Document for part 1

Language

程序使用prolog（SWI-Prolog），测试需配置此环境。

Cross River

实现思路

1. 状态表示。解决一个问题，数据结构的构造尤为重要。本实现通过状态迁移来达到最终状态，状态迁移的过程即为解。状态的表示使用模板**state(Ship,Police,Criminal,Father,Mother,Son1,Son2,Daughter1,Daughter2)**/9，其中每一个变量均可取w或e两个值，表示在河的某一岸。
2. 状态迁移。根据问题描述，我们可以穷举出所有可能的状态迁移。在prolog中，可以用规则来表示这些状态迁移（见代码）。状态迁移有两个要求，一是可划船人和载人数量的限制，这一条决定了状态迁移规则的数量。二是迁移到的状态要是安全状态。（见下条）
3. 安全状态。根据问题描述，我们可以很容易写出不安全状态的条件，取一个非则是安全状态。
4. 有了以上准备之后，我们可以用简单的递归来找到一个解。

```
%the end rule
path(X,X,List):-print(List).

%the recursion path rule
path(Now,Goal,List):-
    % get next possible state
    move(Now,Next),
    % Next is not already in the list
    not(member(Next,List)),
    % add to the list
    append(List,[Next],NewList)
    % recurion for answer.
    path(Next,Goal,NewList).
```

找到最优解

上述实现，按照prolog的消解过程实际上相当于深搜，只是找到了一个解，而不是最优解。找到最优解，按照习惯首先想到广搜，但是用prolog实现广搜实在麻烦；亦或也可以跑完所有的结果，挑出最优解，但是这很low，又很慢。

我的实现方法如下。算法非常简单但是很巧妙，利用了prolog的回溯，代码上与上述相比只是颠倒了顺序。

```
%the end rule
path(X,X,[]).%1

%the recursion path rule.%2
path(Now,Goal,List):-
    %recurion for answer until get next state
    path(Now,Next,NewList),
    not(member(Next,NewList)),
    %require that the Next can change to Goal
    %（not the final goal,but Next in previous path-invoking）.
    move(Next,Goal),
    append(NewList,[Next],List).
```

但是其执行逻辑过程就完全不一样了。如下：

1. 尝试能否经过一步到达目标状态，如果不能，prolog会自动回溯，放弃匹配第一条规则，而匹配第二条规则。
2. 匹配第二条规则就变成了尝试能否经过两步到达目标状态。否则，如上回溯。
3. 直到能够经过n步达到目标状态，程序找到一个解。那这个解自然是最优解之一了。

运行方法

1. 导入cross_river.pl的事实和规则。
2. 在命令行输入：**go(state(w,w,w,w,w,w,w,w,w),state(e,e,e,e,e,e,e,e,e)).**
3. 得到答案如下（不包含注释符号和换行，state模板如前面的定义）：

```
%17-steps.
%[
%state(w,w,w,w,w,w,w,w,w),state(e,e,e,w,w,w,w,w,w),
%state(w,w,e,w,w,w,w,w,w),state(e,e,e,w,w,e,w,w,w),
%state(w,w,w,w,w,e,w,w,w),state(e,w,w,e,w,e,e,w,w),
%state(w,w,w,w,w,e,e,w,w),state(e,w,w,e,e,e,e,w,w),
%state(w,w,w,e,w,e,e,w,w),state(e,e,e,e,w,e,e,w,w),
%state(w,e,e,w,w,e,e,w,w),state(e,e,e,e,e,e,e,w,w),
%state(w,e,e,e,w,e,e,w,w),state(e,e,e,e,e,e,e,e,w),
%state(w,w,w,e,e,e,e,e,w),state(e,e,w,e,e,e,e,e,e),
%state(w,w,w,e,e,e,e,e,e),state(e,e,e,e,e,e,e,e,e)
%]
```

Einstein's Puzzle

实现思路

1. 房屋表示。同样，我们需要为房屋找到好的表示方法。模板：**house(Color,People,Drink,Cigarette,Pet)**/5.
2. 规则表达。有了如上房屋表示之后，问题描述的规则表达成prolog语言就非常简单。唯一复杂一点的是对领居关系的表达，不过由于只有五间房，穷举一下就好了，如下：

```
%X and Y are neibours in H
neibour(X,Y,H):-
    H=[X,Y,_,_,_];H=[_,X,Y,_,_];H=[_,_,X,Y,_];H=[_,_,_,X,Y];
     %Y < X
    H=[Y,X,_,_,_];H=[_,Y,X,_,_];H=[_,_,Y,X,_];H=[_,_,_,Y,X].
```

3. 满足所有规则之后，就已经得到了信息明确的5间房，如下：

```
%H = [
%house(yellow, norwegian, water, dunhill, cat),
%house(blue, dane, tea, blend, horse),
%shouse(red, english, milk, pall_mall, bird),
%house(green, german, coffee, prince, _G165586),
%house(white, swedish, beer, blue_master, dog)
%];
```

运行方法

1. 导入einstein_puzzle.pl的事实和规则
2. 在命令行输入：**keep(X,fish)**，然后得到**X = german**.
3. 亦可问其他问题，但是只提供了keep的接口，即谁养了什么样的宠物的问题。
4. 在命令行输入**houses(H)**，即可得到所有房间的情况，如前一部分第三条所示

Which Car Do Each Man Own

实现思路

1. 车子表示。同样，根据问题描述定义车子的模板**car(Owner,Brand,mpg)**/3。
2. 与前一问题不同的是，该问题加入了谎言，而谎言要等到car的状态都决定了之后才能判断。所以处理方法是先按照排列生成一个可能的状态。
3. 有了车子的状态之后，就可以确定人是否撒谎，然后适配对应的规则，样例如下：

```
%Tito said: Doc gets 20 miles per gallon of gas.
% George's gas mileage is better than Jimmy's
((TM > 20,DM = 20,GM > JM);%truth
not(TM > 20;DM = 20;GM > JM)),%lies
```

4. 最终判断完所有规则之后，即可得该解是否是合理的。如不满足所有条件，prolog最终会回溯到第2步。
5. 最终可得一个满足所有条件的解，程序验证这是唯一解。

运行方法

1. 导入man_car.pl的事实和规则
2. 在命令行输入：**answer()**或者**cars(C)**,即可获得解，如下：

```
% C=[
%   car(george, chevrolet, 25),
%   car(doc, dodge, 10),
%   car(tito, toyota, 20),
%   car(jimmy, ford, 30)
%] ;
```