学校编码：10384　　分类号_____密级_____
学号：20051302288　　UDC_____

厦门大学

硕士学位论文

Gödel 语言编译系统中实现计算可视化

Implementation of Visual Computing for the Compiler of Programming Language Gödel

李玲

指导教师：赵致琢 教授
申请学位类别：硕士
专业名称：计算机系统结构
论文提交日期：2008 年 5 月
论文答辩日期：2008 年 ___月
学位授予日期：2008 年 ____月

答辩委员会主席：_____

评　阅　人：_____

2008 年 5 月

厦门大学学位论文原创性声明

兹呈交的学位论文，是本人在导师指导下独立完成的研究成果。本人在论文写作中参考的其他个人或集体的研究成果，均在文中以明确方式标明。本人依法享有和承担由此论文而产生的权利和责任。

声明人（签名）：_____

2008 年__月__日

厦门大学学位论文著作权使用声明

本人完全了解厦门大学有关保留、使用学位论文的规定。厦门大学有权保留并向国家主管部门或其指定机构送交论文的纸质版和电子版，有权将学位论文用于非赢利目的的少量复制并允许论文进入学校图书馆被查阅，有权将学位论文的内容编入有关数据库进行检索，有权将学位论文的标题和摘要汇编出版。保密的学位论文在解密后适用本规定。

本学位论文属于

1.保密（），在年解密后适用本授权书。

2.不保密（　）

（请在以上相应括号内打“√”）

作者签名：_____日期：_____年 _____月 _____日

导师签名：_____日期：_____年 _____月 _____日

摘　要

Gödel 语言是继 Prolog 语言之后出现的新型说明性通用逻辑程序设计语言，它是建立在多态多类一阶逻辑基础上的强类型语言。Gödel 语言同 Prolog 语言相比，摒弃了 Prolog 语言中的非逻辑成分，增加了参数型多态多类类型系统，引入了延迟计算等新的语言成分，支持模块化程序设计和元程序设计，具有灵活的技术规则和剪枝操作，使 Gödel 语言成为具有较强功能而且比较实用的一种说明性逻辑程序设计语言。

逻辑程序设计语言与过程性程序设计语言如 Pascal、C 等相比，最大的区别是逻辑程序设计语言提供了一种说明性的编程方法。这使得逻辑程序开发人员在程序设计中不需要考虑“如何解决问题”，而只需要说明程序“要做什么”，从而可以使开发人员将精力放在问题的描述上面，从计算模型的层次上探索问题的求解。它简单统一的语法和语句，类型的合一匹配及回溯操作使程序设计与传统的面向过程的程序设计语言相比更加简洁、明确。但是，这些特点使得对于 Gödel 语言的初学者来讲理解上还比较困难。为此，我们引入计算可视化技术，提出并设计实现 Gödel 语言推理过程跟踪显示器的构想。本文重点讨论 Gödel 语言程序计算可视化问题，详细介绍了 Gödel 语言程序计算可视化跟踪显示器的设计与实现。

Gödel 语言推理过程跟踪显示器不仅在程序调试方面扮演着重要的角色，而且它也可以作为教学工作中的一种对 Gödel 语言程序的辅助理解工具。Gödel 语言推理过程跟踪显示器首先要调用推理过程中产生的调试信息（SLDtree.xml），通过装载调试信息在推理过程跟踪显示器内部将调试信息转换为 CShow 类对象，然后通过操作控制命令控制推理过程的显示。这样，程序编译后执行时，在推理过程显示区我们可以动态地实时观察推理机推理的整个过程。基于这样的一种应用需求，结合 Gödel 语言编译程序的实现方法，我们给出了 Gödel 语言计算可视化的总体结构，并对基于 Gödel 语言计算可视化的编译系统的设计和计算可视化显示等两个部分的实现方法和技术进行了详细的介绍，着重讨论了编译系统中调试信息生成时所采用的方法和技术。Gödel 语言推理过程跟踪显示器是一个独立的应用程序，该程序可以直接从操作系统上运行，也可以从 GPDE（Gödel

Programming Development Environment）上运行。此时，Gödel 语言推理过程跟踪显示器可作为开发环境的一部分。

逻辑程序的重要性早已为人们所熟知，但 Gödel 语言作为新型逻辑程序设计语言，它为人们所熟悉和认同需要一定的时间和实践，尤其需要合适的编译系统和开发环境支持。我们设计和实现的 Gödel 语言推理过程跟踪显示器基本上具备了程序调试器的功能，它的实现不仅使有经验的软件开发人员的工作效率可得到提高，而且可以使 Gödel 语言的教学变得更加清晰明了。Gödel 语言推理过程跟踪显示器将为 Gödel 语言的研究及推广起到一定的作用。相信随着 Gödel 语言编译系统实现技术的不断改进，以及其程序设计环境的逐步完善，Gödel 语言及其应用将会得到进一步的发展。

关键词：Gödel 语言；编译系统；计算可视化；

Abstract

Gödel is a declarative, general-purpose programming language succeeded to Prolog in the family of logic programming languages. Compared with the algorithm-based programming language such as Pascal, C and so on, logic programming language provides a declarative program designing method. So the logic language programmers focus on description the problem and quest for the solution at the level of computing model. Its simple, uniform syntax and sentences, powerful inbuilt features of unification and backtracking often allow algorithms to be encoded more elegantly than in other, more conventional languages. Unfortunately, these excellent features bring about difficulties to novices at Gödel.

Our main purpose is to design a visual system (Gödel reference trace tool) which monitors and traces the inference steps of Gödel language and implement this system by C++. These dissertations focus on the visual computing of Gödel language, the description of tracing Gödel language inference process and implementation of system. This chapter, we begin to study the status quo language Gödel give a brief account.

Gödel reference trace tool not only plays an important role in the process of tracking displays debugging procedure, and it can also be used as a assistant tool in the Gödel programming. Gödel reference trace tool first calls the reference process of debugging information (SLDtree.xml), through loading debug information in the reasoning process of tracking monitors internal debugging information will be converted to CShow class, and then through the process control buttons control the display. In this way, after the implementation of procedures for compiling, in the process of reasoning we can display a dynamic real-time observation of the whole reasoning process of reasoning. Based on such an application needs, with

the method of Gödel language compiler system we give Gödel Computing Visualization of the overall structure, and based on the visualization of Gödel language compiler system design and visual display of part of the realization of the two methods and techniques in detail, focused on compiler system debugging information generated by the use of methods and techniques. Gödel reference trace tool tracking display is an independent application procedures, the procedures can be run directly from the operating system, and can also run on GPDE, when Gödel verbal reasoning process tracking monitors as part of the development environment.

The importance of the logic programming has been familiar to people years before. But to be new logic programming language, Gödel needs some time and practice to make itself to be accepted and popular, which especially requires the support of the appropriate compiler and development environment of this language. Our design Gödel inference trace tool has debug function. The motivation for developing this new tool not only increases the productivity of experienced programmers but also to make the teaching of Gödel more effective. Gödel inference trace tool will help Gödel language’s study and be popular. We believe that Gödel language will be able to receive more attention and obtain a more widespread use with the mature of Gödel programming environment.

Key Word: Gödel language; Compiler system; Visual computing;

目 录

Content

第一章 绪 论

Gödel 语言是继 Prolog 语言之后出现的新型说明性通用逻辑程序设计语言，目前还不为人们所广泛熟知。Gödel 语言程序执行过程的计算可视化实现对该语言的推广、深入研究具有重要的意义。本文重点讨论 Gödel 语言程序计算可视化问题，详细介绍了 Gödel 语言程序计算可视化的设计与实现。本章，我们首先对 Gödel 语言研究现状作简单的介绍。

1.1 Gödel 语言简介

Gödel 语言是一种新型通用逻辑程序设计语言[1]，它充分吸收了逻辑程序设计研究领域的最新成果，对 Prolog 语言存在的缺陷作了诸多改进。首先，它借鉴、继承了 Prolog 语言的诸多要素和成分，并从 Prolog 语言的众多变型如 IC-Prolog、NU-Prolog 等以及 ML 语言、Modula-2 语言引入新的语言设计思想和成分，改进了 Prolog 语言中存在的不足并试图解决其中颇有争议的语义问题。其次，在表达能力上，Gödel 语言与 Prolog 语言相比也有了很大的提高，其理论基础不再限于一阶逻辑的 Horn 子集，而是引入多态多类类型系统[2][3]，将语言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，更易于进行程序开发，且改善了 Prolog 语言那种基于字符匹配的盲目搜索算法所造成的低效率。第三，在控制策略上，Gödel 语言引入了延迟和剪枝机制。延迟机制的引入改变了 Prolog 语言中最左文字优先的计算规则，使语言具有更为灵活的计算规则，提高了语言的执行效率。剪枝操作改进了 Prolog 语言中的 cut 操作，它建立在并发语言的 Commit 机制之上，具有更好的说明性语义[4]，可用来对搜索树进行剪枝从而影响搜索的完全性，进而提高问题求解效率。剪枝操作有效解决了 Prolog 语言中 cut 带来的一些语义问题，如：程序的基本逻辑部分界定的不确定性；在存在否定的情况下，cut 的使用是不可靠的，即从程序逻辑部分的完整性来考虑，可能存在不确定的计算结果[1][5]。此外，Gödel 语言的设计融入了一些程序设计方法和技术。例如，通过对抽象数据类型程序设计的支持可实现面向对象程序设计思想[6]，进一步发展并支持模块化程序设计，能够方便地进行大规模

程序开发，发展 Prolog 语言中元程序的思想，以便在智能程序设计中更有效地支持元级程序设计。

1.2 Gödel 语言的主要机制

Gödel 语言主要包括类型系统、模块系统、控制机制等。为了让人们对 Gödel 语言的语法和语义有所认识，下面将结合 Gödel 源程序实例对各主要机制的特性分别进行介绍。

1.2.1 Gödel 语言的类型系统

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，在增强逻辑程序的可表达性方面提出了引入多态多类逻辑的构想，并体现在 Gödel 语言的设计之中。

Prolog 语言的逻辑基础是一阶逻辑的 Horn 子集，语言设计中没有考虑引入数据表示的类型系统，因而其表达能力十分有限。而 Gödel 语言的逻辑基础则是建立在多态多类(polymorphism many-sorted)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测。引入类型系统主要有以下优点。首先，有了类型系统便于知识的表示。由于知识的预期解释在大多数逻辑程序设计应用中一般是被模型化的，引入类型有助于直接获取应用中的相关知识；其次，有了类型信息，能够进行程序的类型检查从而减少程序错误，提高程序员的程序开发效率；第三，项带上类型信息后，在进行匹配搜索时，只需在对应类型的搜索域中进行查找，可以改善 Prolog 语言基于字符匹配的机械的搜索算法所造成的低效率。

Gödel 语言程序的类型集合是由语言声明中基类（BASE）声明和构造子（CONSTRUCTOR）声明来决定的。若仅定义基类而没有构造子，则所有基类的集合就是语言的类型集合；若至少说明了一个构造子，则由基类 BASE 和构造子 CONSTRUCTOR 出发，得到的基本项(ground term)构成语言的类型集合，该类型集合是可数无限集。

例 1.1 模块 M 定义如下：

```
MODULE    M.                          % 本地模块说明
```

```
IMPORT    Integers.                    % 导入模块说明
BASE      Day,Person.                  % 谓词说明
CONSTRUCTOR   List/1.                  % 类型构造子说明
CONSTANT  Nil: List(a).                % 说明常量Nil是一个空表
          Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,
          Sunday: Day.
          Fred,Bill,Mary: Person.
FUNCTION  CONS : a * List(a) → List(a).        % 函数说明
PREDICATE  Member: a*List(a).                  % 谓词说明
DELAY     Member(x,z) UNTIL NONVAR(x) ∨ NONVAR(z).   % 延迟说明
Member(u,CONS(u, x) ← {True}_1.                % 语句
Member(x,CONS(u, z)) ← Member(x,z).
←Member(Monday,[Monday,Thursday,Sunday])
```

模块 M 中声明基类类型 Day 和 Person，类型构造子 List 之后的整数/1 表示一元构造子，程序中出现的 a 代表类型变元，该模块的类型集合 T_M 为 {Day, Person,List(Day), List(Person), List(List(Day)), …}，即：

$Day \in T_M$，　$Person \in T_M$，　$x \in T_M \rightarrow List(x) \in T_M$

当然，程序中实际所需要的类型并不是可数无限的，而是其中一个有限子集，但这并不影响程序的语义。

此外，Gödel 语言还支持多态类型，引入类型变元带来了参数型多态，可进一步提高程序设计的灵活性。一个含有类型变元的多态符号（常量、函数或谓词）可以看成类型集合中所有不同类型的多个符号的集合。例如，模块 M2 中的常量声明 Nil 可以代表类型为 List(Day)的常量 Nil_{Day}，类型为 List(Person)的常量 Nil_{Person}，类型为 List(List(Day))的常量 $Nil_{List(Day)}$，等等。

多态类型的一个重要优点是谓词、函数和数据结构能重用于任何需要的类型实例，同样的定义能用于表示或计算不同的类型，若语言不支持多态类型则需要在程序中给出不同的定义分别进行处理。但是，多态类型语言的类型检查比传统的单态类型语言更为困难，特别，当涉及到类型变量匹配中类型相同的情况时是很难判断的，并且需要合一的支持操作。

1.2.2 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大规模复杂程序的方法，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。

模块化提供了组织大规模复杂程序的方法，能避免程序的不同部分之间的名字冲突，并且提供隐藏执行细节的方法。模块化程序设计思想可上溯至 1960 年代初期，在命令式语言中已有充分的体现。Gödel 模块系统同样基于这一思想，程序可以由模块集$\{M_i\}^n_{i=0}(n\geq 0)$组成，其中，至少包含一个主模块 M_0，子集$\{M_i\}^n_{i=1}$中的模块由 M_0 直接或间接导入。每个模块可分为输出部分(export part)和本地部分(local part)。模块的输出部分为程序的说明部分包括语言说明、控制说明和模块说明。语言说明中声明一些符号，这些符号可用于此模块的任何地方和其他导入这个模块的模块中的任何地方；控制说明和模块说明中给出一些来自其他模块的符号，这些符号同样可用于本模块的任何地方和其他导入本模块的任何地方。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，同样包括语言说明、控制说明和模块说明，其语言说明声明的符号只在本地模块可用；模块说明给出来自其他模块的可以用以本地模块的符号；控制说明以及那些命题和谓词的定义由本地模块中的语言说明来具体定义。

Gödel 语言可选语言说明由以下部分组成：

MODULE 〈模块名〉

IMPORT 〈导入模块列表〉

BASE 〈类型声明〉

CONSTRUCTOR 〈类型构造子声明〉

CONSTANT 〈常量声明〉

FUNCTION 〈函数声明〉

PREDICATE 〈谓词声明〉

DELAY 〈控制谓词求值的条件〉

PROPOSITION 〈命题声明〉

〈语句规则列表〉

下面通过具体的例子来说明 Gödel 语言模块化的思想。

例 1.2 定义模块 M1 和模块 M2 如下：

EXPORT M1.

IMPORT Lists.

BASE Day, Person.

CONSTANT Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,
Sunday:Day.

PREDICATE Append2： List(a)＊List(a) ＊List(a) ＊List(a).

LOCAL M1.

Append2(x,y,z,u) ← Append(x,y,w) ∧ Append(w,z,u).

MODULE M2.

IMPORT M1.

PREDICATE Member：a＊a ＊List(a).

Member(x,y,z) ← Append3（_,[x｜_],[y｜_],z）.

这里我们定义两个模块 M1 和 M2。模块 M1 包含输出部分（EXPORT）和本地部分（LOCAL）。M1 的输出部分说明它所声明或是导入的符号都可以被其它导入这个模块的模块所引用。

例如模块 M1 声明的类型 Day，Person；常量 Monday，Tuesday，Wednesday，Thursday，Friday，Saturday，Sunday 和谓词 Append2 都可以被模块 M2 所使用。在模块 M1 输出部分声明的导入模块 Lists，使 Lists 模块的输出部分所声明的所有符号对于模块 M1 是可用的。并且任何引入模块 M1 的模块都可以应用模块 M1 的输出部分声明的符号和模块 Lists 输出部分声明的符号，而不需在 IMPORT 部分声明引入 Lists 模块。模块 M1 的本地部分定义声明部分所声明的谓词 Append2，此谓词用到的谓词 Append 已经在模块 Lists 中有定义，因而直接引用它就可以。

通过模块 M1 和模块 M2 的对比，我们可以知道模块 M2 只是声明了本地部分。因此它声明时关键字要用 MODULE 而非 LOCAL。模块 M2 通过 IMPORT 声明引入了所有 M1 声明的符号，其中包括谓词 Append2 和所有由模块 Lists 输出部分声明的符号。模块 M2 包含谓词 Member2 的定义。Member2 表明当且仅