Learn Prolog Now 翻译 - 第六章 - 列表补遗 - 第一节，列表合并

内容提要：

列表合并的定义

列表合并的使用

列表合并的定义

我们将会定义一个很重要的谓词：append/3，其中所有的参数都是列表。从声明性角度去看，append(L1, L2, L3)的含义是列表L3是列表L1和列表L2的合并结果（合并意味着连接）。比如，

如果我们查询：

```
?- append([a, b, c], [1, 2, 3], [a, b, c, 1, 2, 3]).
```

或者查询：

```
?- append([a, [foo, gibble], c], [1, 2, [[], b]], [a, [foo, gibble], c, 1, 2, [[], b]]).
```

Prolog会回答true。

但是，如果我们查询：

```
?- append([a, b, c], [1, 2, 3], [a, b, c, 1, 2]).
```

或者查询：

```
?- append([a, b, c], [1, 2, 3], [1, 2, 3, a, b, c]).
```

Prolog会回答false。

从程序性的角度来说，append/3最为有用的作用是连接两个列表。我们可以在第三个参数中使用变量，轻松地达到目的：

```
?- append([a, b, c], [1, 2, 3], L3).

L3 = [a, b, c, 1, 2, 3].
```

但是（我们将会看到）我们也可以使用append/3分割列表。事实上，append/3是一个真正多用途的谓词，我们可以通过它做很多事情，并且通过学习这个谓词，我们可以更好地理解在

Prolog中如何操作列表。

如下是append/3的定义：

```
append([], L, L).

append([H|T1], L2, [H|T3]) :- append(T1, L2, T3).
```

这是一个递归形式的定义。基础子句很简单：将一个空列表和任意列表进行合并，那么结果是和任何列表相同的列表，这显然是正确的。

那么递归的部分呢？含义是：如果我们将一个非空列表，[H|T]和另外一个列表L2进行合并，那么得到的列表是头部为H，并且尾部是T和L2合并的结果。图示可能会更清楚：

Input: [H | T] + L2

Result: [H | T + L2]

但是这个定义的程序性含义是什么？当两个列表发生合并时实际会如何操作？让我们根据细节的分解来进一步学习，例子还是查询：

?- append([a, b, c], [1, 2, 3], X).

当我们进行这个查询时，Prolog将会使用递归子句去进行匹配，并且生成一个新的中间变量（比如称为_G518），如果我们跟踪接下来的步骤，可能如下：

append([a, b, c], [1, 2, 3], _G518).

append([b, c], [1, 2, 3], _G587).

append([c], [1, 2, 3], _G590).

append([], [1, 2, 3], _G593).

append([], [1, 2, 3], [1, 2, 3]).

append([c], [1, 2, 3], [c, 1, 2 ,3]).

append([b, c], [1, 2, 3], [b, c, 1, 2, 3]).

append([a, ,b, c], [1, 2, 3], [a, b, ,c, 1, 2, 3]).

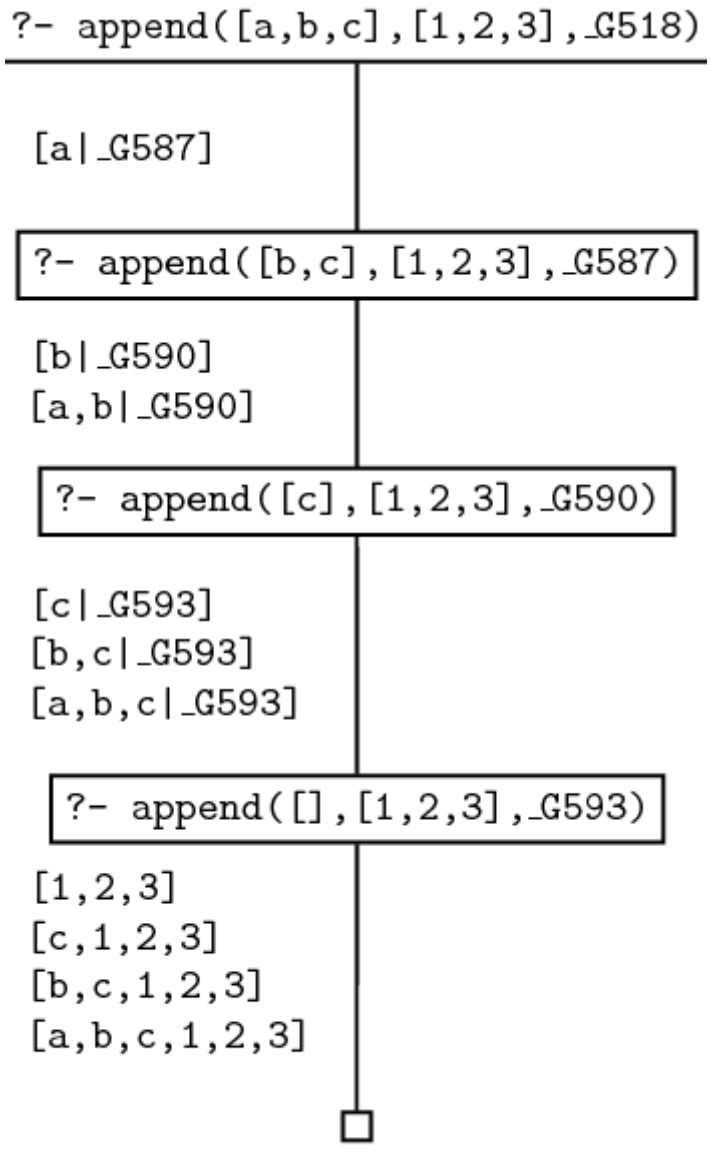
X = [a, b, c, 1, 2, 3].

true

这种求解的模式很清晰：前四行可以看见Prolog通过递归定义的方法遍历完第一个列表；然后，接下来的四行显示，Prolog接下来回填每个中间结果，如何执行的呢？通过依次初始化变量：

_G593，_G590，_G587和_G518。但是这里学习的关键是把握这种基础模式，而不仅仅局限于append/3的实现。所以，我们可以更深入地研究，如下是查询，append([a, b, c], [1, 2, 3], X)

的搜索树，我们将会仔细标记每一个步骤，每一个中间目标，及其每一个变量的初始化值：



1. Goal 1：append([a, b, c], [1, 2, 3], _G518)。Prolog将其和递归规则的头部合一。所以_G518和[a | T3]合一，同时Prolog有了新的目标，append([b, c], [1, 2, 3], T3)，这会为T3生成

一个新的变量，_G587，所以我们可以知道：_G518 = [a | _G587]。

2. Goal 2：append([b, c], [1, 2, 3], _G587)。Prolog将其和递归规则的头部合一，所以_G587和[b | T3]合一，Prolog有了新的目标，append([c], [1,2 ,3], T3)，这会为T3生成一个新的变量，

_G590，所以我们可以知道：_G587 = [b | _G590]。

3. Goal 3：append([c], [1, 2, 3], _G590)。Prolog将其和递归规则的头部合一，所以_G590和[c | T3]合一，Prolog有了新的目标，append([], [1, 2, 3], T3)，这会为T3生成一个新的变量，

_G593，所以我们可以知道：_G590 = [c | _G593]。

4. Goal 4：append([], [1, 2, 3], _G593)。最终，Prolog可以使用基础子句（即，append([], L, L)）了，所以在接下来的连续4个步骤，Prolog会获取Goal 4, Goal 3, Goal 2, Goal 1的答案：

5. Goal 4 的答案： append([], [1, 2, 3], [1, 2, 3])，这是因为当我们使用基础子句匹配Goal 4时，_G593将和[1, 2, 3]合一。

6. Goal 3 的答案： append([c], [1, 2, 3], [c, 1, 2, 3])，为什么？因为 Goal 3 是，append([c], [1, 2, 3], _G590)，同时_G590是列表[c | _G593]，我们已经将_G593和[1, 2, 3]合一，所以

_G590将和[c, 1, 2, 3]合一。

7. Goal 2 的答案：append([b, c], [1, 2, 3], [b, c, 1, 2, 3])。为什么？因为Goal 2是，append([b, c], [1, 2, 3], _G587)，同时_G587是列表[b | _G590]，我们已经将_G590和[c, 1, 2, 3]合一，

所以_G587将和[b, c, 1, 2, 3]合一。

8. Goal 1 的答案：append([a, b, c], [1, 2, 3], [a, b, c, 1, 2, 3])，为什么？因为Goal 1是，append([a, b, c], [1, 2, 3], _G518)，同时_G518是列表[a | _G587]，我们已经将_G587和

[b, c, 1, ,2 ,3]合一，所以_G518将和[a, b, c, 1, 2, 3]合一。

9. 所以Prolog现在已经知道如何初始化X，这个原始的查询变量，它会告诉结果，X = [a, b, c, 1, 2, 3]，正如我们期望的一样。

请仔细思考上面的例子，并且确保完全理解变量初始化的模式，即：

_G518 = [a | _G587]

= [a | [b | _G590]]

= [a | [b | [c | _G593]]]

这种类型的模式就是append/3能够起作用的核心。而且，它展示出更为普通的主题：使用合一去构建结构。在核心层，append/3的递归调用会构建出嵌套模式的变量。当Prolog最终将

最里层的变量_G593和[1,2 ,3]合一，答案就会循环得出，就像是滚雪球一般。但是这仅仅是合一，不是其他什么魔法，就得出了结果。

合并列表的使用

现在我们已经了解了append/3的工作机制，来看看如何实际运用它。

append/3的一个重要应用就是分割一个列表为两个连续的列表，比如：

```
?- append(X, Y, [a, b, c, d]).

X = []

Y = [a, b, c, d];

X = [a]

Y = [b, c, d];

X = [a, b]

Y = [c, d];

X = [a, b, c]
```

Y = [d];

X = [a, b, c ,d]

Y = [];

false

即，我们给出想要分割的列表（这里就是，[a, b, c, d]）作为append/3的第三个参数，同时我们使用变量代表前两个参数。Prolog就会进行搜索，将两个变量初始化，并且合并后的值就是

第三个参数的列表，即分割列表为两个。而且，正如例子的结果显示，通过回溯，Prolog能够找到所有可能的组合值。

可以使用append/3定义许多其他有用的谓词。让我们思考一些例子。首先，我们可以定义一个谓词，找到列表的前缀，比如[a, b, c, d]的前缀是，[], [a], [a, b], [a, b, c]和[a, b, c, d]，在

append/3的协助下，定义prefix/2很容易，其中两个参数都是列表，比如prefix(P, L)将会得出P是L的前缀，如下：

```
prefix(P, L) :- append(P, _, L).
```

这个定义就是说，列表P是列表L的前缀，如果列表L是由列表P和其他列表合并而成的（使用匿名变量代表我们不在乎其他列表具体是什么，我们只是知道这里有其他列表存在）。这个谓词可以

成功找出列表的前缀，并且通过回溯，可以找出所有的可能值：

?- prefix(X, [a, b, c, d]).

X = [];

X = [a];

X = [a, b];

X = [a, b, c];

X = [a, b, c, d];

false

类似地，我们可以定义一个谓词找出一个列表的后缀。比如，[a, b, c, d]的后缀是[], [d], [c, d], [b, c, d]和[a, b, c, d]。同样地，使用append/3可以方便地定义suffix/2，其中两个参数都是列表，

比如suffix(S, L)将会得出S是L的后缀，如下：

```
suffix(S, L) :- append(_, S, L).
```

这个定义就是说，列表S是列表L的后缀，如果列表L是由其他列表和列表S合并而成的，这个谓词可以成功找出列表的后缀，并且通过回溯，可以找出所有的可能值：

?- suffix(X, [a, b, c, d]).

X = [a, b, c, d];

X = [b, c, d];

X = [c, d];

X = [d];

X = [];

false

请确认你能够理解为什么答案是这样的顺序。

现在，可以十分轻松地定义一个谓词找出列表的子列表。[a, b, c, d]的子列表是，[], [a], [b], [c], [d], [a, b], [b, c], [c, d], [a, b, c], [b, c, d]和[a, b, c, d]。稍微想一下就能够知道一个列表的子

列表是这个列表的后缀的前缀，如图：

获取后缀： a, b, c, d, e, f, g, $\underbrace{h, i, j, k, l, m, n, o, p}$

获取后缀：$\underbrace{h, i, j, k, l}$，m, n, o, p

结果：h, i, j, k, l

由于我们已经定义了列表前缀和后缀的谓词，所以子列表的谓词定义如下：

sublist(SubL, L) :- suffix(S, L), prefix(SubL, S).

即，SubL是L的子列表，如果存在列表S：S是L的后缀，并且SubL是S的前缀。这个谓词没有直接使用append/3，但是由于prefix/2和sufiix/2都使用了append/3，所以内部其作用的还是append/3。

分类: Prolog

标签: 列表合并, append

seaman.kingfall
关注 - 1
粉丝 - 4
+加关注

« 上一篇: Learn Prolog Now 翻译 - 第五章 - 数字运算 - 第四节，练习题和答案
» 下一篇: Learn Prolog Now 翻译 - 第六章 - 列表补遗 - 第二节，列表反转

posted on 2015-07-21 13:25 seaman.kingfall 阅读(753) 评论(0) 编辑 收藏