Learn Prolog Now 翻译 - 第十章 - 中断和否定 - 第一节， 中断

中断

自动回溯是Prolog中很有代表性的一个特征。但是回溯可能会导致低效。有时Prolog会浪费时间在一些没有结果的可能性搜索上。如果在回溯行为方面有一些控制机制的话，会是一件比较有意义的事情，但是直到现在为止我们看到只有两种相当初级的方式可以用于这个目的：交换规则顺序，和交换目标顺序。其实有另外一种方式：存在一个内置的谓词：！(英文感叹号)，称为中断，可以提供一种更为直接地控制Prolog搜寻解决方案的方式。

到底中断是什么，它是如何起作用的？它其实是一个特殊的原子，我们可以在子句中使用它。看例子：

```
p(X) :- b(X), c(X), !, d(X), e(X).
```

上面的例子就是一个完美的Prolog规则。中断是这样起作用的：首先，它是一个永远成功的目标；其次，更为重要的是，它有一个副作用。假设有其他一些目标会使用这个子句（我们称之为父目标，比如例子中的p(X)），在进行搜索时，中断会使得规则左边的目标无法回溯，而只能回溯规则右边的目标。让我们通过例子来学习。

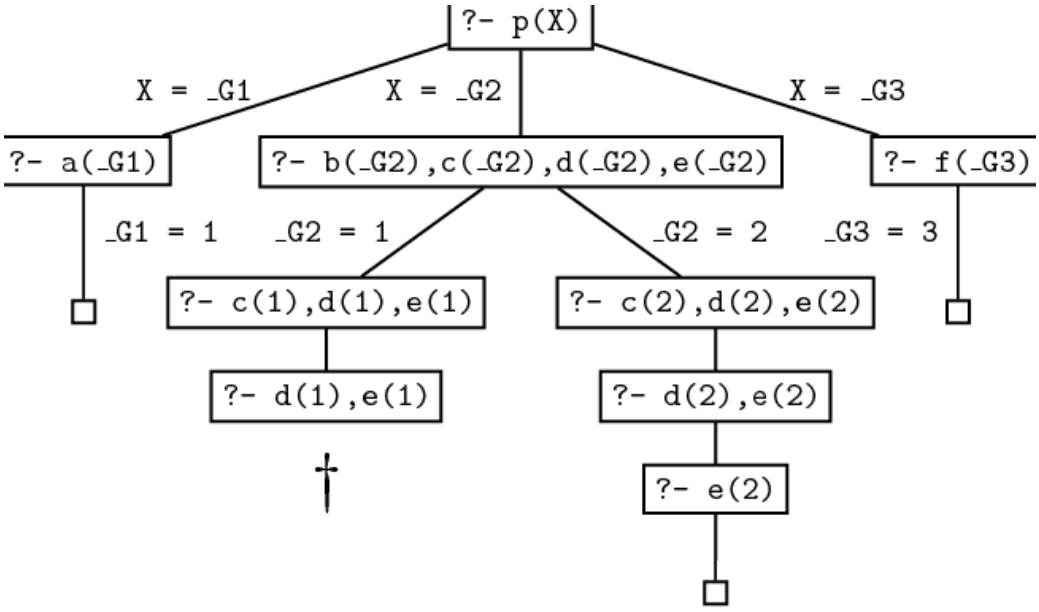
首先思考没有中断的代码：

```
p(X) :- a(X).
p(X) :- b(X), c(X), d(X), e(X).

p(X) :- f(X).

a(1). b(1). b(2). c(1). c(2). d(2). e(2). f(3).
```

如果查询p(X)，我们会得到如下的答案：

```
?- p(X).
X = 1;
X = 2;
X = 3;
false
```

下面是对应的搜索树，注意其中必须回溯的地方，当进入第二个子句，决定满足第一个目标b(1)，回溯后被替换为b(2)。



现在假设我们在第二个子句中加入中断：

```
p(X) :- b(X), c(X), !, d(X), e(X).
```

如果现在查询p(X)，会得到如下的答案：

4. Haskell学习笔记二：自定义类型(767)
5. Learn Prolog Now 翻译 - 第六章 - 列表补遗 - 第一节，列表合并(753)

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)

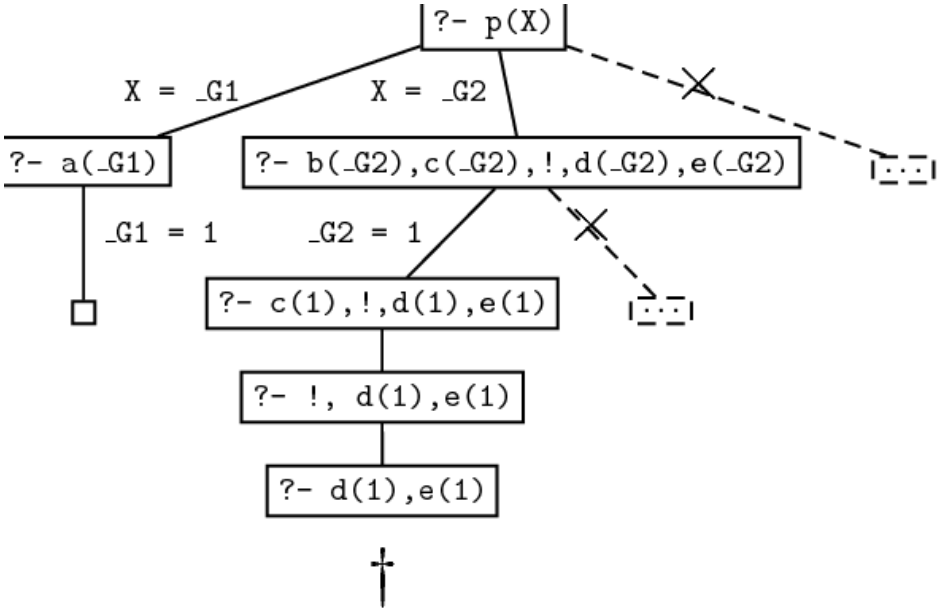
推荐排行榜
1. Haskell学习笔记二：自定义类型(1)
2. Learn Prolog Now 翻译 - 第三章 - 递归 - 第四节，更多的实践和练习(1)

```
X = 1;
false
```

这里发生了什么？让我们思考一下。

1. p(X)首先和第一个子句合一，所以得到新的目标 a(X)。通过将X初始化为1，Prolog将a(X)和事实a(1)合一，从而找到了一个解决方案。目前为止，发生的一切和第一个版本是一致的。
2. 当我们继续搜索第二个解决方案。p(X)与第二个规则合一，所以我们获得新的目标：b(X), c(X), !, d(X), e(X)。通过将X初始化为1，Prolog将b(X)和b(1)合一，所以我们获得新的目标：c(1), !, d(1), e(1)。同时c(1)是知识库中存在的事实，所以目标简化为：!, d(1), e(1)。
3. 现在到了发生巨变的时刻。目标!为真（正如其定义的，这是一个永真的目标）并且会提交目前为止的选择。具体来说，我们会提交X = 1，同时我们也会提交使用的第二个规则。
4. 但是d(1)失败了。这样我们就无法满足目标p(X)。当然，如果我们允许重试将X初始化为2，我们能够使用第二个规则去生成一个解决方案（就是在原始版本程序中发生的）。但是这里我们无法这样做：中断已经在搜索树种删除了这种可能性。同时，如果允许尝试第三个规则，也可以生成X = 3的解决方案。但是我们还是无法这样做：中断同样从搜索树中删除了这种可能性。

如果观察如下的搜索树，你会发现一些树枝被删除：当目标d(1)不能在进行，但是需要回溯寻找新的选择时，搜索已经被停止：



有一个需要强调的要点：中断会提交所有的选择，这些选择是为了满足父目标而将包含中断的子句合一时做出的，并且是从中断左端进行合一的那些选择。比如，在如下规则的模式中：

```
q :- p1, ..., pn, !, r1, ..., rm
```

当我们到达中断的时候，Prolog会提交为了满足q的并且包含中断的子句的所有选择，并且这些选择是运算p1, ..., pm得出的。然而，在r1, ..., rm内，我们能够进行回溯，并且满足目标q之前的其他选择我们也可以进行回溯。通过看下面的例子来明确这些原理。

首先思考没有中断的程序：

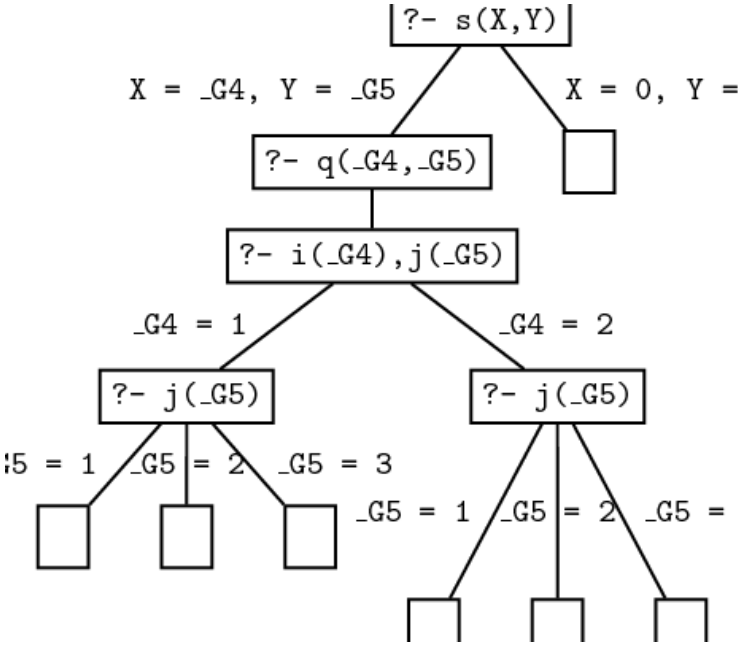
```
s(X, Y) :- q(X, Y).
s(0, 0).

q(X, Y) :- i(X), j(Y).

i(1).
i(2).
j(1).
j(2).
j(3).
```

如下是查询和结果：

```
?- s(X, Y).

X = 1
Y = 1;
```

```
X = 1
Y = 2;

X = 1
Y = 3;

X = 2
Y = 1;

X = 2
Y = 2;

X = 2
Y = 3;

X = 0
Y = 0;
false
```

下面是对应的搜索树：



假设我们在q/2子句中加入中断：

```
q(X, Y) :- i(X), !, j(Y).
```

现在程序的行为如下：

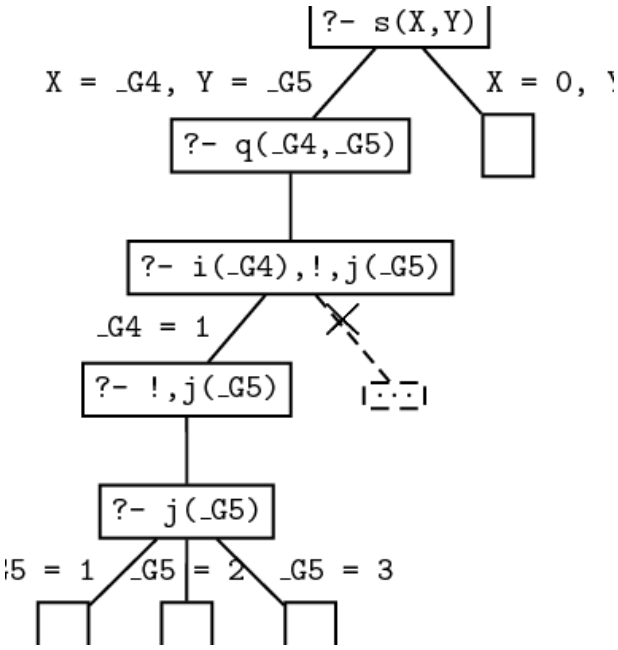
```
?- s(X, Y).

X = 1
Y = 1;

X = 1
Y = 2;

X = 1
Y = 3;

X = 0
Y = 0;
false
```

让我们看看为什么：

1. s(X, Y)首先和第一个规则合一，这会给出新的目标：q(X, Y)。
2. q(X, Y)接着和第三个规则合一，这会给出新的目标：i(X), !, j(Y)。通过将X初始化为1，Prolog将i(X)和事实i(1)合一，这会得出新的目标：!, j(Y)。中断当然为真，同时会提交直到现在为止做出的选择。
3. 但是有哪些选择呢？这里存在：X = 1，和我们正在使用的子句。但是注意：我们没有为Y选择任何的值。
4. Prolog会继续，通过将Y初始化为1，Prolog将j(Y)和事实j(1)合一，所以我们找到了一个解决方案。

5. 但是我们能够找到更多解决方案。Prolog能够对Y尝试其他的值。所以回溯并且将Y初始化为2，所以这样找到了第二种解决方案。事实上还可以继续找到解决方案：再次回溯，通过将Y初始化为3，找到第三种解决方案。
6. 但是这些都是搜索j(X)的匹配值，在中断左边的回溯是不允许的，所以无法将X重新初始化为2，所以这里无法找到类似没有中断程序中X = 2的那些解决方案。回溯到达q(X, Y)之前的目标是允许的，所以Prolog会找到s/2的第二个规则子句。

如下式对应的搜索树：



分类: Prolog

标签: 回溯, 中断

seaman.kingfall
关注 - 1
粉丝 - 4
+加关注

« 上一篇: Learn Prolog Now 翻译 - 第九章 - 语句深究 - 第四节， 操作符
» 下一篇: Learn Prolog Now 翻译 - 第十章 - 中断和否定 - 第二节， 中断的运用

posted on 2015-08-01 09:08 seaman.kingfall 阅读(363) 评论(0) 编辑 收藏

相关博文:

· Learn Prolog Now 翻译 - 第十章 - 中断和否定 - 第二节， 中断的运用
· Learn Prolog Now 翻译 - 第四章 - 列表 - 第一节，列表定义和使用

· Learn Prolog Now 翻译 - 第十章 - 中断和否定 - 第三节， 使用否定作为失败判定
· Learn Prolog Now 翻译 - 第三章 - 递归 - 第一节，递归的定义
· Learn Prolog Now 翻译 - 第五章 - 数字运算 - 第一节，Prolog中的数字运算