Visual Prolog的回溯机制分析

雷英杰，王 涛，赵 晔，王建勋
(空军工程大学 导弹学院，陕西 西安 713800)

摘 要：回溯机制是逻辑程序设计的重要设施。回溯本身是一种获得目标所有可能解的良好方法。在考察Visual Prolog回溯机制作用原理的基础上，通过若干实例，详细阐述了回溯机制所遵循的4个基本原则，即自顶向下逐层搜索原则、从左到右顺序扫描原则、规则匹配原则、事实匹配原则，给出了回溯机制的循环实现方法，深入分析了回溯过程可能发生的各种情况，从而揭示出回溯机制的本质特性和应用机理。结论指出回溯机制具有副作用，需要利用截断机制、失败谓词等对搜索过程进行仔细控制。

关键词：Visual Prolog；逻辑程序设计；编程语言；AI；专家系统
中图分类号：TP182　　**文献标识码**：A　　**文章编号**：1009-3516(2004)05-0080-05

Prolog是人工智能与专家系统领域最著名的逻辑程序设计语言。Visual Prolog意指可视化逻辑程序设计，是基于Prolog语言的可视化集成开发环境，是Prolog开发中心(PDC)最新推出的基于Windows环境的智能化编程工具[1]。Visual Prolog具有模式匹配、递归、回溯、对象机制、事实数据库和谓词库等强大功能[2]。它还支持模块化与面向对象程序设计、系统级编程、文件操作、字符串处理、位级运算、算术与逻辑运算，以及与其它编程语言的接口[3]。

本文在考察Visual Prolog回溯机制作用原理的基础上，通过实例，给出该回溯机制的实现分析，从而揭示回溯机制的本质特性和应用机理。

1 回溯的基本原理

回溯是指示Visual Prolog到何处去寻找问题答案的机制。这种机制使得Visual Prolog具有了通过所有已知事实和规则进行搜索求解的能力。合一(unifcation)是与回溯机制密切相关的一个重要概念。当Visual Prolog试图执行一个子目标written_by(X, Y)时，为了在程序中获得匹配，必须测试每一个written_by子句。在尝试把参数X和Y与每一个written_by子句的参数进行匹配时，Visual Prolog将从头至尾对程序进行搜索。当它找到匹配目标的子句时，将数值绑定到自由变量，以便使目标和子句是同一的；目标被称为与子句合一(unify)。这一匹配操作就叫做合一。

所谓回溯(backtrack)，是指使用“回退再试”对给定问题找到所有可能解的一种方法。当Visual Prolog开始为求解一个问题(或目标)寻找答案时，往往要在多种可能情况中做出抉择。它首先在分支点(即回溯点)设置好标志，然后选择要追踪的第一个子目标。如果该子目标失败，Visual Prolog将回退到上一个回溯点尝试另一个目标。下面看一个例子：

```
PREDICATES
    likes(symbol,symbol)
    tastes(symbol,symbol)
    food(symbol)
```

收稿日期：2004-05-10
基金项目：国家教育部高等学校骨干教师资助计划项目(GG-810-90039-1003)
作者简介：雷英杰(1956-)，男，陕西渭南人，教授，博士生导师，主要从事智能信息处理与智能系统、智能决策等研究.

```
CLAUSES
    likes(bill,X):-
        food(X),
        tastes(X,good).
    tastes(pizza,good).
    tastes(brussels_sprouts,bad).
    food(brussels_sprouts).
    food(pizza).
```

这个程序由两组事实和一个规则组成。规则由likes关系表示,描述Bill喜欢味道好的食物。为了理解回溯机制的工作原理,我们给出下面要进行求解的目标:

```
GOAL
    likes(bill, What).
```

1) Visual Prolog开始试图满足一个目标时,总是从程序的顶部开始寻求匹配。

在本例,Visual Prolog将从顶部开始为子目标likes(bill, What)寻找一个匹配的解。首先它找到与程序中第一个子句的匹配,变量What与变量X合一。与规则头的匹配促使Visual Prolog试图满足整个规则。与此同时,它迁移到规则体,调用第一个子目标:food(X)。

2) 当进行一个新调用时,匹配这个调用的搜索也从程序的顶部开始。

在满足第一个子目标的搜索中,Visual Prolog从顶部开始,试图匹配程序处理过程中遇到的每一个事实或规则头。它在表示food关系的第一个事实处找到这个调用的匹配,变量X被绑定为值brussels_sprouts。因为对调用food(X)来讲,存在着不止一种可能的答案,所以Visual Prolog对下一个事实food(brussels_sprouts)设置一个回溯点。这个回溯点保留一个Visual Prolog开始搜索下一个可能的food(X)匹配的轨迹。

3) 当一个调用已经找到一个成功的匹配时,则调用成功,且依次试验下一个子目标。

在X被绑定为brussels_sprouts的情况下,进行的下一个调用是

tastes(brussels_sprouts, good)

Visual Prolog再次从程序的顶部开始搜索,试图满足这个调用。因为没有找到匹配的子句,所以调用失败,并且Visual Prolog启动自动回溯机制。当回溯开始时,Visual Prolog回退到所设置的最近一个回溯点。在这种情况下,Visual Prolog返回到事实food(brussels_sprouts)。

4) 一旦变量在子句中已经被绑定,则释放它的唯一办法就是通过回溯。

当Visual Prolog回退到回溯点时,它释放所有在该点之后所设置的变量绑定,并开始为最初的调用寻求另一个解。调用是food(X),所以对X绑定的brussels_sprouts被释放。Visual Prolog现在试图从停止的地方重新求解这个调用。它找到一个匹配的事实food(pizza)且返回,这一次绑定变量X的为值pizza。

现在,Visual Prolog带着新的变量绑定迁移到规则的下一个子目标。启动一个新的调用tastes(pizza, good),从程序的顶部开始搜索。这次发现一个匹配,目标成功返回。

因为目标中的变量What与规则likes中的变量X合一,而且变量X被绑定为pizza,所以变量What也是pizza,于是Visual Prolog报告这个解:What = pizza。利用回溯机制,Visual Prolog不仅能找到问题的第一个解,而且可以找到所有的解。回溯的副作用是可能导致Visual Prolog给出多余的答案,而且Visual Prolog不能区分实质上相同的两个解,因此需要仔细控制Visual Prolog的搜索过程。

2　回溯的基本原则

为了更清楚地理解Visual Prolog回溯机制的工作情况,我们设定如下程序:

```
DOMAINS
    name,thing = symbol
PREDICATES
    likes(name, thing)
    reads(name)
```

```
    is_inquisitive(name)
CLAUSES
    likes(john,wine).
    likes(lance,skiing).
    likes(lance,books).
    likes(lance,films).
    likes(Z,books):-
        reads(Z),
        is_inquisitive(Z).
    reads(john).
    is_inquisitive(john).
```

再考察下面的目标,它由两个子目标组成:

GOAL likes(X, wine), likes(X, books)

当评估这个目标时,Visual Prolog 构造一个目标搜索树,其中各个子目标将按顺序成为该树的左、右支树。在目标树搜索过程中,Visual Prolog 遵循以下 4 个基本原则:

1) 自顶向下的逐层搜索原则。子目标必须按自顶向下的顺序被满足。

2) 从左到右的顺序扫描原则。谓词子句根据它们在程序中出现的顺序,从左到右,进行测试。

3) 规则匹配原则。当子目标与规则头匹配时,接下来将测试规则体。规则体进而又将变成一系列新的要被满足的子目标。

4) 事实匹配原则。当在目标树的某一个末端节点(即叶节点)找到一个匹配的事实时,一个目标就得到满足。

对于上例,根据第二个基本原则,Visual Prolog 确定在子句满足时将使用那个子目标。如果子目标 likes(X, wine) 匹配事实 likes(john, wine),则绑定 X 为值 john。而对第二个子目标的调用将以绑定 X = john 来开始一个完整的、全新的搜索过程。第一个子句 likes(john, wine)不匹配子目标 likes(X, books),因为 wine 与 books 是不同的。因此,Visual Prolog 必须尝试下一个子句。但是,lance 不匹配变量 X(因为在这种情况下,X 绑定为 john),所以搜索继续到所定义谓词 likes 的第三个子句:

likes(Z, books):- reads(Z), is_inquisitive(Z).

参数 Z 是变量,因此可以与 X 匹配。第二个参数一样,所以这个调用与规则头匹配。当 X 匹配 Z 时,参数是合一的。由于参数合一,Visual Prolog 将认为变量 X 的值与变量 Z 的值相同,故变量 Z 的值也为 john。接下来的搜索将遵照回溯的第三个基本原则来确定。

目标树包括子目标 reads(Z)和 is_inquisitive(Z),这里 Z 绑定为值 john。Visual Prolog 搜索匹配两个子目标的事实,从而得到最终的结果目标树。

Visual Prolog 根据搜索开始的情况,以不同的方式使用搜索过程的结果。如果目标是来自规则体中子目标的一个调用,则在调用返回之后,Visual Prolog 试图满足规则中的下一个子目标。如果目标是来自用户的询问,则 Visual Prolog 直接回答:X = john。

正如在上例中所看到的,一旦满足目标,Visual Prolog 将回溯寻找其它替代方案。如果子目标失败,它也回溯,希望以失败子目标被其它子句满足这样一种方式来重新满足以前的子目标。

3 回溯的循环实现

回溯是一种获得目标所有可能解的良好方法。但是,即使目标没有多个解,也可以用循环来实现回溯。这可以通过定义下列两个子句谓词来实现。

```
repeat.
repeat :- repeat.
```

这样,就使得 Visual Prolog 的控制结构认为它有无限多个不同的解。谓词 repeat 的目的是允许回溯无限。下面的例子是通过使用 repeat 来连续接收字符和打印字符,直到用户按下回车键。

```
PREDICATES
    repeat
    typewriter
CLAUSES
    repeat.
    repeat:-repeat.
    typewriter:-
        repeat,
        readchar(C),/* 读一个字符,绑定到变量C */
        write(C),
        C = '\r',!. /* 是回车键吗? 否,则失败 */
GOAL
    typewriter(),nl.
```

上例程序显示repeat如何工作。规则"typewriter :- ..."描述一个过程,识别从键盘键入的字符并输出该字符到屏幕上,直到用户按下回车键结束。

谓词typewriter工作如下:①执行repeat(不做任何事);②读字符到变量C;③输出C;④检查C是否为一个回车键;⑤如果是,则结束。否则,回溯并寻找另外的解。子目标write或readchar都不产生备选的解。因此回溯始终到repeat,总会有备选的解;⑥程序再次向前进行,读下一个字符,输出它,并且检查它是不是回车键。

注意,当回溯通过绑定C的readchar(C)时,就释放对C的绑定。当使用回溯获得目标的备选解时,这种释放是必需的,但是这样做就难于把回溯用于任何其它的目的。其原因是,尽管回溯过程可以重复运行任意多次,但是它不能"记住"从一次重复到下一次重复的任何事情。当执行回溯经过建立变量值的步骤时,所有变量释放它们的值。还没有一种简单的方法为反复循环保持一个计数器、一个合计或任何其它的进展记录。

4　回溯过程分析

为了实现一个子目标,Visual Prolog从定义谓词的第一个子句开始搜索。这样会发生下列情况:

1) 找到一个匹配子句,发生下列三种情况:①如果有另一个可能再满足该子目标的子句,则Visual Prolog将放置一个指针(指示一个回溯点),指向下一个匹配的子句;②所有子目标中的自由变量与子句中的值相匹配,则被绑定为对应的值;③如果匹配子句是规则头,那么规则体接着被评估。为了使调用成功,子目标体必须成功。

2) 不能找到一个匹配子句,目标失败。Visual Prolog回溯,试图重新满足前一个子目标。当处理到达最后一个回溯点时,Visual Prolog释放所有回溯点后被赋新值的变量,然后试图重新满足原始调用。

Visual Prolog从程序的顶部开始搜索。当它回溯到一个调用时,新的搜索从所设置的最近一个回溯点开始。如果搜索不成功,则再次回溯。如果回溯用尽所有子目标的全部子句,则目标失败。

5　结语

回溯机制是逻辑程序设计的重要设施。回溯本身是一种获得目标所有可能解的良好方法。本文在考察Visual Prolog回溯机制工作原理的基础上,通过若干实例,详细阐述了回溯机制所遵循的4个基本原则,给出了回溯机制的循环实现方法,深入分析了回溯过程可能发生的各种情况,从而揭示出回溯机制的本质特性和应用机理。

然而,回溯也有副作用,在盲目搜索时,它可能导致Visual Prolog给出多余的答案,而Visual Prolog自己不能辨识实质上相同的两个解,因此需要利用截断、失败谓词等对Visual Prolog的搜索过程进行仔细地控制[2-3]。

参考文献:

[1] 雷英杰, 邢清华, 孙金萍, 等. Visual Prolog 智能集成开发环境评述[J]. 空军工程大学学报(自然科学版), 2002, 3(5): 39-43.

[2] 雷英杰, 张 雷, 邢清华,等. Visual Prolog 语言教程[M]. 西安:陕西科学技术出版社,2002.

[3] 雷英杰, 邢清华, 孙金萍,等. Visual Prolog 编程、环境及接口[M]. 北京:国防工业出版社,2004.

(编辑:田新华)

Analysis of the Backtracking Mechanism in Visual Prolog

LEI Ying-jie, WANG Tao, ZHAO Ye, WANG Jian-xun

(The Missile Institute, Air Force Engineering University, Sanyuan, Shaanxi 713800, China)

Abstract: The backtracking mechanism is an important facility for logic programming. The backtracking itself is a good approach to obtaining all the possible solutions to a goal. On the basis of observing and studying the operating elements of backtracking mechanism in Visual Prolog with a group of instances, the four fundamental principles, i.e. those of bottom - up, right - left sequencing, rule matching and fact matching, are expatiated in detail. A backtracking approach implemented by means of repetition is explored. All kinds of possible cases in the course of backtracking are deeply analyzed and investigated, thus the essential characteristics and the applied fundamentals of the backtracking mechanism are revealed. Finally, the conclusion indicates that the backtracking mechanism has side - effects, and it is necessary to control carefully the searching courses by using a cut mechanism, a fail predicate and otherwise.

Key words: Visual Prolog; logic programming; programming language; AI; expert system