Learn Prolog Now 翻译 - 第十一章 - 知识库相关操作和解决方案的收集 - 第二节， 解决方案的收集

解决方案的收集问题

在Prolog的查询中，可能存在会有很多解决方案的情况。比如，假设有如下的知识库：

```
child(martha, charlotte).
child(charlotte, caroline).
child(caroline, laura).
child(laura, rose).

descend(X, Y) :- child(X, Y).
descend(X, Y) :- child(X, Z), descend(Z, Y).
```

如果我们进行查询：

```
?- descend(martha, X).
```

将会有四个满足条件的解决方案（分别是：X = charlotte, X = caroline, X = laura, X = rose）。

但是Prolog是一个接一个地去生成解决方案的。有些时候我们希望能够通过一个查询得出所有解决方案，并且希望以一种干净、可用、形式化的结果来显示。Prolog有三个内置谓词可以用来达成这个目标，分别是：findall，bagof，和setof。从本质上来说，所有的这些谓词都是通过一个查询收集所有的解决方案，并将其放在一个列表中，但是它们之间存在重要的不同之处，我们将会详细学习。

findall/3谓词

如下的查询：

```
?- findall(Object, Goal, List).
```

将会构建一个所有Object满足目标Goal的对象列表。通常来说，Object是一个简单的变量，这个查询可以读做：请给我一个包含所有Object初始化后能够满足Goal的值的集合。

下面是一个例子。假设使用上面的知识库，如果进行查询：

```
?- findall(X, descend(martha, X), Z).
```

我们希望得到一个列表Z，其中包含了所有X能够满足descend(martha, X)的值，Prolog会回答：

--Benjamin Yan


```
X = _7489
Z = [charlotte, caroline, laura, rose]
```

但是Object不是必须为一个变量，它也可能是一个包含了变量的复杂语句，其中的变量也会出现在Goal中。比如，我们希望构建一个新的谓词fromMartha/1，当其中的参数为Martha的后辈时为真，那么可以进行如下的查询：

```
?- findall(fromMartha(X), descend(martha, X), Z).
```

即，希望能够得到一个列表Z，其中包含了所有fromMartha(X)能够满足descend(martha, X)的值，Prolog会回答：

```
X = _7616
Z = [fromMartha(charlotte), fromMartha(caroline),
fromMartha(laura), fromMartha(rose)]
true
```

如果进行下面的查询，会发生什么？

```
?- findall(X, descend(mary, X), Z).
```

因为知识库中没有满足条件的解决方案，所以findall/3会返回一个空列表。

也有可能存在这种情况，findall/3前两个参数不是同一个变量，但是findall/3也能起到作用。比如，如果你对具体是那些人是Martha的后辈不感兴趣，而只是关心Martha有多少个后辈，可以使用下面的查询：

```
?- findall(Y, descend(martha, X), Z), length(Z, N).
```

bagof/3谓词

谓词findall/3很有用，但是在有些方面显得比较粗糙。比如，进行下面的查询：

```
?- findall(Child, descend(Mother, Child), List).
```

会得到如下的答案：

```
Child = _6947
Mother = _6951
List = [charlotte, caroline, laura, rose, caroline, laura,
rose, laura, rose, rose]
```

这个答案是正确的，但是有时候，我们希望根据不同的Mother，有不同的分组结果显示，这样可读性会更好。bagof/3可以达到这个目的，如果进行查询：

```
?- bagof(Child, descend(Mother, Child), List).
```

将会得到如下的结果：

```
Child = _7736
Mother = caroline
List = [laura, rose];

Child = _7736
```

```
Mother = charlotte
List = [caroline, laura, rose];

Child = _7736
Mother = laura
List = [rose];

Child = _7736
Mother = martha
List = [charlotte, caroline, laura, rose];

false
```

可以看出，bagof/3比findall/3可读性更好，它能够获取出我们期望的更具结构性的解决方案。而且，bagof/3可以完成findall/3相同的工作，借助一个特殊的语法，名为^：

```
?- bagof(Child, Mother^descend(Mother, Child), List).
```

上面查询的含义是：给出一个列表，其中的元素都是Child的值，并且满足descend(Mother, Child)，同时不需要为每个Mother生成特殊的结果列表，所以Prolog的回答是：

```
Child = _7870
Mother = _7874
List = [charlotte, caroline, laura, rose, caroline, laura,
rose, laura, rose, rose]
```

请注意这里的结果和使用findall/3的完全一致。所以，如果你希望使用简单方式获取所有的解决方案，建议还是使用findall/3，因为不需要显式地使用^进行拼接。

findall/3和bagof/3之间有一个重要的区别，就是如果bagof的第二个参数的目标无法满足时，查询会导致失败（findall/3这种情况会返回空列表），所以查询bagof(X, descend(mary, X), Z)会返回false。

最后需要注意的是，思考下面的查询：

```
?- bagof(Child, descend(Mother, Child), List).
```

正如我们上面看到的，这里存在四个解决方案。但是，Prolog是一个接一个去生成这些结果的。有没有一种更好的方式将这些解决方案收集到一个列表中？确实可以做到，最简单的方式是使用findall/3，进行查询：

```
?- findall(List, bagof(Child, descend(Mother, Child), List),
Z).
```

将bagof/3的解决方案收集到一个列表中：

```
List = _8293
Child = _8297
Mother = _8301
Z = [[laura, rose], [caroline, laura, rose], [rose],
[charlotte, caroline, laura, rose]]
```

另外一种方式是使用bagof/3：

```
?- bagof(List, Child^Mother^bagof(Child, descend(Mother,
Child), List), Z).
```

```
List = _2648
Child = _2652
Mother = _2655
Z = [[laura, rose], [caroline, laura, rose], [rose],
[charlotte, caroline, laura, rose]]
```

以上例子可能不会经常使用到，但是它们演示了这些谓词能够提供的灵活性和强大功能。

setof/3谓词

谓词setof/3和bagof/3很类似，但是存在一个有用的区别：setof/3得出的结果列表中的元素是有顺序，并且没有冗余的（即，列表中不包含重复的元素）。

比如，假设存在下面的知识库：

```
age(harry, 13).
age(draco, 14).
age(ron, 13).
age(hermione, 13).
age(dumbledore, 60).
age(hagrid, 30).
```

现在假设我们希望得到一个包含所有在知识库中有描述年龄的人的列表，可以进行如下查询：

```
?- findall(X, age(X, Y), Out).

X = _8443
Y = _8448
Out = [harry, draco, ron, hermione, dumbledore, hagrid]
```

但是我们希望能够得出排序后的列表，可以使用如下的查询：

```
?- setof(X, Y^age(X, Y), Out).
```

注意，和bagof/3类似，如果我们不希望setof/3为每个Y单独生成X的列表，那么使用^进行过滤，得到的答案是：

```
X = _8711
Y = _8715
Out = [draco, dumbledore, hagrid, harry, hermione, ron]
```

注意上面的列表是根据字母排序的。

现在假设我们希望能够收集知识库中所有出现的年龄，可以进行如下的查询：

```
?- findall(Y, age(X, Y), Out).

X = _8847
Y = _8851
Out = [13, 14, 13, 13, 60, 30]
```

但是这个结果比较粗糙，没有排序，也包含了重复元素。通过使用setof/3，我们可以得到更加整洁的列表：

```
?- setof(Y, X^age(X, Y), Out).

X = _8981
Y = _8985
Out = [13, 14, 30, 60]
```

以上三个谓词为我们收集查询的结果提供了很强的灵活性。对于大多数的情况，findall/3可以搞定，但是如果需要更加精细的输出结果，可以结合bagof/3和setof/3来达成。但是请记住findall/3和两位两个谓词之间的重要不同：在没有符合条件的情况下，findall/3会返回空列表，其余两个谓词则会返回错误。

分类: Prolog

标签: 解决方案的收集, findall, bagof, setof

seaman.kingfall
关注 - 1
粉丝 - 4
+加关注

« 上一篇: Learn Prolog Now 翻译 - 第十一章 - 知识库相关操作和解决方案的收集 - 第一节， 知识库相关操作
» 下一篇: Learn Prolog Now 翻译 - 第十二章 - 文件相关操作 - 第一节， 使用不同文件组织程序

posted on 2015-08-06 16:15 seaman.kingfall 阅读(648) 评论(0) 编辑 收藏