Gödel 语言类型系统

王炳波， 赵致琢， 晏 松
(厦门大学 计算机科学系，福建 厦门 361005)

摘 要：Gödel 语言是一种说明性逻辑程序设计语言。该语言基于一阶逻辑，引入了一个多态多类的类型系统和多种新的语言成分，支持抽象数据类型和模块化程序设计等技术，语言本身也具有很强的说明性语义。详细介绍了 Gödel 语言的类型系统及其构造，对在逻辑程序设计语言中引入类型系统的作用进行了一些深入的分析和讨论。

关键词：Gödel; Prolog; 类型系统; 多态多类; 类型推理
中图法分类号：TP311 **文献标识码：**A **文章编号：**1000-7024 (2005) 12-3432-04

Type system in programming language Gödel

WANG Bing-bo, ZHAO Zhi-zhuo, YAN Song
(Department of Computer Science, Xiamen University, Xiamen 361005, China)

Abstract：Gödel is a declarative logic programming language. It is based on first order logic with a polymorphic many-sorted type system. Having a strong declarative semantic Gödel can be used to realize abstract data types and programming in modules. The type system was introduced and the function of the type system in a logic programming language was discussed.

Key words：gödel; prolog; type system; polymorphic many-sorted; type deduction

1 引 言

Gödel 语言是继逻辑程序设计语言 Prolog 之后提出的一个既实用又具说明性的逻辑程序设计语言。众所周知，Prolog 相对于其它许多高级语言在描述上是更高级的和更有效率的，比较直观地反映人的思维结构和概念描述方式。然而，由于 Prolog 语言不包含类型说明机制，使得它的表达能力很有限。而且，Prolog 程序不具有清晰的说明性语义，甚至包含了复杂的过程语义，这意味着分析、转换、优化、验证、调试 Prolog 程序非常困难[1]，程序员的负担很重。

Gödel 语言的出现可以改变这种状况。通过比较 Prolog 语言和 Gödel 语言，我们从中可以看到 Gödel 语言一些好的设计思想和新的语言成分，类型系统和新的语言成分的引入改善了 Prolog 语言的非说明性特征。Gödel 语言的主要机制包括一个类型系统、一个模块系统、一些控制机制、元程序设计和输入/输出部分。Gödel 语言是一种多态多类的语言，其类型系统是一个强类型系统，这使得语言具有更强的表达能力和实用性。Gödel 语言程序设计采用模块化结构，拥有更灵活的计算规则，通过控制剪枝等操作，可以辅助提高推理树的搜索效率；采用抽象数据类型来处理元程序设计，为元程序设计提供了一个良好的环境；Prolog 语言程序的输入/输出接口对程序的说明性语义有很大影响，为了改善这种情况，Gödel 语言专门为程序员提供了处理 I/O 操作的系统模块。这些机制都使语言的说明性语义明显提高。

Gödel 语言中引入的类型系统是比较复杂的，我们将在本文中主要介绍和讨论这一类型系统。

2 类型系统与程序设计语言

在程序设计语言中引入类型系统，对高级程序设计语言发展的影响是深远的。程序设计语言理论的研究表明，借助类型检查可以排除程序中可能存在的一大类语义问题。类型之所以能起到这个作用，是因为它包含了一定的语义信息[2]。类型系统通常给程序加上一个静态的类型结构，给所有的常数、变量以及函数符附加上类型信息，显式或隐含地指出对象的取值范围和允许施加在其上的运算操作。这样，编译时可通过检测类型错误，把大量程序错误限制在早期，避免了运行时的大量测试工作，从而提高程序的正确性和执行效率，这也是程序设计语言使用类型的最根本的动机[3]。可见，构造程序设计语言的类型系统，一方面为描述语言的语义提供了部分数学模型，是语言的一种严格规范；另一方面，对提高该语言程序的编译质量，保证程序的正确性具有重要意义。

类型系统规定程序中的变量在使用前必须说明它的类型，这规范了变量的取值范围和所能对它进行的操作。高阶类型系统[4]支持在不同模块中定义和实现数据类型，支持抽象数据类型，并能根据所提供的不同类型参数实现多态，有力地支持了模块化程序设计、面向对象程序设计和软件重用。当

收稿日期：2004-10-09。
作者简介：王炳波 (1979-)，男，陕西西安人，硕士，研究方向为逻辑程序设计； 赵致琢 (1957-)，男，河南焦作人，博士，教授，研究方向为逻辑程序设计和分布式基础算法。

程序中有足够的类型信息时，类型系统只检测变量和函数的使用是否与所给的类型一致，即进行类型的检测。在有的程序设计语言中，程序中不包含完整的类型信息，编译器将由类型系统根据程序中变量的使用情况推导出所缺省的类型信息，即进行类型推理[7]和重构。这就需要引入一个类型指派系统在类型系统实现时进行类型推理和类型检测，并规定如何重建一个项的类型。不难理解，在程序设计语言中使用高阶类型系统是实现很多程序设计技术的基础，例如，面向对象的程序设计、模块化程序设计、函数式程序设计等。

下面我们来讨论 Gödel 语言中使用类型的方法以及 Gödel 语言所引入的类型系统。

3 Gödel 语言的类型系统

Gödel 语言的类型系统是一个高阶的多态多类的类型系统。在这个类型系统的基础上，语言支持在不同的模块中说明和实现多种数据类型，实现了对抽象数据类型、模块化程序设计、程序重用等程序设计技术的支持。我们还将看到 Gödel 语言类型系统在实现时引入的用来进行类型推理和类型检测的类型指派系统。

3.1 Gödel 语言程序中类型的说明

首先，我们来看看 Gödel 语言中进行类型说明的语法片段。

```
类型→类型变量| 基类|Constructor(n)<'('> TypeSeq( n)<')'>
类型变量→<小写名字>     %G? del 语言中可以说明基类、类型构造函数和类型变量
基类→用户命名          %基类的说明
Constructor(n) →用户命名
ConstructorDecl→<'CONSTRUCTOR'> ConstrDecl{<,> ConstrDecl}   %构造函数的说明
PredicateDecl→<'PREDICATE'> PredDecl{<;> PredDecl}   %谓词的说明(含有类型信息)
PredDecl→UserNameSeq[<':'> PredicateSpec(n)]<':'> 类型{<'*'> 类型}(n-1)
```

图 1　Gödel 语言中类型的语法片段

如图 1 中所示，关键字 BASE 开头的语言说明给出了多态多类语言的基本类型，称为基类。类型变量类似于变量，在语言中不做说明，到运行时进行动态的类型绑定。Gödel 语言中使用结构类型，类型变量是可以作为参数出现在结构类型中的。CONSTRUCTOR 说明了用户命名的含有 n 元类型参数[5]（称为选取子）的结构类型（称为构造子），可以为选取子指派语言说明产生的所有的类型。可以视基类为‘ 常量 ’，选取子为‘ 变量 ’，构造子为‘ 函数 ’来生成语言的所有类型的集合。Gödel 语言中以关键字 PREDICATE 说明的谓词都是基于类型的，对谓词的每一个参数都需要进行类型说明，具体的细节将在下一节中看到。

3.2 多态、多类的类型系统

现在我们给出程序里用来定义多态多类语言的语言说明细节。

我们的主要目的是解释 Gödel 类型系统。这里用只含一个模块的简单程序来方便地说明。一个模块包含模块说明、语言说明、控制说明和语句。采用关键字 LOCAL 进行模块说明，给出模块的名称。语言说明定义了一个多态多类语言，控制说明包含计算规则。语句是语言定义中的命题和谓词的公式。Gödel 类型系统，是基于带参数多态的多类逻辑。我们先讨论类型系统的多类方面。考虑图 2 中的模块 Lists。

```
MODULE          M                                   % 模块说明
BASE            Day, Person.                        % 基类说明
CONSTRUCTOR         List/1.                         % 类型构造函数
CONSTANT        Nil: List(a);                       % 常量说明
                Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday: Day.
                Fred, Bill, Mary: Person.
FUNCTION        Cons: a * List(a) -> List(a).       % 函数说明
PREDICATE       Append: List(a) * List(a) * List(a).    % 谓词说明
                Append3: List(a) * List(a) * List(a)* List(a).
Append( Nil, x, x).                                 % 语句
Append3(Cons(u, x), y, Cons(u, z))<- Append(x, y, z).
Append3(x, y, z, u) <-Append(x, y, v)&
                Append(v, z, u).
```

图 2　多态多类的类型系统的例子

语言说明以关键字 BASE、CONSTRUCTOR、CONSTANT、FUNCTION、PROPOSITION 或 PREDICATE 开头，分别说明了：基类、构造子、常量、函数、命题和谓词。在模块 Lists 里，BASE 说明 Day、Person 是基类。CONSTANT 说明 Nil 是 List(a) 类型的一个常量，Monday、Tuesday 是 Day 类型的常量，Fed、Bill 是 Person 类型的常量。FUNCTION 说明了 Cons 是一个二元函数，它映射一个二元组（第 1 个参数是 a 类型，第 2 参数是 List(a) 类型）到一个 List(a) 类型的元素。PREDICATE 说明了 Append 是一个三元谓词，每个参数都是 List(a) 类型，谓词 Append 的功能是向列表中追加元素。子句和目标子句用语言说明中定义的语言写出。注意变量以小写字母开头，常量以大写字母开头。语言说明的集合和模块 Lists 的语句一起构成了模块 Lists。

模块 Lists 里说明了一元构造子 List。注意，构造子本身不是一个类型，但程序中出现的所有基类和构造子都必须说明。没有构造子，则所有类型的集合只是所有基类的集合。如果有一个构造子被说明了，则类型的集合将把基类视为“常量”，把构造子视为“函数”来构造获得函数类型，比如我们可以构造 List(Day)、List(Person)、List(List(Day)) 等这样一个可数无限的类型的集合。Gödel 语言中变量的类型不需说明，由系统指派。这个类型指派必须满足语句是一个多类公式的要求。所以，这样的一个指定必须是惟一的。类似于数理逻辑中变量的使用，同一条语句里的不同变量指定不同的类型，而不同语句里同一变量的类型却不一定相同。例如，在模块 Lists 中，第 1 条语句里的变量 x 指定为类型 List(a)，第 2 条语句里的变量 u 指定为类型 a，变量 x，y，z 指定为类型 List(a)。

下面我们介绍类型系统的另一个方面，即参数多态性。通常一个程序员希望使用一个带有类型参数的谓词。例如，谓词 Append 通常要求其能对任何类型列表进行追加。为此，我们给类型系统添加参数多态性。

模块 Lists 中，a 是一个类型变量。对一个多态多类语言，我们扩展类型的概念。一个类型是一个项，它是这样构造的：将基类视为“常量”，类型参数（选取子）视为“变量”，构造子视为“函数”。对模块 Lists 来说，所有类型的集合将是 Day，Person，List(Day)，List(Person)，List(List(Day))，List(List(Person))，a，List

(a)，List(List(a))，等等。其中，把不含参数的类型称为是单态的，模块 Lists 中单态的类型是 Day, Person, List (Day), List (Person)，List(List(Day))，等等。

如果一个符号的类型说明中包含了一个参数，则该符号是多态的。一个多态符号可看成是单态类型的符号的集合。例如，模块 Lists 里对多态常量 Nil 的说明，其说明了类型为 List (a)的常量是由单态类型的常量组成的一个可数无限集合，a 覆盖所有基本类型。可以将这个集合里的不同常量设想成由 a 值标注的常量。这样 Nil 说明了单态类型 List(Day)的常量 NilDay，类型为 List(Person)的单态类型常量 Nilperson，等等。

我们同时也要把解释、类型指派等在多类逻辑中的一般概念扩展到多态多类逻辑中去。对于一个多态多类语言，解释中的每一基本类型对应一个域，这些域两两不相交。例如，在模块 Lists 中，具有多态类型 List(a)的常量 Nil 指派到一个可数无限的多态多类的类型域的元素的集合。用一个基类代替 List(a)的类型参数 a，生成的元素再代替 List(a)的类型参数 a，这样就会递归地生成这个类型常量所对应的多态的类型元素的集合，这也就相当于多态多类语言中对 Nil 所表示的一个多态类型常量的指派。同理，我们也可类似地定义函数和谓词的指派。

3.3 抽象数据类型

通常，一个含有类型说明的语言具有定义新的数据类型并在该类型上定义操作的能力[6]。一个抽象数据类型包括一个类型的集合和该类型上的操作的集合。类型表示方法的细节和操作的细节程序员是不可见的，这就很好地实现了数据掩蔽与操作封装，能提高语言的说明性表达能力，便于程序员在概念级进行程序设计。

在 Gödel 语言中，程序是模块的集合，一个模块由这个模块的本地部分和输出部分组成。例如，图 3 中为系统模块 Lists 的输出部分程序段。输出部分为使用该模块的其它模块提供了外部的接口，说明了其它模块可访问的数据类型和操作。

```
EXPORT    Lists.                                  %模块说明
IMPORT    Integers.                               %导入的模块说明
CONSTRUCTOR    List/1.                            %类型构造函数说明
CONSTANT    Nil : List(a).                        %函数说明
FUNCTION    Cons : a * List(a) -> List(a).        %谓词说明
PREDICATE    Member : a * List(a);
             Append : List(a)*List(a)*List(a);
             Permutation : List(a)*List(a);
             Delete : a* List(a)*List(a);                    …… ……
```

图 3　抽象数据类型的例子

Lists 模块的输出部分用 EXPORT 关键字来说明，相应地本地部分是用关键字 LOCAL 来说明(如图 2 所示)，模块名与输出部分是一致的。IMPORT 说明了在 Lists 模块中引入了另一个系统模块 Integers，它规定了我们在 Lists 中的谓词 Sort 的说明里可直接使用模块 Integers 的输出部分所说明的类型 Integer，而对于 Integer 类型的细节我们是不可见的。在任何使用 IMPORT 来引入 Lists 模块的程序中都可以使用该模块的输出部分已说明了的类型、常量和谓词等，但它们的实现细节同样隐藏在了模块 Lists 的本地部分。可见这样的程序段提供了对 Lists 抽象数据类型中数据的掩蔽和操作的封装，也为很好地支持面向对象的程序设计打好了基础。

3.4 类型系统的实现说明

Gödel 语言引入类型系统，大大提高了语言的表达能力。通过对 Gödel 语言类型系统的分析可知，在语言编译系统的实现中，需要对 Gödel 语言写的源程序进行静态类型检测(在语义分析过程中进行，如图 4 所示)。从而排除部分的语义错误，并把抽象语法结构映射到它的类型语义，用以在程序的执行过程中进行类型的推理和指派。程序中的语法信息为语言的翻译提供了非常重要的信息，决定了类型的绑定时间以及执行类型错误的检查。

Gödel 语言中变量没有语言说明，也没有默认的类型说明，变量是无固定类型的，这一点与 prolog 相同[6]。而且，Gödel 语言引入了类型变量，对参数化类型进行了抽象，将类型作为一个对象来处理，它们的值是在程序运行过程中匹配、推导出来的，需要使用一个类型推导和指派系统以便进行类型的动态检查来支持，如图 4 所示的动态类型系统。编译程序将类型信息作为类型标识附加在产生的代码上，在推理机运行过程中根据类型标识进行类型指派和类型推理。类型指派系统[7]由类型的推理规则组成，是类型系统实现时进行类型检测和类型推理的算法，它规定了如何重建每个项的类型。Gödel 语言中动态类型系统位于推理机内部，在推理机运行过程中进行类型检查、指派和类型推理。推理机运行的过程实际上是对项进行匹配搜索的过程，在项进行匹配的同时，根据一致匹配的规则对类型也进行相应的匹配绑定操作，从而实现带类型的逻辑推理。显然，在这一过程中，类型的指派和绑定是动态可变的。

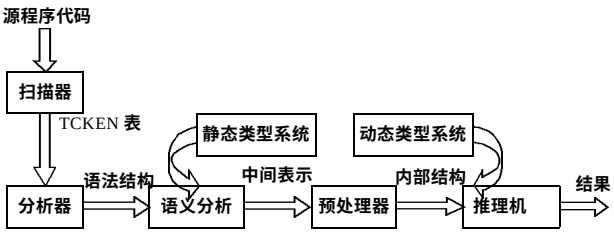

图 4　类型系统在编译器中的实现说明

4 Gödel 语言类型系统的作用讨论

前面我们已经了解了 Gödel 语言类型系统的实现构想，下面我们将对 Gödel 语言的类型系统以及引入类型系统所带来的语言在程序设计方面的优势和程序执行的效率等问题做一些分析和讨论。

4.1 引入类型系统增强了知识表示

在逻辑程序设计语言中引入类型的目的在于提高语言的表达能力，可以更方便地表示各种知识。预期解释在大多数逻辑程序设计应用中是被类型化的，所以，使用一个类型语言是在应用中获得相关知识的最直接的方法。Gödel 语言中引入类型系统之后能更确切地表达现实世界中概念和概念之间的关系。一个数据对象的类型决定了它的属性、取值以及这

些数值的逻辑组织结构和其上允许进行的各种操作。语言中采用类型使得知识的表示类型化；抽象数据类型的实现可使知识的表示结构化；模块系统则将知识的表示模块化进而可实现网络化。这些都说明 Gödel 语言能方便地表示出知识与知识之间的结构关系。同时，知识的表示是进行逻辑推理的基础，Gödel 语言中采用带类型的数据结构进行知识的表示，为逻辑推理提供了很好的结构数据模型。这样的带有类型的数据模型可以更直观、更精确地表现事物的内部结构以及事物的外部行为，是推理机执行的基础。很明显，Gödel 语言对事物的可表达性和描述的准确性都将比 prolog 有明显的提高。

4.2 引入灵活的多态多类类型系统

Gödel 语言允许多态形式，称为参数多态。类型变量涉及了所有类型，这样就允许用户使用各种参数类型的谓词。例如，常用的谓词 Append，在多态版本中，其列表中的元素虽然同样固定一致但却可以是任意类型的。又如图 3 中所示，系统模块 Lists 为用户提供了一个可以生成元素为任意类型的链表的模板，体现了 Gödel 语言对模板技术的支持。该模块是可以被任何其它用户模块方便引入的，这样就可以实现在 Gödel 程序中进行程序代码的复用。

Gödel 语言的类型系统是一个强类型系统，程序里的每一个项和它的类型必须进行语言说明。然而，变量的类型是由它们的上下文推测出来的。尽管如此，我们认为程序中有足够的信息可被编译程序和推理机用于判定类型并保证程序的安全执行，而不需要显式地给出变量的类型说明。结构类型只定义了构造子，对于所对应的选取子即类型参数的值的绑定是通过匹配过程完成的，这增强了类型的安全性。例如，图 2 中模块 M 对多态常量 Nil 说明了类型为 List (a) 的单态类型常量的一个可数无限集合，a 可取遍所有基本类型。a 的具体取值，依赖于程序的运行过程中对 a 进行的动态的匹配操作，要到运行过程中才进行类型的指派和绑定，而且，这种对类型变量的绑定还是可以动态改变的，这就很好地对类型进而对程序进行了保护，提高了程序的安全性。

注意到说明了构造子和基类，则类型集合是可数无限多的，这意味着语言说明规定的类型比程序员实际想要的多。例如，在模块 M 里，程序员真正想要的类型是 Day 和 List(Day)，这只是实际说明的类型集的一个很小的子集。我们可以采用一个更灵活的语言说明，允许程序员只说明类型 Day 和 List (Day)。但这种语言说明肯定使程序更加复杂，需要将类型变量的指派限定到原来的可数无限集的一个有限的子集上，但是，由于类型的指派要到程序执行过程中才能完成，这个有限的子集就很难在类型说明时就静态地确定，所以这种更灵活的语言说明将难以简单实现。而且这样做也并不值得，因为在预期解释里含有多余类型并不会带来问题，只不过增大了预期解释而已。

4.3 类型引入和 Gödel 语言搜索的执行效率

我们知道动态类型检测将使得程序设计更灵活[6]而不需要变量说明，但是，在程序执行中要保存类型信息，就需要额外的存储空间，且动态检查是靠软件来实现的，因此会降低程序的执行效率。在 Gödel 语言中，这不会影响程序的执行效率，相反，类型的引入将提高程序的执行效率。因为，Gödel 语言进行类型检查可以缩小匹配执行的搜索范围，对有类型信息的项进行匹配搜索时，只需查找相应类型的搜索域就可以了，从而改善了 Prolog 语言中在整个搜索空间上盲目搜索的情况。Gödel 语言中类型的取值集合决定了论域的搜索范围，它利用类型来决定谓词、函数等所对应的具体操作。例如 :根据外部调用的名称和参数类型，环境才能确定正确的调用操作。同时，随着硬件技术的日新月异，计算机的存储空间越来越大，而由于动态类型检测所带来的存储空间问题就将越来越小。不难理解，在编译器中设置类型机制，依赖于它，类型推导也会带来编程上很大的灵活性，此外，类型变量和数据对象的绑定在程序执行时也是可以按需改变的，它可使程序员从大量的数据类型中摆脱出来，减少了程序员的负担。

4.4 对抽象数据类型的支持

抽象数据类型在软件工程上有许多应用，包括对程序员提供高级别的抽象和不影响现存代码对一个类型执行改变的能力。Gödel 语言提供了大量抽象数据类型上的操作，从而使得程序员很少需要选择数据结构中的个别元素来进行处理。一个模块提供了该类型上的操作集合，模块系统用来隐藏这些操作的执行细节。Gödel 系统模块提供了大量重要的抽象数据类型，如 Program（表示 Gödel 程序的项的类型）和 Theory（表示多态多类一阶理论的项的类型）等，而且许多其它 Gödel 系统模块提供了至少是部分抽象的类型。语言所采用的一个很好方法是为各种应用提供了大量可直接使用的数据类型（大多数是抽象的），同时，也为程序员提供定义他们自己抽象数据类型的机制。

4.5 元程序设计

Gödel 语言采用抽象数据类型来处理元程序设计，发展了相应的方法，但最根本的方法还是在元程序设计中引入类型。目标级表达式包含目标级变量都由元级别的带类型的基项（不含变量的项）来表示。我们称这种表示方法为基本表示。Gödel 语言对基本表示提供了大量的支持，使用基本表示来完成对表示目标级表达式的元级项做处理。例如 Program 是主要抽象数据类型之一，它是一个项（用来刻画一个 Gödel 语言程序对象）的类型。Program 类型的项是一个 Gödel 语言目标程序的一个基本表示。系统模块 Programs 提供了这种抽象数据类型上的大量操作，包括添加和删除一个程序里的语句、访问语言说明，以及许多其它操作。这种使用基本表示来处理元程序设计的方法是说明性的，表明在不同抽象数据类型上操作的谓词都可以被说明性地理解。这种抽象数据类型，连同相应的系统模块所提供的大量有用的操作，表明 Gödel 系统为元程序设计提供了一个合适的环境。

5 结 论

Gödel 语言是一个包含了多态多类高阶类型系统的逻辑程序设计语言，它很好地支持了模块化程序设计、抽象数据类型、元程序设计等程序设计技术，使语言具有比 Prolog 更强的表达能力，在实际应用中也更实用。由于引入了类型系统，使 Gödel 语言拥有很好的说明性语义。前人的研究已经表明，一个语言的说明性表达能力越好，其程序就越容易实现并行。换

(下转第 3438 页)

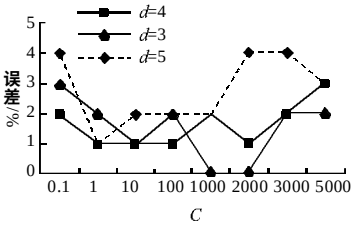

图 2　d 不同值时参数 C 与误差的关系

C=1000 时，SVM 误差较小。

3.3　齿轮运行状态分类器的建立

按照上述的训练样本，依据式(3)求优化系数a_i，按判别式(5)分别建立对应 3 种齿轮状态的 3 个两类分类器 SVM1--SVM3，它们的支持向量的个数分别为 8、10、8。每个两类分类器可从 3 种齿轮状态中识别出是否是正常或处于什么故障状态。将 3 个两类分类器按二叉树形式组合，便成为 1 个可分离 3 种齿轮状态的故障分类器。在分类测试中，先将要测试数据样本 X 输入分类器 1，若 SVM1 判别式$f_1(x)$输出为 1，则确认为正常状态；否则自动输入给分类器 2，若 SVM2 判别式$f_2(x)$输出为 1，则确认为磨损故障；否则自动输入给分类器 3，若 SVM3 判别式$f_3(x)$输出为 1，则确认为断齿故障；若输出不为 1，说明测试样本不属于这 3 种状态，可能出现其他故障。

3.4　故障识别结果

为了验证建立的故障分类器，对减速箱齿轮的正常、磨损和断齿各模拟 3 个时域的数据样本，经小波包分解后得到的特征向量如表 1。将表 1 的特征向量数据输入到故障分类器中，得到的分类结果如表 2。从表 2 得知建立的故障分类器是切实可行的。

表 1　减速箱齿轮的特征向量样本

m^2/s^4x10^{-4}

运行状态	特征向量	频率范围/HZ							
		0~100	100~200	200~300	300~400	400~500	500~600	600~700	700~800
正常	X1	0.1237	0.1954	0.4236	0.4621	0.1828	0.2900	0.5125	0.4263
正常	X2	0.1256	0.2253	0.3908	0.3141	0.1272	0.2523	0.3215	0.2687
磨损	X3	0.2761	0.4334	0.6794	0.8635	0.2680	0.3646	0.5247	0.5765
磨损	X4	0.2708	0.4731	0.8219	0.8993	0.3178	0.4788	0.5891	0.8026
断齿	X5	0.4062	0.5384	1.0769	1.4123	0.4586	0.7141	0.9397	1.1660
断齿	X6	0.4281	0.4457	0.8359	1.2767	0.4011	0.7015	0.9919	1.0025

表 2　识别结果

输入向量	SVM1	SVM2	SVM3	输出结果
X1	1(0.79)	-1(-0.93)	-1(-0.61)	正常
X2	1(1.41)	-1(-0.53)	-1(-1.07)	正常
X3	1(-0.81)	-1(0.64)	-1(-0.46)	磨损
X4	-1(-0.76)	1(0.87)	-1(-1.07)	磨损
X5	-1(-1.37)	-1(-0.62)	1(1.03)	断齿
X6	-1(-0.23)	-1(-0.49)	1(0.43)	断齿

*括号中的数值为 sign()函数的括号内的值。

4　结　论

本文在小波包提取特征向量的基础上，建立了基于支持向量机的多故障分类器。这种诊断方法具有算法简单、可对故障在线分类和故障分类能力强的优点，能较好地解决旋转机械故障诊断中状态的模式识别问题。

参考文献:

[1] 钟秉林,黄仁.机械故障诊断学[M].北京:机械工业出版社, 1998.

[2] Vapnik V N .The nature of statistical learning theory [M]. Germany: Springer-Verlag,1999.

[3] 程正兴.小波分析算法及应用[M]. 西安: 西安交通大学出版社, 1998.

[4] Weston J, Watkins C. Multi-class support vector machines [R]. U K: Royal Holloway University of London, 1998.

[5] Hsu C W, Lin C J. A comparison of methods for multi-class support vector machines[J]. IEEE Trans on Neural Networks, 2002, 13(2):415-425.

[6] Chapelle O, Haffner P, Vapnik V N. Support vector machine for histogram-based image classification [J]. IEEE Trans on Neural Networks, 1999,10 (5):1055-1064.

[7] Steve R.Support vector machines for classification and regres-sion [R]. U K: University of Southampton, 1998.

(上接第 3435 页)

言之，类型系统的引入使 Gödel 语言有很好的执行效率，在许多情况下也确实可以实现并行计算。相信随着 Gödel 语言研究的不断深入，Gödel语言将会对说明性逻辑程序设计产生深远的影响。

参考文献:

[1] 刘椿年,曹德和. PROLOG 语言,它的应用与实现[M]. 北京:科学出版社, 1990.

[2] 陈睿,蔡希尧. 基于类型系统的元数据模型[J]. 软件学报,1995,6 (5):67-70.

[3] Gunter C A. Semantics of programming languages:structures and techniques,Foundation of computer[M].USA:MIT Press, 1992.

[4] Barendregt H. Problems in type theory[DB/OL].http://www. cs. kun.nl/~henk/papers.html.

[5] Donahue J, Demers A. Data types are values[J]. ACM Trans on Programming Languages and Systems, 1985, 7(3):426-445.

[6] Terrence W, Pratt Marvin, Zelkowitz V. Programming languages design and implementation[M].北京:电子工业出版社,1998.

[7] 蒋慧, 张兴元, 王元元. 类型系统的构造、实现及其在程序设计语言中的应用[J].南京大学学报(自然科学), 2001,37(2):24-26.

[8] Hill P M, Lloyd J W. The Gödel programming language[M]. USA: MIT Press, 1994.