

公告

昵称: seaman.kingfall
园龄: 4年3个月
粉丝: 4
关注: 1
[+加关注](#)

搜索

常用链接

[我的随笔](#)
[我的评论](#)
[我的参与](#)
[最新评论](#)
[我的标签](#)

我的标签

练习题(6)
合一(3)
递归(3)
中断(2)
类型变量(2)
数字(2)
列表(2)
Haskell(2)
recursive(2)
比较(2)
[更多](#)

随笔分类

Haskell(2)
Prolog(32)

随笔档案

2015年8月 (7)
2015年7月 (22)
2015年6月 (5)

最新评论

1. Re:Learn Prolog Now 翻译 - 第一章 - 事实，规则和查询 - 第一节，一些简单的例子学习！
--深蓝医生
2. Re:Learn Prolog Now 翻译 - 第一章 - 事实，规则和查询 - 第一节，一些简单的例子翻译了这么多了，而且每天一篇，不能望其项背啊。
--Benjamin Yan

阅读排行榜


1. Learn Prolog Now 翻译 - 第三章 - 递归 - 第一节，递归的定义(1168)
2. Learn Prolog Now 翻译 - 第一章 - 事实，规则和查询 - 第一节，一些简单的例子(1087)
3. Learn Prolog Now 翻译 - 第一章 - 事实，规则和查询 - 第二节，Prolog语法介绍(781)
4. Haskell学习笔记二：自定义类型(767)

Learn Prolog Now 翻译 - 第二章 - 合一和证明搜索 - 第二节， 证明搜索


证明搜索

上一节我们已经学习了合一，本节我们继续学习Prolog是如何通过搜索知识库去决定输入的查询是否能够满足。我们将会学习证明搜索，并通过简单的一个例子去涵盖这个基础的概念。

假设我们有如下的知识库：



```
f(a).  
f(b).  
  
g(a).  
g(b).  
  
h(b).  
  
k(X) :- f(X), g(X), h(X).
```



如果我们查询：

`?- k(Y).`

这个查询的结果十分明显，即`k(b)`，但是Prolog是如何将其求解出的了？让我们继续看。

Prolog读入整个知识库，然后尝试将查询`k(Y)`和知识库中的事实或者规则头部进行合一。搜索知识库是自上而下的，并在第一个可能合一的位置，找到对应的信息。这个例子只是有一种可能：`k(Y)`和规则，`k(X) :- f(X), g(X), h(X)` 的头部合一。

当Prolog将查询中的变量和事实或者规则中的变量合一时，会生成一个新的变量（比如`_G34`）去代表共享值，所以原始的查询转换为：

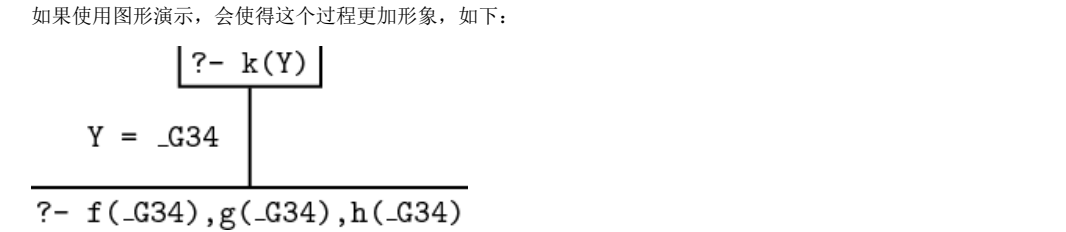
`k(_G34)`

并且Prolog知识库中的规则转换为：

`k(_G34) :- f(_G34), g(_G34), h(_G34).`

现在是什么情况了？查询说：“我想要找到符合属性`k`的人”。规则说：“如果要找到符合属性`k`的人，那么这个人也必须符合属性`f`，`g`和`h`”。所以如果Prolog找到符合属性`k`，`g`和`h`的信息，那么就能满足原始的查询。所以Prolog使用下面的目标去替代原始查询：

`f(_G34), g(_G34), h(_G34).`



在盒子中的都是查询或者目标。具体地讲，我们的原始目标是证明`k(Y)`，所以它出现在最顶层的盒子中。当我们将`k(Y)`和数据库中的规则头部合一时，`_G34`作为分享值的新中间变量，被赋值给`X`，`Y`，所以我们有了新的目标：`f(_G34), g(_G34), h(_G34)`，出现在第二个盒子中。

现在，无论是否存在一系列的子目标需要证明，Prolog都会通过自左向右的顺序，尝试一个一个的去满足。在最左侧的目标是`f(_G34)`，即“想找到一个满足属性`f`的信息”。Prolog尝试搜索自上而下搜索知识库。第一个找到能够和查询合一的是事实，`f(a)`。这会满足目标`f(_G34)`，并且剩余另外两个目标。现在，当我们将`f(_G34)`和`f(a)`合一，`_G34`会被初始化为`a`，而且这个初始化会将目标列表中所有的`_G34`都替换为`a`，所以剩余的目标列表看上去是这样的：

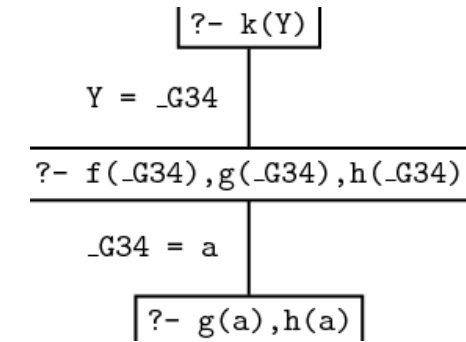
评论排行榜

- 1. Learn Prolog Now 翻译
- 第一章 - 事实, 规则和查询
- 第一节, 一些简单的例子
(2)

推荐排行榜

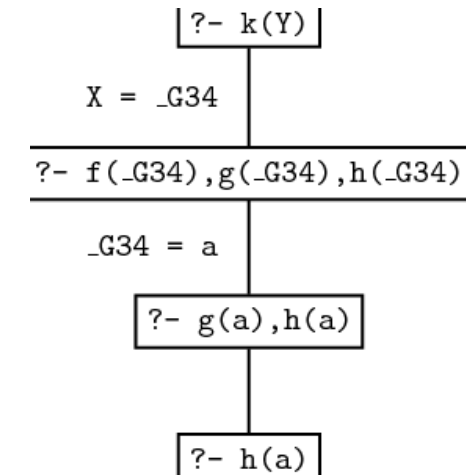
- 1. Haskell学习笔记二: 自定义类型(1)
- 2. Learn Prolog Now 翻译
- 第三章 - 递归 - 第四节, 更多的实践和练习(1)

$g(a), h(a)$
当前的证明搜索图形如下:



但是 $g(a)$ 是知识库中的事实, 所以剩余目标列表中的第一个目标已经满足了, 所以我们还剩下:

$h(a)$
证明搜索的图形如下:



但是无法再证明 $h(a)$, 最后一个目标。因为关于属性 h , 我们通过知识库能够知道的唯一事实是, $h(b)$, 它无法和目标 $h(a)$ 合一。

所以, 接下来会发生什么? Prolog会承认犯了错误, 并且检查是否在对目标进行合一时, 有其他可能的值。通过上面图形展示的路径, Prolog会回到可以有合一替代的节点。

现在, 在知识库中没有其他可以和 $g(a)$ 合一的选择了, 但是存在其他与 $f(_G34)$ 合一的方式。存在多种合一可能的节点被称为选择节点。Prolog会保持对选择节点的追踪, 所以

当它发现一种选择是错误的时候, 能够回到选择节点并且尝试其他的路径。这个过程被称为回溯, 它是Prolog证明搜索的基础。

那么继续我们的例子, Prolog回溯到选择节点。这个节点在上面图形中是下面的目标列表:

$f(_G34), g(_G34), h(_G34)$.

Prolog必须重新开始工作, 他必须尝试重新满足第一个目标。在知识库中, 通过将事实 $f(b)$ 和 $f(_G34)$ 合一, 可以满足目标。这个合一会将 $_G34$ 初始化为 b , 所以剩余的目标列表是:

$g(b), h(b)$.

$g(b)$ 也是知识库中事实, 也能够被满足, 所以剩余:
 $h(b)$.

而且, 这个也是知识库中的事实, 所以这个目标也满足了。现在Prolog的目标列表已经为空。这意味着原始查询所需要的每一个目的都满足了, 所以原始查询是能够成功的, 并且,

Prolog也发现了为了达成目标而并且进行的初始化, 即将变量 Y 初始化为 b 。

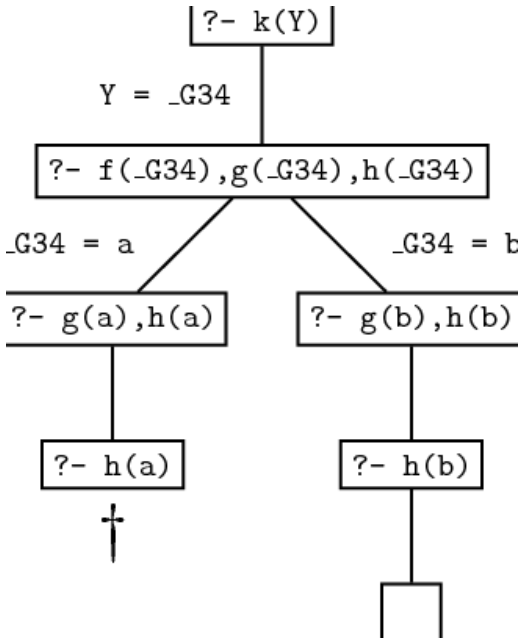
考虑当我们输入 “;” 查询是否有其他解决方案也很有趣:

这会强制Prolog进行回溯, 去搜索是否有其他可能。但是上面的例子没有其他解决方案了, 因为知识库中没有其他可能再去将 $h(b), g(b), f(_G34)$ 或者 $k(Y)$ 进行合一, 所以Prolog

会回答false。另一方面，如果存在另外包含了k的规则，Prolog会按照我们之前描述的方式继续尝试，即在知识库中自上而下的搜索，从左自右地满足目标列表，如果有失败就回溯到

选择节点。

让我们看看整体的搜索过程，如下图：



这个图形是树的形式，事实上，这是我们第一个关于搜索树的例子。这种树的非叶子节点是为了满足证明搜索不同步骤的当前目标列表，树的边保存了为了满足当前目标而对知识库

中的事实或者规则进行合一的变量初始化信息（注意，当前目标是指目标列表的第一个即最左边的目标），叶子如果还包含没有被满足的目标，那么就是Prolog失败的点（错误的路径，

没有解决方案存在）；叶子如果不包含任何的目标，就是一种可能的解决方案。从根节点到成功的叶子节点的路径，会给出为了满足原始目标而必须进行的变量初始化的值的全部信息。

接下来我们看另外一个例子，假设我们有如下的知识库：

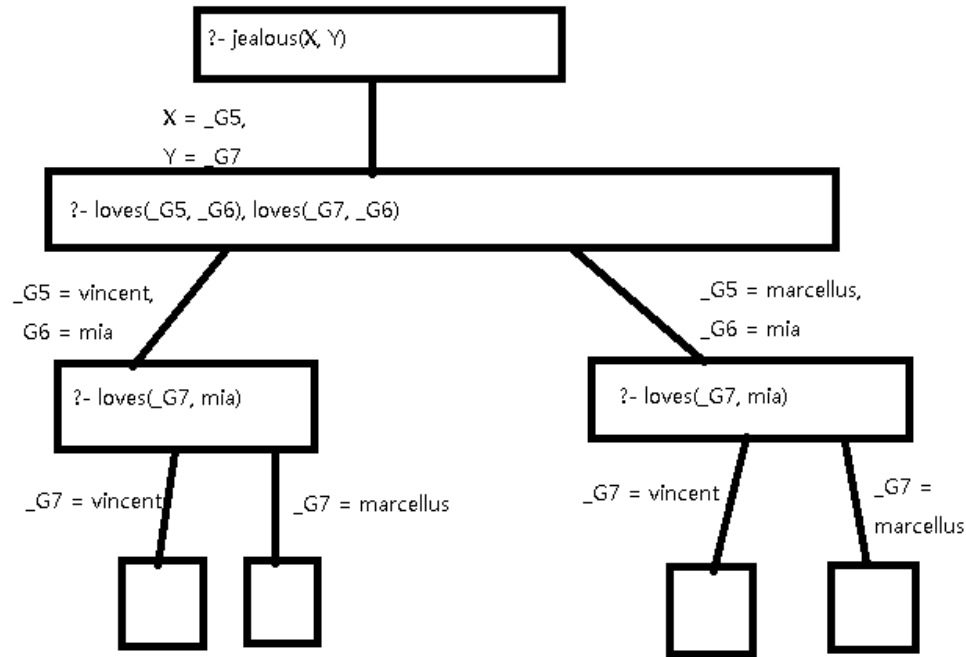
```
loves(vincent, mia).
loves(marcellus, mia).

jealous(A, B) :- loves(A, C), loves(B, C).
```

如果我们查询：

?- jealous(X, Y).

对应的搜索树如下：



在知识库中只有一种针对 `jealous(X, Y)` 的合一方式，即使用如下的规则：

```
jealous(A, B) :- loves(A, C), loves(B, C).
```

所以我们的目标列表变成了如下：

```
loves(_G5, _G6), loves(_G7, _G6).
```

现在，我们需要在知识库中针对 `loves(_G5, _G6)` 进行合一。有两种方式可以做到（事实1和事实2），所以这也是一个选择节点。这两种情况，都是导致目标列表剩余 `loves(_G7, mia)`，

这个目标也可以通过知识库中的两个事实满足。最后所有的叶子节点都没有不能满足的目标，所以意味着一种有四种满足原始查询的解决方案。通过路径上的变量初始化信息，可以得出

如下的四种解决方案：

1. `X = _G5 = vincent; Y = _G7 = vincent`
2. `X = _G5 = vincent; Y = _G7 = marcellus`
3. `X = _G5 = marcellus; Y = _G7 = marcellus`
4. `X = _G5 = marcellus; Y = _G7 = vincent`

请仔细分析上面的例子，并确保能够完全理解。

分类: Prolog

标签: 证明搜索, proof search, 合一

好文要顶

关注我

收藏该文

seaman.kingfall

关注 - 1

粉丝 - 4

+加关注

0 0

« 上一篇: Learn Prolog Now 翻译 - 第二章 - 合一和证明搜索 - 第一节， 合一

» 下一篇: Learn Prolog Now 翻译 - 第二章 - 合一和证明搜索 - 第三节， 练习题和答案

posted on 2015-07-02 13:10 seaman.kingfall 阅读(471) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

- 注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。
- 【推荐】超50万C++/C#源码：大型实时仿真组态图形源码
- 【活动】看雪2019安全开发者峰会，共话安全领域焦点
- 【培训】Java程序员年薪40W，他1年走了别人5年的路

相关博文:

- 原因和证明
- Learn Prolog Now 翻译 - 第五章 - 数字运算 - 第二节, 数字运算与列表
- euler证明
- Learn Prolog Now 翻译 - 第一章 - 事实, 规则和查询 - 第二节, Prolog语法介绍
- 数学证明和科学证明

最新新闻:

- 知否 | 太空垃圾如何清理? 卫星测试用鱼叉击中太空垃圾碎片
 - 一线 | “美团配送”品牌发布: 对外开放配送平台 共享配送能力
 - 苍蝇落在食物上会发生什么? 让我们说的仔细一点
 - 科学家研究板块构造变化对海洋含氧量影响
 - 日本程序员节假日全员加班? 都是“令和”惹的祸
- » 更多新闻...

Copyright @ seaman.kingfall
Powered by: .Text and ASP.NET
Theme by: .NET Monster