在增强逻辑程序设计语言的可表达性方面提出了引入多态多类逻辑的构想[3,4]，并体现在 Gödel 语言的设计之中。Prolog 语言的理论基础是一阶逻辑的 Horn 子集，语言设计中没有考虑引入数据表示的类型系统，因而其表达能力和作用十分有限。Gödel 语言的设计吸收了 Prolog 语言中正反两方面的经验和教训，将语言的理论基础建立在多态多类的一阶逻辑基础之上，使语言具备了更强的描述能力，更易于程序开发和编译实现中的错误检查。

作为具有多种类型的语言，Gödel 语言程序的类型集合由语言声明中的基类（BASE）声明和构造子（CONSTRUCTOR）声明决定。其中，构造子是一个类型生成函数，用于构造新的类型，它以类型变量作为其参数，类型变量的值域为基本类型和构造子可能生成的所有类型。模块 M5 中基类声明（BASE）给出了两个用户自定义类型 Day 和 Person；构造子声明（CONSTRUCTOR）给出了构造子函数 List，构造子函数后的 /1 表示该构造子是一元构造子，即只有一个类型变量作为参数。程序中的 'a 即为类型变量，模块 M5 的类型集合 T_{M5} 为{Day, Person, List(Day), List(Person), List(List(Day)), …}。在多类的基础上，Gödel 语言可以支持多态类型，这极大地提高了程序设计的灵活性。类似于面向对象语言中的多态函数，Gödel 语言中的谓词或函数可以处理多种不同类型的输入，可以方便地把已定义的多态谓词、函数和数据结构重用于任何需要的类型实例。Gödel 语言的多态称为参数型多态，通过引入类型变量，使谓词或函数中的参数类型为一个类型集合，从而使得该谓词或函数可以处理不同的类型。如在模块 M5 中，通过构造子 List 和类型变量 'a，使程序不仅能够处理 Day 型链表的联结操作，如 Append([Monday, Friday], [Thursday], x)，此时 'a 类型为 Day；也能处理 Person 型链表的连结操作，如 Append([Fred], [Barney, Wilm], x)，此时 'a 类型为 Person，只要 'a∈T_{M5} 该程序即可处理。而在 Prolog 语言中，这将需要多个程序才能实现。通过上述机制，Gödel 语言实现了对多态多类程序设计的支持。

1.3 Gödel 语言的延迟计算机制

现代软件工程研究的一个重要目标是有效地实现软件的重用、维护和升级。软件重用在逻辑程序中的本质是要求实现谓词的重用[5]，并在此基础上实现模块级别的重用，而谓词重用的一个困难问题是不知道谓词的调用方式，从而造成程序陷入死循环。例如，如果我们对模块 M5 中的谓词 Append 进行了如下调用：

Append(x,[Fred, Barney, Wilma], z)

那么必然导致程序陷入死循环。这是因为此时表 x 中的元素必须按顺序放在表[Fred,Barney,Wilma]的头部，为了能够得到这些元素，推理程序会去比较表 z 和表[Fred, Barney,Wilma]中的元素，并取出表 z 中属于表 x 的那部分作为表 x 的解，而不幸的是 z 此时也是变量，由于 z 是变量推理程序将去比较表 x 和表[Fred,Barney,Wilma]，从而陷入循环。逻辑程序设计很难走向大规模软件设计领域，这也是其中的一个原因。Gödel 语言通过引入延迟声明和模块系统，基本解决了这一问题。Gödel 语言引入了延迟计算，通过程序中的延迟计算声明（DELAY），明确地给出谓词（子目标）被扩展执行的条件。

引入延迟声明后，程序员可以在模块的输出部分，通过延迟声明定义谓词的调用方式（没有进行延迟声明的谓词，默认为可以任何方式调用），然后向其他程序员披露模块的输出部分。这样，想要重用这一模块的程序员，只要根据所输出部分声明的条件进行调用就可以避免陷入死循环。如在模块 M5 中，为 Append 谓词加入延迟声明如下：

DELAY Append(x,_,z) UNTIL NONVAR(x)∨NONVAR(Z)

该声明指出 Append 谓词被执行的条件是第一个或第三个列表参数不为变量。所以我们上面的调用显然违法，在此种情况下 Append 谓词可保证终止执行，这是因为此时程序至少能够给出所有可能的解的组合。如我们考虑一种极端的情况：

Append(x, y, [Fred,Barney])

此时程序将给出解：

x=[],y=[Fred, Barney];

x=[Fred], y=[Barney];

x=[Fred, Barney],y= []。

因此，延迟声明为谓词的重用提供了有效的方法，并结合语言的模块系统提供了利用逻辑程序设计语言进行团队协作开发应用系统的一种新方式，从而使语言具有基于中间件的软件设计能力。

2 结束语

本文在逻辑程序设计技术与软件工程开发方法的结合方面作了一些初步的探讨，并以一种新型的逻辑程序设计语言 Gödel 为例，对该种语言对现代软件主要设计方法的支撑情况做了一些考察和分析。结果表明，该语言巧妙地融入了多种现代软件开发思想，如面向对象、模块化和软件重用等，为逻辑程序设计语言应用于大型软件开发开辟了道路。逻辑程序在软件开发领域具有天然的优势，相信随着对该语言研究的深入，未来逻辑程序设计语言能够更多地应用于大型软件开发之中。

参考文献：

[1] P.M.Hill, J. W. Lloyd. The Gödel Programming Language [M]. Cambridge USA,MIT Press,1994.

[2] 徐殿祥，郑国梁. 对象式逻辑程序设计 [J]. 计算机研究与发展，1996.33(1):17~23

[3] P.M.hill, R.W.Topor. Semantics for typed logic programs [A]. Types in Logic Programming [C]. Cambridge USA, MIT Press, 1992:1~62

[4] A.Mycroft,R.A.O'Keefe.A polymorphic type system for Prolog. Artificial Intelligence[J],1984.23(1):295~307

[5] S. Genaim, M. Codish. Inferring Termination Conditions for Logic Programs using Backwards Analysis [A].Proceedings of the Eight International Conference on Logic for Programming, Artificial Intelligence and Reasoning [C].Springer- Verlag,2001: 681~690