doi:10.3969/j.issn.1673-4785.2009.04.011

Gödel 语言控制机制的研究与实现

高 伟,赵致琢,李慧琪,昌 杰
(厦门大学 信息科学与技术学院,福建 厦门 361005)

摘 要:Gödel 语言是在 Prolog 语言基础上发展而来的一种新型逻辑程序设计语言,而控制机制是逻辑程序设计语言的核心内容.针对 Prolog 语言控制机制存在的问题,引出了 Gödel 语言中新的控制机制,包括 DELAY 延迟机制和剪枝操作然后通过实例分析,表明了这些新机制能有效地避免递归谓词的低效或无限循环调用,并能够实现子目标的协同执行,从而提高系统的运行效率.针对这一有效改进,在对 Gödel 语言控制机制比较深入研究的基础上,最后给出了 Gödel 语言控制机制的实现算法.该算法已在研发的 Gödel 语言编译系统中得以实现,通过实例测试,验证了算法具有较高的效率.

关键词:Gödel 语言;控制机制;延迟;剪枝
中图分类号:TP312 **文献标识码**:A **文章编号**:1673-4785(2009)04-0345-07

Research and implementation of the control facility of Gödel language

GAO Wei, ZHAO Zhi-zhuo, LI Hui-qi, CHANG Jie
(College of Information Science and Technology, Xiamen University, Xiamen 361005, China)

Abstract: Gödel, a new logic programming language that emerged from Prolog, has at its core a control facility. After an analysis of problems with the control facility in Prolog, the authors proposed new control facilities for Gödel which include a 'delay computing' and a 'pruning' operation. Examples showed that adoption of the new facilities effectively prevents inefficient or infinite loop calling of a recursive predicate and allows coroutining between subformulas, so that the efficiency of the system is considerably improved. Furthermore, an algorithm was proposed that could provide the control facility in Gödel. The algorithm was applied in the Gödel compiler developed by our group. The high efficiency of the algorithm was verified through testing.

Keywords: Gödel language; control facility; delay; pruning

作为一个典型的逻辑程序设计语言,Prolog 具有语法简单、清晰、可读性好等优点,有较强的可表达性能力.但是,Prolog 语言也存在不足之处,主要是其缺乏类型、执行效率不高、控制机制不完善等,不足以支撑大型软件系统的开发[1].

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 语言存在的缺陷做了诸多改进,引入了类型系统,增加了新的语言成分,这些都使得 Gödel 语言成为一种功能更加强大、高效的说明性逻辑程序设计语言[1-2].众所周知,Prolog 语言控制机制存在不足,针对这一问题,Gödel 语言引入了新的语言成分,改进了控制机制,即延迟计算和剪枝操作.深入分析和理解 Gödel 语言各种新的语言成分,对于语言编译系统的实现具有重要的意义.

作为一种新型的逻辑程序设计语言,Gödel 语言能否真正走向成功,主要取决于其是否有一个高效率的编译或解释实现系统.迄今为止,就是语言的设计者们,也仅实现了一个旨在验证语言功能的原型系统.在对 Gödel 语言的控制机制比较深入的分析和理解的基础上,本文给出了 Gödel 语言控制机制的一种实现算法,从而能够为该语言的完全实现提供支持.

收稿日期:2008-09-04.
通信作者:赵致琢. E-mail:zzzhao@xmu.edu.cn.

1 Prolog 语言中的控制问题

为便于理解和分析,还需要先来回顾 Prolog 语言控制机制存在的问题.

例 1 设有 Prolog 程序 P 和目标子句 G: $\leftarrow B_1 \wedge B_2 \wedge \cdots \wedge B_q$, 则 Prolog 程序的运行机制可概括为

1)从左到右,即逐个解决子目标 $B_1, B_2, \cdots, B_q$;

2)深度优先,即在子目标 B_1 完全解决之前不去处理 $B_2, \cdots, B_q$;

3) 从上到下,即解决 B_1 时先试用程序中它的第 1 条定义子句 C_1,若成功转入 C_1 的体,直到后面解题遇到困难或要求更多的解时才会回到原处去试用后面的定义子句(回溯)[3].

上述过程体现了 Prolog 程序执行时基本的控制机制. Prolog 系统由于严格按照从左到右、深度优先、从上到下的匹配控制机制,而缺乏其他有效的控制机制,这样,在许多情况下将导致系统执行的低效率且可能引起无限循环,也可能破坏系统的完备性.

例 2 考虑下面的 Prolog 程序 $P1$:

1) Append([ ], X,X).

2) Append([H | T],Y,[H | Z]) Append (T, Y,Z).

3) Append3 (X, Y, Z, U)

←Append (X, Y, W) Append (W, Z, U).

Append (X, Y, Z) 是表的联结操作,表示将表 X 放在表 Y 前面构成一个大表 Z;Append3 (X, Y, Z, U) 表示将表 X、Y、Z 按顺序联结成一个大表 U.

如果目标为←Append3([1,2],[3],[4], X). Prolog 程序会将 3 个表合并在一起得到 X =[1,2,3,4];但是,如果把 Append3 定义用于表的分解就会出现问题. 例如,如果目标为:←Append3 (X, [3],[4],[1,2,3,4]),根据 Append3 的定义,第 1 个调用 Append(X, [3],W),将与 Append 的第 1 个定义子句相匹配, X 实例化为[ ],Append 的第 2 个调用失败产生回溯. 第 1 个调用重新匹配后, X 实例化为长度为 1 的表,第 2 个调用再次失败产生回溯. 第 1 个调用再重新匹配, X 实例化为长度为 2 的表,这时,第 2 个调用 Append 成功, X =[1,2]. 由此可见这个计算的时间开销数量级为 X 长度的平方,而不是理想的与 X 长度成正比的数量级. 进一步考虑回溯的情况. 不难看出,回溯使第 1 个调用将 X 实例化为长度为 3 的表,第 2 个调用当然失败. 此时,再次回溯到第 1 个调用, X 实例化为长度为 4 的表,如此下去系统将进入无限循环[4].

在 Prolog 语言中,为解决此问题而引入了附加的控制机制 cut[5],如可在上例中 Append 的第 1 条定义子句后增加 cut 原语(!),改写为

Append([ ], X,X)←!.

虽然这样做可以避免上述的无限循环,但不能提高系统效率,而且影响了程序的可读性,甚至会影响一些调用的正确性. 另外,Prolog 语言编译系统中有时也可采用重新调整定义子句和子目标的次序等方法来解决一些控制问题. 例如,当 Append3 用于表的分解时,可改变定义中的 Append 的子句顺序,以提高效率. 但这一方法缺乏通用性,对一些问题无济于事. 例如. 目标:

←Append3(X,Y,Z ,[1,2,3]).

对 Prolog 系统来说,无论如何重新安排子目标与子句,或是使用 cut,程序在输出列表[1,2,3]的所有可能的划分后都将陷入无限循环. 因此,需要一种既能提高效率又能避免死循环的控制机制,而且,还应能较好地体现程序的说明性语义.

2 Gödel 语言中的 DELAY 延迟机制

针对 Prolog 语言控制机制存在的问题,Gödel 语言对 Prolog 语言的控制机制进行了改进,引入了 DELAY 延迟机制和剪枝操作. 虽然 Gödel 程序的执行和 Prolog 程序一样都是采用最左深度优先搜索策略的反驳—消解方法;但其推理过程在从左到右、深度优先的控制基础上,合理使用延迟计算和灵活运用剪枝操作,可以解决上述死循环的问题.

下面是一个典型的 Gödel 语言计算排序问题的程序实例 $G1$.

```
MODULE      Slowsort.
PREDICATE   Slowsort: List( a ) * List( a );
            Permutation:List( a ) * List( a );
            Sorted:List( a );
            Delete : a * List( a ) * List( a ).
DELAY       Slowsort( x,y ) UNTIL
              NONVAR( x ) \/ NONVAR ( y );
            Sorted([ ]) UNTIL TRUE;
            Sorted([ _ | x ]) UNTIL NONVAR( x );
            Delete( _,y,z ) UNTIL
              NONVAR( y ) \/ NONVAR( z ).
Slowsort( x,y ) ← Sorted( y ) Permutation( x,y ).
Sorted([ ]) ← {True}_1.
Sorted([ _ ]) ← {True}_1.
```

Sorted([x,y | z]) ← { x ≤ y }_1 Sorted([y | z]).
Permutation([ ],[ ]).
Permutation([x | y],[u | v])
　　← Delete (u,[x | y],z) Permutation (z,v).
Delete (x,[x | y],y) ← {True}_2.
Delete (x,[y | z],[y | w]) ← { x　y }_2 Delete(x,z,w).

模块 Slowsort 给出了实现排序的 Gödel 语言程序. 程序中 Slowsort(x,y) 表示对列表 x 排序后得到列表 y, 其中,调用的谓词 Permutation (x,y) 表示列表 y 是列表 x 的一个排列,而谓词 Sorted (x) 表示 x 是一个有序列表. DELAY 说明表示当 x 和 y 至少有一个已约束时,SlowSort 才能计算;而 Sorted (x) 调用只有当列表 x 为空或者在前2个元素已知的情况下才可以执行,否则延迟计算. 谓词 Sorted 有3个定义子句并且都包含了剪枝号为1的剪枝,这表示如果在匹配时有一个子句匹配成功,则不再尝试匹配其他2个含有相同剪枝符号1的子句. 从这个例子不难看出 Gödel 语言引入的这些新机制使语言具有更强的说明性,同时语言的控制也变得更加灵活[3]. 下面来详细介绍 DELAY 延迟机制.

2.1　DELAY 延迟说明的语法和语义

DELAY 延迟说明的语法形式为

DELAY Atom UNTIL Cond.

其中,Atom 是原子. Cond 语法规则如下:

Cond→Cond1 | Cond1 { ∧Cond1} | Cond1 { ∨ Cond1},

Cond1 → NONVAR (Variable) | GROUND (Variable) | TRUE | (Cond).

其中,Cond 对原子中出现的变量延迟调用条件进行说明,Variable 是出现在 Atom 中的变量,关键字 NONVAR 表示延迟条件是变量,已实例化,关键字 GROUND 表示变量为基本项. 程序通过 DELAY 说明限制谓词过程的调用,调用如果满足则进行处理,否则延迟挂起调用.

将前面的 Prolog 程序 *P*1 改写为带 DELAY 延迟说明的 Gödel 程序 *G*2 如下:

PREDICATE　Append: List(a) * List(a) *
　　List(a);
　　Append3: List (a) * List (a) *
　　List(a) * List(a).
DELAY　Append(X,_,Z) UNTIL
　　NONVAR(X) ∨ NONVAR(Z).
Append([ ], X,X) .
Append([H | T],Y,[H | Z]) ←Append (T,Y,Z).
Append3(X, Y, Z, U)←Append(X, Y, W) ∧
←Append(W, Z, U).

程序中的 DELAY 说明表示调用 Append 时第1个参数和第3个参数必须至少有一个是非变量,如果满足不了该条件,则调用被延迟挂起. 系统在引起延迟的变量上做标记,以示某过程调用等待此变量被实例化为非变量,一旦标记变量被实例化,所有其上等待的过程调用立即被唤醒,成为当前优先执行的子目标. 例如:←Append(X ,[2],[1,2]) 满足 DELAY 说明,过程被成功调用. 然而对于:←Append(X,[2], Y),由于第1个参数和第3个参数同时为变量不满足 DELAY 说明,过程调用将被延迟,并在这些变量上做标记. 一旦 X、Y 中任一变量被实例化为非变量,延迟的 Append 立即被唤醒重新调用执行.

2.2　DELAY 延迟机制的使用

运用 DELAY 延迟机制能确保某些调用被充分实例化后才运行,从而增强程序的可终止性以避免死循环. 另外,DELAY 延迟的使用可以实现子目标的协同执行,从而增强程序的有效性. 下面通过例子来说明这2点.

1)考察前面的 Prolog 程序 *P*1 中的目标:←Append3(X, [3],[4], [1,2,3,4]).

在 Gödel 程序 *G*2 中的执行情况:

第1个调用执行 Append (X , [3], W),发现 Append 的第1、第3个参数同时为变量,不满足 Append 的 DELAY 说明,调用被延迟,并在第1、第3个变量上做标记. 然后,调用 Append3 的第2个子目标 Append(W ,[4],[1,2,3,4]),它满足 DELAY 说明,将 W 实例化为[1 | T_1] ,立即唤醒 W 上的延迟调用,重新执行第1个 Append 调用, Append (X , [3], [1 | T_1]) 满足 DELAY 说明,合一成功将 X 实例化为 [1 | T_2] . 接着调用 Append(T_2, [3], T_1) , 调用被延迟;再选下一个子目标 Append(T_1, [4], [2,3,4])继续执行,如此往复执行直到目标合一成功. 从调用的执行过程可见,2个 Append 调用是并发执行的,它们的时间开销与 X 的长度成正比,要低于 Prolog 系统中的时间数量级(X 长度的平方),因此,系统效率将得到很大提高. 注意:Prolog 程序 *P*1 调用目标←Append3 (X,Y,Z, [1,2,3]),将陷入无限循环.

但是,在 Gödel 程序 *G*2 中由于存在 Append 的 DELAY 说明,该目标在输出所有可能的划分后正常结束(具体调用过程不再累述). 由此可见,DELAY

说明即可避免 Append 过程被低效地执行,也可使其避免无限循环.

2)下面通过对 Gödel 程序 $G1$ 的实例运行可更有效地说明 Gödel 程序中 DELAY 延迟机制的合理使用可以促进子目标的协同执行,从而增强程序的有效性.

设程序目标为←Slowsort([6,1,5,2,4,3], x).

第1个调用 Sorted(x),发现 x 为列表变量,不满足 Sorted 的 DELAY 说明,调用被延迟并在变量 x 上做标记.然后,调用 Slowsort 的第2个子目标 Permutation([6,1,5,2,4,3], x),满足 Permutation 的延迟说明,调用成功,x 被实例化为[6|v_1],但还是不满足 Sorted 的 DELAY 说明,故继续执行 Permutation 的调用.当 x 被实例化为[6,1|v_2]时,满足 Sorted 的 DELAY 说明,此时立即唤醒 x 上的延迟调用,重新执行 Sorted(x)调用.由于 x 的前2个元素[6,1]不是有序的,程序放弃排列结果[6,1|v_2],Sorted(x)将重新延迟.Permutation 调用回溯,重新产生排列;当 x 被实例化为[1,6|v_2]时又满足 Sorted 的 DELAY 说明,并立即唤醒 x 上的延迟调用,重新执行 Sorted(x)调用.这时,由于 x 的前2个元素[1,6]是有序的,Sorted 进行递归调用.依此循环处理,程序将很快得到[6,1,5,2,4,3]的排序结果[1,2,3,4,5,6].

如果没有采用上面基于 Sorted 上的 DELAY 说明,那么程序运行时每次都要完整地算出列表 x 上的每个元素,而长度为 n 的列表的排列个数为 $n!$,所以,计算时间复杂度为 $O(n!)$.然而,如果采用 DELAY 说明后执行这样的目标所花的时间将大大减少,时间复杂度为 $O(n^2)$.可见 DELAY 延迟计算促进了子目标的协同执行,从而产生的惰性计算的效果,使得 Gödel 程序的排序比对应的 Prolog 程序的排序效率更高.

3 Gödel 语言中的剪枝操作

在 Gödel 语言的控制机制中,除了 DELAY 延迟机制外,还有一个重要的控制机制,即剪枝操作.有点类似于 Prolog 中的 cut 机制,Gödel 语言也提供了对反驳—消解树进行不完全搜索的方法,即剪枝操作.但剪枝操作与 cut 不同,剪枝操作有效地解决了 Prolog 中 cut 带来的诸如程序的逻辑部分界定的不准确性、存在否定时 cut 造成求解结果不可靠等问题.剪枝与 cut 机制最本质的区别在于 cut 机制需要程序员了解程序的归结过程,而剪枝只需要程序员声明剪枝条件,无需了解归结的控制过程.这既减轻了程序员的负担,又增强了程序说明性语义与过程性语义的一致性.

3.1 commit 剪枝

Gödel 语言的剪枝(commit)机制是对 Prolog 中 cut 的改进,它建立在并发语言的 commit 机制之上,具有更好的说明性语义,可用于对搜索树进行剪枝,进而影响搜索的完全性,以提高问题求解的效率.Gödel 语言剪枝算子是 commit,一般形式为{…}_n(n 代表整数标号),有2个特殊形式:一解剪枝(one-solution commit)和条形剪枝(bar commit).一解剪枝形式为{….},当它找到{}所包含的公式的一个解之后返回,其他可能的解将被剪除.条形剪枝形式为"|",其含义相当于合取,其作用范围是语句体内出现在其左边的公式,只求出其辖域内公式的一个解,而搜索树中其他谓词定义中包含"|"的语句的相关分支均被剪除.Gödel 语言中剪枝操作的2种特殊形式经过处理,在计算机内部它们都可用一般形式{…}_n 来表示.一个剪枝操作{Formula}_n 的括号{...}指明了该剪枝在语句体中的作用范围,而标记 n 是剪枝号(标签),用于标识某一剪枝.当公式 Formula 成功时,这个定义语句中的所有其他包含相同标记 n 的语句所对应的树枝将被剪掉.条形剪枝操作记为"|",每条 Gödel 程序语句最多只能含有一个"|"."|"的作用范围就是在语句体内出现在"|"左边的那个公式.如果一个程序中某语句含有"|",那么该程序将在求出"|"辖域中公式的一个解后,剪除由同一谓词定义的语句中包含了"|"的语句所产生的搜索树中的所有其他分支.举例如下,设某程序中有3条语句如下所示:

Partition([],_,[],[]) ← |,

Partition([x | xs],y,[x | ls],bs) ← $x \leqslant y$ |

Partition (xs, y, ls, bs),

Partition ([x | xs],y,ls,[x | bs]) ← $x > y$ |

Partition (xs, y, ls, bs).

上述语句定义了快速排序程序中的分割函数,即当第1个参数列表不为空时,根据表中元素 y(列表元素)的值把该列表分为比 y 大的部分和比 y 小的部分,分别存在第4个和第3个参数列表中.3条语句中都含有"|",它的目的是保证这3条语句互斥.即当列表为空时第1条语句被执行,而第2和第3条语句对应的分枝被剪除,因为此时第2和第3条语句根本没有必要执行;而当列表不空时第2和第3条语句中的某一句将被执行,另外2条程序语

句对应的分枝将被剪除.如果没有这个“|”,那么3条语句将被一起尝试执行,这将浪费大量的时间.由这个例子可以看出,剪枝操作在保证程序正确性的基础上,极大地提高了执行效率.

已经知道,条形剪枝是commit剪枝的特例,它可以转换为commit剪枝.转换方法如下:将“|”左边的子目标作为commit剪枝的条件(对于事实子句,此时条件为True),并附加剪枝号.例如,可以将上面的条形剪枝语句转换为如下commit剪枝语句:

Partition([ ],_,[ ],[ ]) ← {True}_1,

Partition([x | xs],y,[x | ls],bs)

← {$x \leqslant y$}_1 ∧ Partition(xs, y, ls, bs),

Partition([x | xs],y,ls,[x | bs])

← {$x > y$}_1 Partition(xs, y, ls, bs).

True为系统关键字,即剪枝条件永远为真.因此,对于第1条子句只要合一成功即可剪除第2条子句和第3条子句对应的搜索分枝.

3.2　DELAY说明和剪枝的应用

如同Prolog中的cut一样,剪枝操作也有可能会剪去有效答案,但是,在Gödel程序中可以通过设计一个好的DELAY说明来控制剪枝.一个重要的启发就是DELAY说明应足够强,从而避免由于过早地剪去一个不合适的树枝而导致无法预料的错误.下面的Gödel程序$G3$通过改进程序$G1$中的对Delete的延迟声明有效地说明了这一点.

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

DELAY Delete (x, [y | _],_) UNTIL NONVAR (x) ∧ NONVAR(y).

Delete (x,[x | y],y) ← {True}_2.

Delete (x,[y | z],[y | w]) ← { $x \neq y$ }_2 ∧ Delete (x,z,w) .

假设Delete的DELAY说明被删除了,现在来考察目标:

← Delete(x ,[1,2,3], y) ∧ x =2.

目标中的第1个原子与Delete定义里的第1条语句的头部相匹配,因此,系统求得x = 1并将剪掉其他的匹配语句,然而接下来调用1=2会导致目标产生不可预料的错误.

现在假设Delete调用已经根据程序中的Delete的延迟说明进行了延迟,从而先执行x =2,然后唤醒x上的延迟调用Delete(2,[1,2,3], y),那么,目标就能返回预期的结果.由此可见,Gödel语言中的DELAY延迟机制既能够发挥剪枝的有效性,又保证了程序的完备性.

剪枝运算在编写风格良好的Gödel程序中较少使用,代之以否定或if-then-else等说明性结构,这主要是为了保证程序具有清晰的说明性语义.尽管剪枝操作可能影响程序的说明性语义,但合理的剪枝可以使程序执行速度和问题求解的效率大大提高,有着优越的过程性语义,在逻辑程序设计语言中仍有重要的意义.

4　Gödel语言控制机制的算法设计与实现

由上面所述知,Gödel语言中通过引入新的控制成分DELAY延迟机制和剪枝操作,完善了逻辑程序设计语言的控制机构,可有效地提高系统的效率.然而,由于实现难度较大,迄今为止,即使是语言的设计者们,也仅实现了一个旨在验证语言功能的原型系统,一些新的语言成分并没有实现.下面,对Gödel语言控制机制的实现算法进行讨论,希望能够为该语言的完全实现建立一些方法和技术基础.

Gödel语言的控制机制可概括为

1)从左到右选取子目标进行判定.如子目标有DELAY声明,先判断DELAY条件是否满足再决定是否执行,若不满足,则先取下一个子目标;

2)采用深度优先的方法对搜索树进行搜索、匹配和判定;

3)按程序中子句的顺序选取目标的可匹配项进行匹配尝试和处理,当一个子目标匹配成功时,若有剪枝操作,则不对与其剪枝号相同的其他可匹配项进行匹配尝试,即剪掉搜索树的部分分枝.

Gödel语言中的控制机制是Gödel语言实现的关键部分之一.原理上,控制机制只是实现带有延迟和剪枝的从左到右、深度优先、从上到下的匹配算法,较为简单.但是,在实现一个真正实用的系统时,还得考虑各种因素,实现起来比较复杂[6-7].Gödel语言中的控制机制作为编译系统实现的一部分,其实现在技术上是将程序运行时的各种信息分为多栈存储(运行栈、环境栈、解除栈和延迟栈),以此来简化运行栈的控制,从而使控制机制获得较高的效率.借助前面实例分析的流程获得的思想,下面给出Gödel语言控制机制的算法流程图(图1).

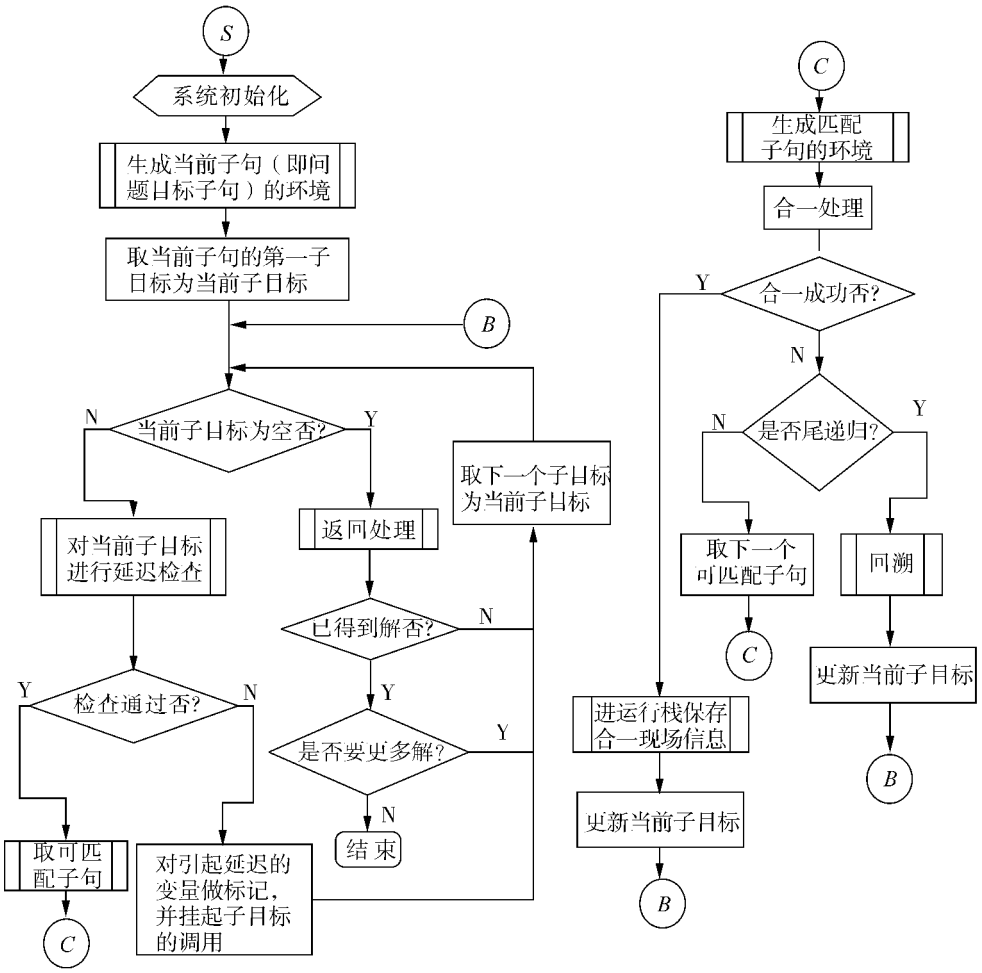


图1　Gödel 语言控制机制的算法流程图

Fig. 1　Outline of the control facility algorithm for Gödel

流程图中各个模块的主要功能如下：

1）生成当前子句的环境模块：根据当前子句中的变量信息在环境栈中开设变量空间；

2）返回处理模块：当一个子目标求解成功后，本模块则返回到该子目标的父节点，取下一子目标进行求解. 如果系统返回到问题目标节点（即消解树的根），则系统给出问题的求解结果——成功或失败，最后再询问是否要更多结果还是退出程序；

3）对当前子目标进行延迟检查模块：对当前子目标进行延迟条件检查，如果不满足 DELAY 说明，则挂起该子目标的调用直到条件满足，并对引起延迟的变量做标记保存在延迟栈中；

4）取可匹配子句模块：搜索与当前子目标匹配的子句，在搜索之前必须检查该子目标的上一个尝试合一子句是否含有剪枝符号，如果有则函数在搜索过程中不再考虑含有同一剪枝符号的程序子句，以此达到剪枝的目的；

5）回溯模块：当前子目标如果与所有的可尝试合一子句均合一失败，则必须回溯，即回溯到上一次合一现场；

6）进入运行栈保存合一现场信息模块：将消解过程中的一系列合一现场信息，即在每一次合一成功后，将表征合一现场的程序对象推入运行栈中. 并在环境栈中保存合一时产生的变量置换信息，以便在回溯或返回时恢复合一前的现场. 在子目标合一成功后，若将某一变量实例化了，而此变量曾使某一子目标的调用进程不满足 DELAY 说明被延迟挂起，那么这时要将此进程唤醒成为当前子目标，然后重新匹配.

5　结束语

本文研究的 Gödel 语言控制机制的算法已在研发的 Gödel 语言编译雏形系统中得以实现[2-3]，同时，通过计算可视化功能的实现和部分实例的测试[8-11]，验证了本文的算法具有很高的效率. 然而，由于 Gödel 语言引入了许多新的语言成分，要做到 Gödel 语言的完全实现，目前还有很多工作要做. 其中，变量的多态性、并行化是很重要的 2 个方面. 变

量的多态性可以参考面向对象程序设计的编译技术尝试实现,需要建立动态类型系统,目前建立的动态类型系统还不能与推理机一起通过比较复杂的测试实例.对于Gödel语言运行机制的并行化工作可以分为2个阶段,即首先实现单机多进程的运行,然后实现多机联合运行.这将是本课题组下一步需要重点着手开展的工作.

参考文献:

[1]HILLl P M, LLOYD J W. The Gödel programming language [M]. London:MIT Press,1994:79-99.

[2]李松斌. Gödel语言编译系统中推理机的设计与实现[D].厦门:厦门大学,2007.
LI Songbin. Design and implementation of inference machine for Gödel compiler[D]. Xiamen: Xiamen University, 2007.

[3]苏剑煌. Gödel语言编译系统的设计与实现[D]. 厦门:厦门大学,2007.
SU Jianhuang. Design and implementation for the compiler of programming language Gödel[D]. Xiamen: Xiamen University,2007.

[4]肖　楠.具有并发延迟功能的PROLOG语言控制策略的设计与实现[J].小型微型计算机系统,1989,10(3):36-43.
XIAO Nan. Design and implementation for the control facility of Prolog languange with the functional of concurrent and delay[J]. Mini-Micro Systems,1989,10(3):36-43.

[5]刘椿年,曹德和. PROLOG语言,它的应用与实现[M].北京:科学出版社,1990:10-23.

[6]LLOYD J W. Foundation of logic programming[M]. [S. l.]:Springer-Verlag, 1984:5-15.

[7]HILL P M. The completion of typed logic programs and SLD-NF resolution [C]//Proceedings of the 4th International Conference. [S. l.],1993:321-326.

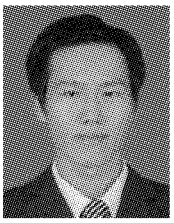
[8]王良霖. Gödel语言程序计算的可视化研究[D].厦门大学,2008.
WANG Lianglin. The study for the visualization of Gödel programs[D]. Xiamen: Xiamen University,2008.

[9]李　玲. Gödel语言编译系统中实现计算可视化[D].厦门:厦门大学,2008.
LI Ling. Implementation of visual computing for the compiler of programming language Gödel[D]. Xiamen: Xiamen University,2008.

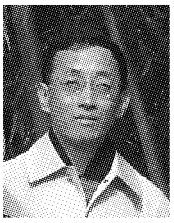
[10]涂序彦. 广义智能系统的概念、模型和类谱[J]. 智能系统学报,2006, 1(2):7-10.
TU Xuyan. Concept,model and kinds of generalized intelligent system[J]. CAAI Transactions on Intelligent Systems,2006, 1(2):7-10.

[11]杨春燕,蔡　文.可拓信息-知识-智能形式化体系研究[J].智能系统学报,2007, 2(3):8-11.
YANG Chunyan, CAI Wen. A formalized system of extension information-knowledge-intelligence[J]. CAAI Transactions on Intelligent Systems,2007, 2(3):8-11.

作者简介:



高　伟,男,1985年生,硕士研究生,主要研究方向为逻辑程序设计语言Gödel及其程序设计环境.



赵致琢,男,1957年生,教授,硕士生导师,主要研究方向为计算模型与分布式基础算法、逻辑程序设计语言、计算机科学教育研究.先后获得2000年福建省优秀教学成果奖一等奖、2001年国家级优秀教学成果奖二等奖.

李慧琪,女,1973年生,博士研究生,主要研究方向为逻辑程序设计语言Gödel及其程序设计环境.