笨办法学 Prolog

前言

> 原文出处：http://fengdidi.github.io/blog/2011/11/15/qian-yan/

我一直以来想写一部帮助想学Prolog的朋友学习Prolog的教程，因为我在学习Prolog的过程中，发现有关Prolog的教学文档很少很少，中文的文档更是几乎没有。这给我学习Prolog带来很大的困难，基本上都是在一边摸索一边学习的。所以我幻想着有一天能够写一篇Prolog的入门教程，来帮助其他想学Prolog的朋友对其有一个初步的了解。 这个想法在我心里存在了很久了，但是一直却没有付诸实施。其原因一是我没有太多的时间写这些文章，二是我没有一个如何写这部教程的点子。

直到台北小码农同学给我推荐了一个系列的教程叫《Learn XXX The Hard Way》，并且邀请我一起来仿照着写Prolog和Scheme的教程。我大致的阅读了一下《Learn Python The Hard Way》,发现那本书很适合对计算机了解不多，没有学过编程，但对编程感兴趣的朋友学习使用。那本书以习题的方式引导读者一步一步学习编程，从简单的打印一直讲到完整项目的实现。也许读完那本书并不意味着读者已经学会了编程，但至少读者会对编程语言以及编程这个行业有一个初步的了解。所以，我决定接受台北小码农的提议，仿照着这种格式来写一篇关于Prolog的入门教程。

这部教程假设读者有一定的英文基础，却不需要读者有半点的编程基础，反而，之前有过编程基础的朋友反而会发现学习起来有一定的吃力，因为Prolog的思考方式和其他的程序语言完全不同，所以你在写程序的时候需要时时刻刻地转换你的思维。

在学习这部教程的时候，你一定要记得一个道理，就是“万事开头难”。做事一定要不怕困难，要坚持，半途而废的话就永远不会成功。我本人就是一个喜欢半途而废的人，通常做一件事，每次都是想的比做的多，通常遇到一点儿困难以后就放弃了。所以我一直都没有成功。所以我在这里本着对我的读者负责的态度，一定要坚持把这部教程写完，这样我就总算是坚持做完了一件事。希望大家监督我哦～

既然我都开始坚持做完一件事了，优秀的你，一定要不管是什么原因，一定坚持下去。如果你碰到做不出来的加分习题，或者碰到一节看不懂的习题，你可以暂时跳过去，过一阵子回来再看。只要坚持下去，你总会弄懂的。

一开始你可能什么都看不懂。这会让你感觉很不舒服，就像学习人类的自然语言一样。你会发现很难记住一些单词和特殊符号的用法，而且会经常感到很迷茫，直到有一天，忽然一下子你会觉得豁然开朗，以前不明白的东西忽然就明白了。如果你坚持练习下去，坚持去上下求索，你最终会学会这些东西的。也许你不会成为一个编程大师，但你至少会明白程序是怎么工作的。

如果你通读了这部教程，却还是不知道编程是怎么回事。那也没关系，至少你尝试过了。你可以说你已经尽过力但成效不佳，但至少你尝试过了。这也是一件值得你骄傲的事情。

许可协议

第0章：为什么要学习Prolog

为什么要学习Prolog

当我在大学的课程表里面发现Prolog这门课的时候，我十分惊讶，我在想：为什么我要学习Prolog呢？我会使用Java, C++和php编程，这些语言已经强大到几乎可以实现任何功能，而且有很多很多写好的函数库来供你使用，为什么我还要学习什么Prolog？

当我开始学Prolog的时候，我发现这个语言的语法真是太奇怪了，有别于一般的编程语言，Prolog的程式是基于谓词逻辑的理论。最基本的写法是定立物件与物件之间的关系，之后可以用询问目标的方式来查询各种物件之间的关系。系统会自动进行匹配及回溯，找出所询问的答案。但是当我真正了解Prolog的时候，才发现，正因为Prolog的这种特性才让他异常的强大，简单的说，它是一个会自己思考的语言，它可以通过搜索自己的知识库来找到问题的答案，这是其他的程序语言所做不到的。在这里，我无法过多的解释Prolog的强大之处，我相信通过一段时间的学习，你可以逐渐发现Prolog是一门值得学习的语言。你可能在将来的软件开发事业中使用不到Prolog，但是你有关Prolog的知识能够帮助你更好的使用其他的语言。

Prolog的用途

目前来说，Prolog主要用在人工智能和计算机语言的研究领域。Prolog和LISP是两个主要的研究人工智能算法的工具，一个有趣的现象是：在美国，研究者们喜欢用LISP，在欧洲，研究者们更倾向与使用Prolog进行开发。这两门语言没有谁好谁坏之分，个人喜好罢了，事实上，在Prolog下面可以非常简单地实现一个LISP解析器，同样的，在LISP下也可以轻易的实现一个Prolog解析器。在后面的章节里面，我和台北小码农会教大家怎么做。

其次，得益于Prolog的模式匹配功能，Prolog非常适合快速的开发一个语言的解析器，这使得很多计算机科学家在开发新的程序语言时，喜欢用Prolog先写一个实现，然后观察大众的反应，如何大众认为这个语言很好，就用更快的语言如C++来重新写解释器，如果大众的反应不好，就再用Prolog进行修改。

第1章：配置开发环境

这道习题几乎没有代码内容，它的主要目的是让你在计算机上安装好Prolog。你应该尽量照着说明进行操作。

安装SWI-Prolog

MacOS

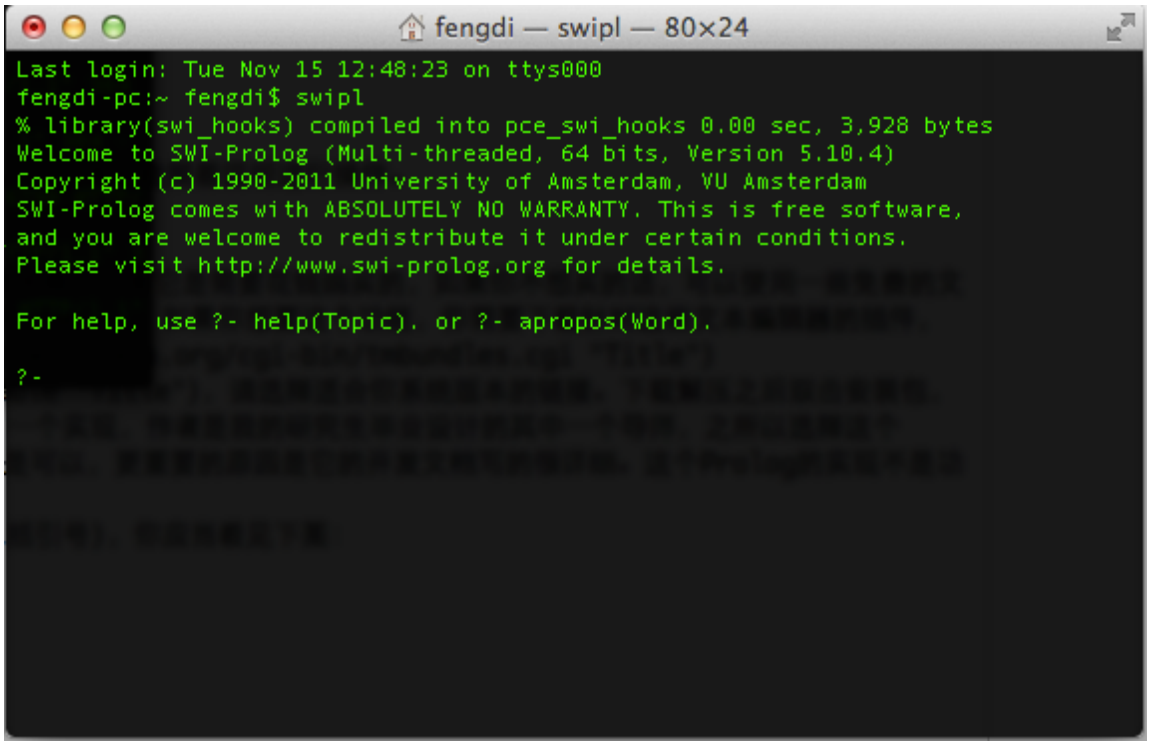
1. 找一个你最喜欢的文本编辑器。在Mac系统下，TextMate也许是最好的选择，但是它是需要花钱购买的，如果你不想买的话，可以使用一些免费的文本编辑器比如Kod。需要注意的是，这写编辑器本身都是不支持Prolog代码高亮的，如果你想要这个功能，你需要下载针对这些文本编辑器的插件，其中TextMate的插件可以在这里下载到TextMate Bundle

2. 下载SWI-Prolog，请选择适合你系统版本的链接。下载解压之后双击安装包，等待一段时间以后，你的Prolog就安装好了。SWI-Prolog是Prolog的一个实现，作者是来自阿姆斯特丹大学的Jan，之所以选择这个Prolog实现作为开发的环境，一个原因是因为它很稳定，运行速度也算是可以，更重要的原因是它的开发文档写的很详细。这个Prolog的实现不是功能最多的，但是我个人认为是最好用的，也是最适合Prolog的初学者使用。

3. 当你安装好Prolog以后，进入命令终端，输入：

    ```
    swipl
    ```

    你应当看见下图：



Windows

1. 第一步同样是找一个自己喜欢的文本编辑器，个人推荐Notepad++，你可以轻易的在Google上搜寻到下载地址。
2. 下载SWI-Prolog，选择Windows的安装包，下载解压之后双击安装包，等待一段时间以后，你的Prolog就安装好了。
3. 与MacOS不同的是，在Windows下，你可以不必去命令行下面输入”swipl”,你可以直接双击桌面上的快捷方式就可以打开SWI-Prolog了。打开以后的界面应该和MacOS下的界面类似。

Linux

我相信使用Linux系统的朋友应该都懂得如何安装一个小小的软件吧？所以在这里就不赘述了~

Hello World!

好像在大部分的程序语言的时候，第一个要编写的程序都是“Hello World!”。虽然“Hello World”程序不能显示出Prolog的特性，我在这里也姑且做一个“Hello World!”的程序吧，目的是让大家试一下你们刚才下载的SWI-Prolog是否工作。

按照之前的方法进入SWI-Prolog，在命令行下输入：

```
writeln('Hello World!').
```

需要注意的是，这行代码一定要以英文中的句号”.”来结尾，Prolog中的“.”和C语言中的“;”一样，都是代表一段代码的结尾。再者，Hello World!字符串一定要以单引号来包裹。 如果输入正确的话，你将看到如下输出：

```
Hello World!
true.
```

这里的“Hello World!”很好理解，这是我们要求程序输出的，那么那个奇怪的“true”是哪里来的呢？请注意，在Prolog终端输入的时候，没一个语句都是以“?-”这样两个字符开头的，它代表我们输入的程序代码其实是对Prolog系统的一个查询（问询），一旦用户输入了查询，Prolog系统会运用它的知识库来判定这个查询是真(true)是假(false). writeln是Prolog系统自己定义的一个语句, 它的作用是向当前的显示设备输出一个字符串并且换行, 所以很显然, 这个语句是真的, 因为Prolog知道有这个语句. 这就是为什么程序的最后有一个”true”. 有意思的是,因为整个过程中Prolog都是在试图证明这个语句是真是假, 向屏幕输出”Hello World!”这件事实际上是执行这个语句的”副作用”(side effect)!在Prolog中, 很多任务都是靠副作用来实现的, 包括输入输出, 甚至是参数的传递.

最后，如果想要退出SWI-Prolog，输入：

```
halt.
```

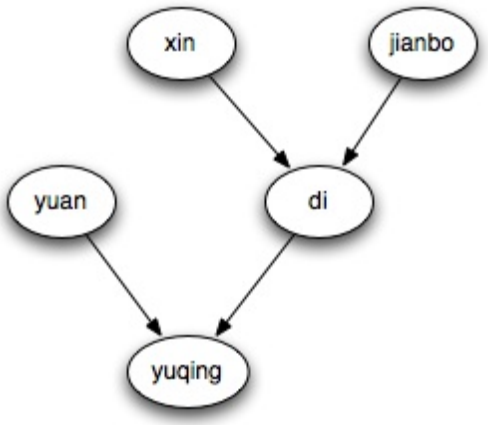
同样，不要忘记最后的“.” ~

好了, 到这里, 这一章就算是结束, 因为这一章讲的内容很基本, 我就不提供习题了. 下一章我们将正式开始学习有关Prolog语言的知识! 敬请期待!

第2章：谁是谁的爸爸

家谱

假设我们有这样一个家谱图：



我们现在的任务是将这个家谱图写成程序代码的形式。请打开你最喜欢的文本编辑器，输入以下代码。

```
male(di).
male(jianbo).
female(xin).
female(yuan).
female(yuqing).
father(jianbo,di).
father(di,yuqing).
mother(xin,di).
mother(yuan,yuqing).
grandfather(X,Y):-father(X,Z),father(Z,Y).
grandmother(X,Y):-mother(X,Z),father(Z,Y).
daughter(X,Y):-father(X,Y),female(Y).
```

这段代码里面的每一行都代表一个子句(clause)。其中带有“:-”的子句叫做规则(rule)，不带有”:-“的子句叫做事实(fact)。另外，在Prolog里面诸如”di”和”jianbo”这类以小写英文字母开头的名称我们称它们为原子(atom)，以大写英文字母为开头的名称我们称它们为变量，例如上面程序里面的”X”和”Y”。顾名思义，原子是常量，即它的值是不可变的，而变量的值可以改变。最后需要讲的是，在Prolog里面”,”代表逻辑关系中的”且”，我们回在后面的章节里面看到，”;”代表逻辑关系里面的”或”。

已经被这些名称搞得头晕了？没关系，我会在之后的教程里面详细的介绍Prolog的数据类型和术语，在这里，你只需有初步的了解即可。

保存上述代码到你的磁盘的某个地方，例如在Mac系统里，我把它存到“~/prolog/chapter2.pl”,然后依照第一章里面讲的那样，进入SWI-Prolog。在SWI-Prolog里面输入如下查询：

```
?- consult('path/to/your/chapter2.pl').
```

在我的电脑里，我应该这么输入：

```
?- consult('~/prolog/chapter2.pl').
```

这里”consult”的意思是让SWI-Prolog加载你编写的程序，然后编译它。输入完这句查询以后，敲击回车键，你应该得到如下输出：

```
% /Users/fengdi/prolog/chapter2.pl compiled 0.00 sec, 3,816 bytes
true.
```

如果你得到了上述的输出，那么恭喜你，你的第一个程序完成了。如果你得到的是其他的错误的输出，请重新检查你的程序代码是否输入正确(不过要记得，千万不要因为想要保证代码输入的不出错而直接复制粘贴代码，那样的话你学不到真正的东西)。下面，让我们考验一下我们的SWI-Prolog现在都知道些什么。在SWI-Prolog里面输入下面一个查询：

```
grandfather(X,yuqing).
```

令人惊讶的事情发生了！你得到了下列输出：

```
X = jianbo.
```

你的电脑告诉你，” yuqing”的祖父是”jianbo”。现在请看之前我们编写的”chapter2.pl”程序代码，我们在程序里根本没有明确的说明谁是谁的祖父，我们只是给了一个规则：

```
grandfather(X,Y):-father(X,Z),father(Z,Y).
```

我们说，当X是Z的父亲并且Z是Y的父亲的时候，X是Y的祖父。然后我们问，yuqing的祖父是谁，Prolog就能自动帮我们找到答案！

下面再看一个例子：

```
parent(keyuan,jianbo).
parent(jianbo,di).
parent(di,yuqing).

ancestor(X,Y):-parent(X,Y).
ancestor(X,Y):-parent(X,Z),ancestor(Z,Y).
```

在这个例子里面，我们定义了一个回溯的规则” ancestor” ，规则可以这样解读：我们可以说X是Y的祖先基于两个条件：X是Y的parent，或者存在一个Z，使得X是Z的parent并且Z是Y的祖先。请读者仔细的想一

下，是不是这个道理呢？为了证明我们的程序的正确性，我们输入一下查询：

```
?- ancestor(keyuan,yuqing).
```

Prolog会返回：

```
true.
```

这证明了我们的程序是正确的，因为根据常识，keyuan是yuqing的曾祖父，所以keyuan是yuqing的祖先。

好了，今天的新内容就讲到这里，下面是一个习题，你可以自己试验一下。

加分习题

1. 试着用Prolog描述一下你的家谱，并且做一些简单的查询。(小提示：在编写你的家谱的时候，你可以试着用一些新的事实，比如:”sister(your_sister,you)” “brother(your_brother,you)”等等）

第3章：Prolog是如何回答问题的

在上一章节里面，我给大家演示了一个Prolog特有的神奇的功能：它能够回答你提出的问题！在这一章里面，我将简单的解释一下Prolog是如何能够回答你的问题的。 首先，还是自己试着把下面的程序写一下，然后加载到SWI-Prolog里面。

```
parent(di,yuqing).
parent(keyuan,jianbo).
parent(jianbo,di).

ancestor(X,Y):-parent(X,Y).
ancestor(X,Y):-parent(X,Z),ancestor(Z,Y).
```

然后我们问Prolog:

```
?- ancestor(keyuan,yuqing).
```

Prolog系统会试图证明这个查询，很显然，程序会返回”true.”那么，Prolog系统是如何找到答案的呢？当我们向Prolog提问”ancestor(keyuan,yuqing)”的时候，Prolog首先会在它的知识库里面寻找”ancestor”的定义，当然，它找到了两个”ancestor”的定义：

```
ancestor(X,Y):-parent(X,Y).
ancestor(X,Y):-parent(X,Z),ancestor(Z,Y).
```

按照默认的规定，它会首先取第一个规则：“ancestor(X,Y):-parent（X,Y).”由于这个规则里面的X,Y是变量，Prolog会给这两个变量赋值，使得：

```
ancestor(keyuan,yuqing) = ancestor(X,Y)
```

很显然，X=keyuan,Y=yuqing。之后，Prolog会把ancestor换成后面的条件：

```
ancestor(keyuan,yuqing) ==> parent(keyuan,yuqing).
```

之后，Prolog试图证明”parent(keyuan,yuqing)”,根据上面的程序，很显然，这是错的，会返回：” false” 。那么，之后Prolog会怎么办呢？记得上面说过，ancestor有两个规则吧，我们之前根据惯例选择了第一个规则，现在我们可以试一下第二个规则：” ancestor(X,Y):-parent(X,Z),ancestor(Z,Y).”根据这个规则，Prolog会把”ancestor(keyuan,yuqing).”换成了：

```
parent(keyuan,Z),ancestor(Z,yuqing).
```

之前Prolog需要证明一个式子(ancestor(keyuan,yuqing).)，现在Prolog需要证明两个式子了。因为这两个式子中间是一个逗号，逗号的意思是”且 “，所以只有当这两个式子都是”true”时候，整个的查询才是”true”。首先，Prolog会尝试第一个”parent(keyuan,Z).“它会在知识库里面找到：” parent(keyuan,jianbo)”。于是，Z=jianbo。这时候，我们之前的那两个式子变成了：

```
parent(keyuan,jianbo),ancestor(jianbo,yuqing).
```

然后，Prolog尝试证明第二个式子：” ancestor(jianbo,yuqing)”。同样的道理，Prolog首先找到的是：” ancestor(X,Y):-parent(X,Y)” ，于是想证明”ancestor(jianbo,yuqing)”就变成了要证明”parent(jianbo,yuqing)”。很显然”parent(jianbo,yuqing)”会返回”false”。这时候，程序会试着把”ancestor(jianbo,yuqing)”替换成”parent(jianbo,Z1),ancestor(Z1,yuqing)”。要注意的是，此处我们不用Z作为变量了，而是用Z1作为变量，这叫做变量的重命名。原因是我们之前已经用过Z做变量了，为了把现在这个Z变量和之前的Z变量区别开来，我们把现在的Z变量重命名为Z1。很显然，无论变量的名字如何，它都是一个变量，理论上我们可以把它赋值成任何值，所以修改名字不会改变变量的含义。根据我们的知识库，此时Z1应该等于di。于是我们得到了两个式子：

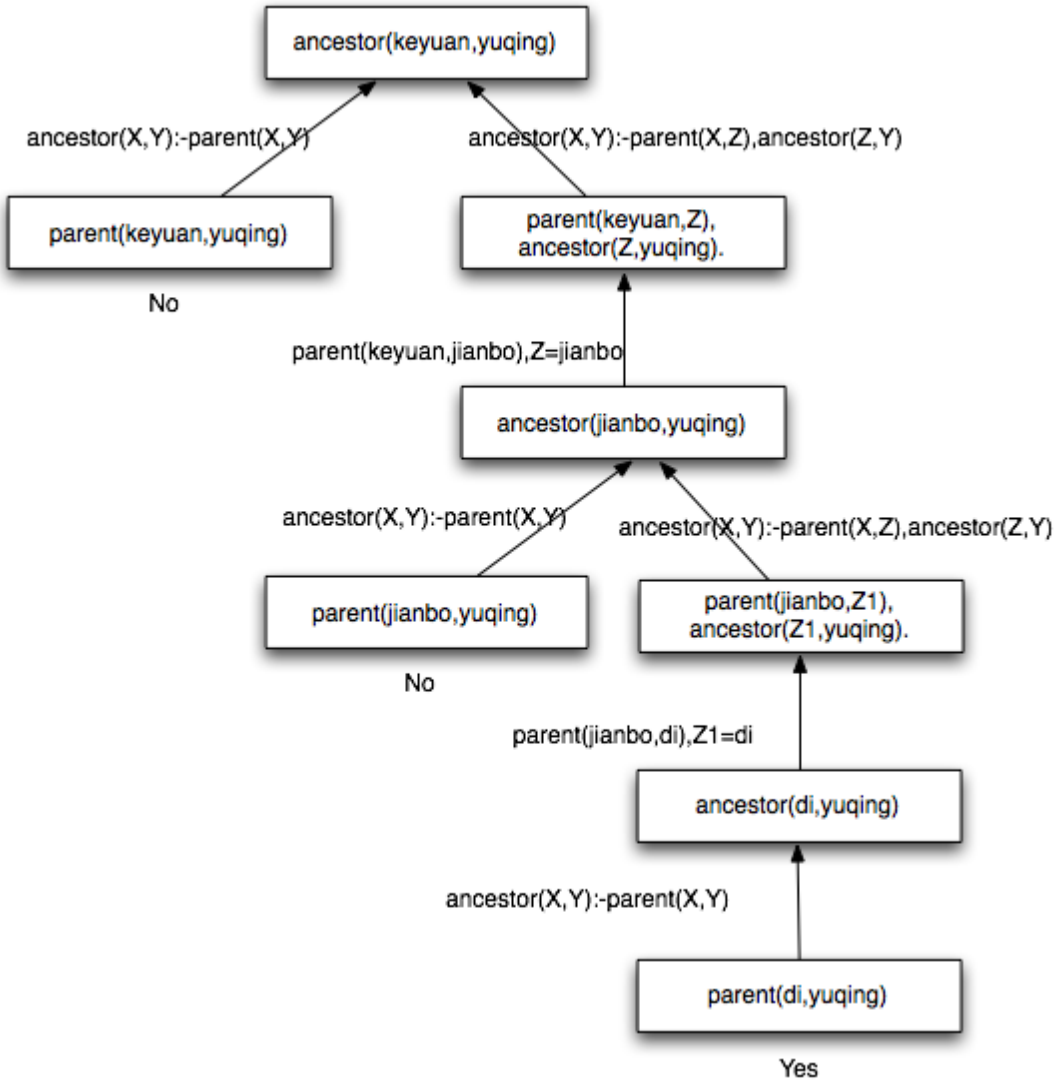
```
parent(jianbo,di),ancestor(di,yuqing).
```

把”ancestor(di,yuqing).”换成”parent(di,yuqing)” ，我们得到一个事实(fact),所以”parent(di,yuqing)”是真。 综上所述，我们其实是用回溯的方式把”ancestor(keyuan,yuqing).”换成了：

```
parent(keyuan,jianbo),parent(jianbo,di),parent(di,yuqing).
```

由于上面的三个式子都是真的，所以”ancestor(keyuan,yuqing).”是真的。

下面这个图显示了Prolog是如何证明你的查询的：



通过这个简单的例子，我们可以得出，Prolog通过三个方面来尝试着证明你给的查询(query)：

- 匹配(unifing/matching)。例如上面的”ancestor(keyuan,yuqing) = ancestor(X,Y)”，Prolog试图从它的知识库里面找出最符合你的查询的规则或者是事实。
- 变量重命名(variable renaming)。例如上面的”parent(jianbo,Z1),ancestor(Z1,yuqing)”，因为在同一个查询(query)里面已经有了”Z”，所以Prolog系统将重复的Z命名为Z1。(事实上，在Prolog内部，变量根本不会用”Z”,”Z1”这样的名字，Prolog的编译程序的时候就已经将”Z”换成了”G132329392049”这类名字以保证名字不会重复)。
- 回溯(back-tracking)。这是Prolog最最重要的一个特性。Prolog是用深度优先(depth-first search)的算法来寻找答案的。当一个规则或者是事实不符合时，Prolog会通过回溯的方式回到之前的状态，然后去尝试另外的规则或者是事实，知道你的查询(query)被证明为止。如果所有的可能性都搜索过了，你的查询仍然不能得到证实，那么Prolog会认为你的查询证实不了，返回”false”。有关回溯算法的详细介绍你可以去网络上搜索一下。

到这里，这一章就结束了，希望你对Prolog程序的执行顺序有了一个初步的了解。说实话，Prolog的程序回溯执行方式有时候我自己都想不明白，特别是遇到特别复杂的问题的时候。这个时候，最好的办法就是拿一张纸一支笔，亲自画一下程序的回溯图，就想上面的那个图一样，这样就能帮助你理解程序了~

加分习题

1. 我们有如下代码：

   parent(tom,bob). parent(pam,bob). parent(pam,liz). parent(pam,bob). male(tom). male(bob). female(liz). female(pam). sister(X,Y):-

   ```
   parent(Z,X),
   parent(Z,Y),
   female(X),
   X\=Y.
   ```

   请比照着上面的教程里面”ancestor(keyuan,yuqing)”的执行顺序图画一个 ?-sister(liz,bob). 的执行图。(注：程序中的”X\=Y”是X不等于Y的意思)

2. (这道题有点难度哦)请看下图：

L1	L2	L3	L4	L5	
L6		L7		L8	
L9	L10	L11	L12	L13	L14
L15				L16	

我们的目的是要在上面表格中白色的方格(带有LXX标识的方格)里面填上英文单词，可供选择的单词有：

```
word(d,o,g).   word(r,u,n).   word(t,o,p).   word(f,i,v,e).
word(f,o,u,r). word(l,o,s,t). word(m,e,s,s). word(u,n,i,t).
word(b,a,k,e,r).   word(f,o,r,u,m).   word(g,r,e,e,n).
word(s,u,p,e,r).   word(p,r,o,l,o,g). word(v,a,n,i,s,h).
word(w,o,n,d,e,r). word(y,e,l,l,o,w).
```

试着写出一个规则solution.

```
solution(L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12,L13,L14,L15,L16).
```

第4章：Prolog程序的两种意义

单词谜题

在进行下一章之前，我想先把上一章加分习题的第二题答案给大家揭晓一下，因为台北小码农给我反应说这一道题有点难了，连学过Prolog的人都不一定会做。确实，这是一道难题，但是这道题的难点不在于程序本身逻辑上的复杂或者是用到了什么高级的语法，这道题的难点在于你必须要清楚地理解Prolog程序的意义才能得出答案。好了，先给大家看一下答案的程序：

```
solution(L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12,L13,L14,L15,L16):-
    word(L1,L2,L3,L4,L5),
    word(L9,L10,L11,L12,L13,L14),
    word(L1,L6,L9,L15),
    word(L3,L7,L11),
    word(L5,L8,L13,L16).
```

以上就是这个单词谜题的答案。在编写Prolog程序的时候，之前有编程基础的朋友一定要摒弃之前编程的习惯，因为像C,Java这样的程序语言，程序员解决问题的方式实际上是把解决问题的步骤用程序一步一步地表示出来。换句话说，就是程序员要用程序语言告诉电脑应当怎样一步一步地做，才能找到问题的答案。Prolog不是这样，Prolog程序员让程序解决一个问题时，只需要把想要得到的答案的形式表述出来，Prolog系统会自动帮你找答案。就像上面这个程序一样，你告诉Prolog系统，答案应该是什么样子的呢？首先，我们要将5个单词填入表格中，同时，单词和单词之间有可能共享同一个英文字母。所以这儿有两个“约束”，一个约束是单词字母的个数，第二个约束是单词里面可以出现的字字母。让我们先写水平排列的两个单词：

```
word(L1,L2,L3,L4,L5), <-----单词1
word(L9,L10,L11,L12,L13,L14)    <-----单词2
```

之后，我们加上竖直的三个单词：

```
word(L1,L6,L9,L15), <-----单词3
word(L3,L7,L11),    <-----单词4
word(L5,L8,L13,L16) <-----单词5
```

单词1和单词3共享字母“L1”,单词4和单词1共享字母“L3”,单词4和单词2共享字母“L12”，依次类推，我们可以把所有的约束都写出来。这样，我们就告诉Prolog系统最后的答案长得什么样儿了。有了答案的形式，Prolog就会用我之前提到的三个策略(回溯，变量重命名，模式匹配)来找出答案。

程序的意义

上述的程序其实昭示了Prolog程序的其中一个意义：陈述性意义(declarative meaning)。Prolog程序具有

陈述性意义就在于Prolog都是描述的逻辑学上的一个事实或者是一个规则。例如：

```
mother(pam,bob).
```

这句程序语句陈述了一个事实：pam是bob的妈妈。再看：

```
grandmother(X,Y):-mother(X,Z),father(Z,Y).
```

这个语句陈述的是一个规则：当X是Z的母亲并且Z是Y的父亲时，X是Y的母亲。当我们在Prolog系统上输入“grandmother(X,bob)”这样的查询时，实际上就是让Prolog根据自己知道的逻辑规则来试图证明你的查询。这就是程序的陈述性意义。

程序的第二个意义是过程性意义。想Java和C这样的语言一样，Prolog的程序也具有过程性意义，即解决问题的步骤。在举例子之前，我先讲一个在Prolog里面常用的语句：” write” ，它的作用是向显示屏输出信息。它的参数可以是任意类型的数据，数字，字符串，甚至一个Prolog的规则都可以直接被输出。另外一个语句是” nl” ，它的作用是使Prolog的屏幕输出换行。下面给一个程序，你就能理解” write” 和” nl” 的作用了：

```
write('I have a dream'),nl,
write('that my four little children will one day live in a nation'),nl,
write('where they will not be judged by the color of their skin'),nl,
write('but by the content of their character.'),nl,
write('This is form Martin Luther King').
```

Prolog会这样输出：

```
I have a dream
that my four little children will one day live in a nation
where they will not be judged by the color of their skin
but by the content of their character.
This is form Martin Luther King
true.
```

上面这个程序其实就揭示了Prolog程序的过程性意义，它告诉Prolog系统：要想输出上面这一段话，首先要输出’I have a dream’ ，然后换行，接着输出’that my four little children will one day live in a nation ‘，然后换行，依次类推。它实际上是告诉了Prolog系统一个解决问题的步骤。这就是程序的过程性意义。

其实，就Prolog本身的意义来说，编写Prolog程序的时候是不提倡加入太多的过程性意义的，尽量要以描述性意义为主，因为Prolog本身就是一个描述性语言。但是，程序员们在实际开发的时候发现，他们离不开过程性意义的程序，因为现实生活中解决问题本身就是需要步骤的，例如你想从北京到泰安，你要先从北京到天津，然后从天津到沧州，然后从沧州到济南，最后从济南到泰安。所以，在Prolog里面就不可避免的出现过程性意义为主的程序。

加分习题

1. 下面我们来试着把单词谜题的答案打印的屏幕上，你要写一个规则” print_solution “，试着把答案这样打印：

```
word1 is XXXXXX
word2 is XXXXXX
word3 is XXXXXX
word4 is XXXXXX
word5 is XXXXXX
```

   之后，试着解释一下你写的程序的过程性意义和陈述性意义。