Gödel语言延迟声明语句的语义及其实现方法

曹炳义 赵致琢
(厦门大学计算机科学系 厦门 361005)

摘 要 Gödel语言因语言成份复杂而缺乏严格的语义基础和成熟的编译器,因此推出后它一直发展缓慢。对此采用进化代数描述了其主要语言成分延迟声明语句的过程性语义,然后介绍了依据该语义的具体实现方法并给出运行流程图和C语言描述。最后通过一个例子来具体说明延迟计算在基于扩展Warren机的编译系统中的执行情况。实验结果表明了其可行性。

关键词 Gödel语言,延迟计算,进化代数,过程性语义,扩展Warren机
中图法分类号 TP312 **文献标识码** A

Semantics of Delay Declaration in Logic Programming Language Gödel and its Implementation

CAO Bing-yi ZHAO Zhi-zhuo
(Department of Computer Science, Xiamen University, Xiamen 361005, China)

Abstract The logic programming language Gödel is developed slowly since its appearance due to its complex language components and the lack of rigorous semantic foundation and mature compilers. In this paper, we firstly described the procedural semantics of its delay computation using evolving algebra. Then the specific implementation methods were introduced with a flow chart and a description by C language. Finally the execution of delay computation in the compiler based on extended Warren's Abstract Machine was illustrated. Its feasibility was proved by the implementation.

Keywords Programming language Gödel, Delay computation, Evolving algebra, Procedural semantics, Extended WAM

1 概述

Gödel[1]语言是继Prolog之后出现的通用逻辑程序设计语言,它具有一个多态多类的类型系统、有灵活的计算规则和剪枝操作、支持模块化程序设计,因此相对于Prolog语言具有明显的优势。然而它一直发展缓慢,没有得到广泛的应用。究其主要原因是没有一个成熟的编译器来支持它的应用。最初的Gödel编译器Bristol Gödel由于采用SICStus Prolog来实现,因此其效率较低且无法完全实现Gödel的语言功能,至今仍停留在实验室阶段。Gödel语言的各种语言成分在增强它的表达能力的同时带来了复杂性,命令式程序设计语言的编译方法和技术完全不同于具有递归性、说明性的逻辑程序设计语言,其编译系统遇到实现技术上的困难。其次,各种语言成分的引入也对如何建立合适的语义理论基础提出了挑战。除了在该语言问世之初,由创始人之一P. M. Hill等人发表了一批学术论文[2]外,之后少有相关的研究,国内更缺少相关工作。

我们的研究工作一方面试图为Gödel语言建立起严格的说明性语义和过程性语义,另一方面采用扩展的Warren机来描述它的操作语义,并以此为基础实现Gödel语言的编译系统。本文主要介绍Gödel语言的重要成分之一,延迟计算的处理方法。首先从进化代数[3]的角度抽象地描述了该语言的执行过程模型和延迟计算的过程性语义。然后详细介绍了延迟声明语句的实现方法,给出了目标选择的处理流程和C语言描述。最后通过一个具体例子来说明延迟计算在抽象机中的运行情况。

2 Gödel语言的延迟计算机制

与Prolog相比,Gödel具有灵活的计算规则,不必总是固定的最左文字优先。对如何选取子目标,Gödel语言给出了一些原则,这些原则包括:

(1)可以选择任何系统实现者觉得方便的顺序来扩展子目标;

(2)当多个子目标的延迟计算条件满足时,选择唤醒哪一个被挂起的子目标进行扩展也由系统实现者自己定义;

(3)如果所有的子目标都被延迟,则当前目标无法计算。

延迟计算作为Gödel语言程序执行过程中重要的控制机制,具有多方面的作用。首先,延迟计算类似于过程性语言中的when语句,是when语句在逻辑程序中的变形,它的引入有利于程序的正常终止。其次,延迟计算有利于提高逻辑程序的执行效率。这是因为通过延迟可使子目标得到协同执行(比如在排序操作上),从而使其具有比最左文字优先更高的效率。

3 进化代数

进化代数(evolving algebra)的概念来自于Gurevich,它

到稿日期:2011-06-21 返修日期:2011-09-21 本文受国家自然科学基金(69383004)资助。
曹炳义(1985-),男,硕士生,主要研究方向为逻辑程序设计基础及其应用;赵致琢(1957-),男,博士,教授,主要研究方向为逻辑程序设计基础、计算机科学教育等,E-mail:zzzhao@xmu.edu.cn(通信作者)。

曾被成功地应用于 Prolog 和 Modula-2 等语言的语义描述[4,5]中。根据进化代数理论，程序中的抽象数据可以用论域中的项来表示，论域上允许的操作由函数代表。通过执行函数更新

$f(t_1,\cdots,t_n):=t$

来及时进化。它的执行可以理解为改变或定义函数 f 的参数。进化代数的进化方法决定于一个基于有限集的过渡规则。

If *condition* then *updates*

这里的 *condition* 通常是布尔表达式，它的真值将同时触发所有更新。

4 进化代数描述 Gödel 中的延迟计算

在求解一个子句时，Prolog 的执行机制一般是最左优先，即对子句$\leftarrow B_1,B_2,\cdots,B_n$ 的求解顺序是从左向右，逐个解决子目标 $B_1,B_2,\cdots,B_n$。这种机制便于编译器的编译和推理机的执行。编译器只要按顺序编排好子目标，并让推理机一条条执行，排在当前子目标之前的都是求解过的，排在之后的是还未求解的。由于 Gödel 引入了延迟计算机制，它对子目标的求解顺序就更加灵活，程序员通过显式的延迟声明(DELAY)能部分地控制计算规则，通过在程序中进行延迟声明，明确地给出谓词(子目标)被扩展执行的条件。有经验的程序员可以利用这些规则编写出高效的逻辑程序，但是，Gödel 语言的编译实现难度也相应地增加了。下面，将利用进化代数给出 Gödel 语言中延迟计算的过程性语义描述，并在下一节中介绍一种实现方法。首先给出 DELAY 语句的语法：

DELAY Atom UNTIL Cond

Cond Cond1 | Cond1 {& Cond1} | Cond1 {(Cond1}

Cond1 NONVAR (Variable) | GROUND (Variable)

| TRUE | (Cond)

其中，Atom 为原子谓词或逻辑表达式。Cond 是 condition 的缩写，表示 Atom 被调用执行的条件，条件可以是多个子条件的析取或合取组合。子条件 NONVAR (Variable)代表 Atom 中某变量被实例化，GROUND (Variable)代表 Atom 中某项为基项，TRUE 即无条件为真。

对于 Gödel 程序，我们定义论域 C 代表所有可能的计算状态，其中包含一个当前计算状态 c。对于任何 $s\in C$ 都包含它所代表的计算状态的所有信息。当程序要扩展一个当前子目标并执行时，就生成一个以 c 为父节点的子节点，这样程序的后续计算就可以用一棵以 c 为根的子树来表示。

Gödel 的延迟计算体现在当前计算节点的目标子句的选择中。定义函数

$select$: Goal×Delay×Sub→Lit∪Condition

来描述 Gödel 的目标选择，其中，Goal 是目标的集合，Delay 是所有 DELAY 声明语句的集合，Lit 是符号的集合，Condition 是 Gödel 条件的集合，Sub 是置换的集合。*select* 表示在给定的延迟声明和置换下，从给定目标中选择一个未延迟目标或者一个条件。如果在当前置换和当前程序中，不存在不需延迟的符号或条件，则计算终止。

定义

$next\_term=select(goal(c),delay(P),sub(c))$

其中，$delay(P)$是在程序 P 中出现的延迟声明的集合，$sub(c)$代表当前运行节点下可行的置换，$goal(c)$代表当前运行节点下所有未执行的目标。程序运行过程中调用 *next_term* 来选择下一个要执行的子目标。

下面将定义过渡规则，前面介绍过，过渡规则是用来描述进化代数的具体进化方法的。这里定义两个过渡规则来描述在一般情况下程序选择子目标和扩展执行的条件及方法。从函数 select 的定义可以看出，它描述了一般的目标选择方法，包括具有和不具有延迟声明的子目标。

我们并不能假定 Gödel 程序的运行过程是静态有穷搜索，假设运行过程从初始程序 $\boldsymbol{P}$ 和查询 $\boldsymbol{Q}$ 开始，那么运行过程表现为代数系统在计算过程中动态的进化，其初始条件为：

$C=\{root,c\}$, $goal:=Q$, $P:=P$, $statements:=$NULL

候选语句初始为空，对于一般子句(用户定义谓词)，计算过程为在当前运行状态调用 *next_term* 选择一个子目标，然后根据该子目标在程序中搜寻它的定义语句，选择其中一个进行节点扩展并执行。

过渡规则 1：

If *predicate*(*next_term*)

then *statements*:= *search*(*next_term*,*P*)

意为如果运行 next_term 得到一个用户定义的谓词，就在程序中搜寻该谓词的所有定义语句。

过渡规则 2：

If $t\in$ *statements* then

stetement:= *statement*\{*t*}

Extend *t* EndExtend

如果 t 是 *next_term* 的定义语句，则一个新的节点将被建立，并作为当前节点的子节点。此规则将被当前节点执行多次来扩展候选定义语句。

5 延迟声明的实现方法

上一节给出了延迟计算的抽象的语义描述方法，它是根据 Gödel 语言设计者给出的原则和延迟声明格式抽象出来的，因此适用于任何系统。下面将根据此描述给出延迟计算机制的一个具体的实现方法。首先，详细说明了目标子句的求解过程，然后给出了求解流程图和 C 语言描述。最后，通过一个例子具体说明。

对目标子句$\leftarrow B_1,B_2,\cdots,B_n$ 求解的工作步骤如下：

(1)初始化，在进行目标求解前，假定系统已完成了必要的初始化工作，包括将各个子目标执行入口记录下来；

(2)如果当前找不到还没求解且不满足延迟计算的子目标，则目标子句求解结束；

(3)获取下一个要执行的子目标 t，检查其标识是否需要延迟计算。如果需要，返回步骤(2)；

(4)执行子目标 t 对应的代码段；

(5)转步骤(2)。

从上面的描述中可以看出，整个求解过程是在不断地进行循环操作，直到找不到可以执行的子目标为止。其求解流程和 C 语言描述如图 1 和图 2 所示。

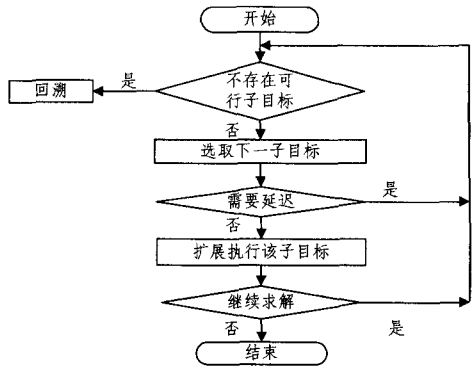

图 1　目标子句求解流程图

```
goal_solve()
{
  do{
    do{
      if(no_applicable_goal())
      backtrack();
else
      t = next_term();
    } while(delay(t))
    extend(t);
    ......
  } while(continue ==1)
}
```

图 2　C 语言描述

下面讨论一个含有延迟计算声明的子句的处理问题。我们使用扩展的 Warren 机指令来刻画此段程序的执行过程。基于扩展的 Warren 机来实现 Gödel 语言是我们研究小组近年来的研究课题，该实现方法具有执行效率高，且能够完全实现 Gödel 语言的各种语言成分的特点，已在理论和实践方面取得了重要进展[6-8]。延迟计算是 Gödel 语言的主要特点之一，它的实现将为该语言的应用提供重要支持。

例 1

DELAY P(x) UNTIL NONVAR(x)

P(x)

Q(a)

←P(x),Q(x)

该程序在基于扩展 Warren 机的编译系统中生成的抽象机指令如下：

DELAY P(x) UNTIL NONVAR(x)	s1:get_variable X_1, A_1 nonvar_check X_1 process_delay P
P(x).	P:trust_me get_variable X_1, A_1 proceed
Q(a).	Q:trust_me get_constant a, A_1 proceed
←P(x),Q(x).	start:allocate init_clause s2 put_ term T_1, s3 put_ term T_2, s4 s2:next_term s5 s3:put_variable Y_1, A_1 call P jump s2 s4:put_variable Y_1, A_1 call Q jump s2 s5:deallocate

指令序列从 *start* 开始执行，由 *s2* 处 *next_term* 指令进行子目标选择，该指令根据抽象机环境栈中的参数状态从 $P(x)$, $Q(x)$ 中选择一个来执行。假设先选的是谓词 $P(x)$，那么抽象机将调用程序段 P，因程序中有关于 P 的延迟声明，将运行 *s1* 做延迟条件检查。因为 x 此时仍为变量，所以条件检查结果为需要延迟。程序跳回 *s2* 执行 *next_term* 选择下一个子目标 $Q(x)$，在执行完程序段 Q 后，x 被约束到常量 a。当再次选择并执行 P 时，根据延迟条件检查结果就不再需要延迟，从而得到解 $x=a$。

结束语　延迟计算是 Gödel 语言的重要成分之一，其过程性语义的建立和基于扩展 Warren 机的实现是该语言研究的一个重要进展，同时也是逻辑程序设计语言编译技术的一次有意义的探索。它为该语言其它语言成分的描述和实现，以及其它逻辑程序设计语言编译系统的实现提供了重要参考。本文的进一步工作将关注子目标的有效选择，改变简单的通过循环判断来确定下一子目标的方法，通过抽象机的动态运行状态来优先选择能够减少尝试次数、尽快获得解的子目标。

参 考 文 献

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

[2] Hill P M, Lloyd J W, Shepherdson J C. Properties of a pruning operator [J]. Journal of Logic and Computation, 1990, 1(1): 99-143

[3] Gurevich Y. Logic and the challenge of Computer Science [M]. New York: Computer Science Press, 1988: 1-57

[4] Gurevich Y, Morris J. Algebraic operational semantics and Modula-2 [C]// Computer Science Logic. 1987, 1988: 81-101

[5] Börger, E. A Logical Operational Semantics of Full Prolog[C]// Computer Science Logic 1989. Springer, 1992: 36-64

[6] Li Hui-qi, Zhao Zhi-zhuo. A polymorphic type system in logic programming [C]// Proceedings of the 3rd International Conference on Intelligent System and Knowledge Engineering. 2008: 125-130

[7] 高伟，赵致琢，等. 逻辑程序设计语言 Gödel 的说明性语义[D]. 厦门：厦门大学计算机系，2009

[8] 林永鹏. Gödel 语言编译系统设计[D]. 厦门：厦门大学，2010

[9] 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

[10] 苗占禄，刘椿年，等. Warren 抽象机(WAM)的数据结构和解释实现[J]. 北京工业大学学报，1999, 25(1): 56-63