

猴子吃香蕉问题的解决

《人工智能导论》实验报告

专 业 名 称： 计算机科学与技术

学 生 姓 名： 马玉坤

学 生 学 号： 1150310618

指 导 教 师： 李钦策

同 组 人 员： 李伟枫, 许浩禹, 张宁

二〇一八年一月

摘 要

在此实验中，我们查阅资料 [1]，编写程序，使用产生式系统、框架系统、语义网络等解决了猴子吃香蕉的问题。

关键词: 人工智能导论; 产生式模型; 知识表示

目 录

| | |
|---------------------------|----------|
| 摘要 | I |
| 1 简介/问题描述 | 1 |
| 1.1 解决问题的解释 | 1 |
| 1.2 问题的形式化描述 | 1 |
| 1.3 解决方案介绍（原理） | 1 |
| 2 算法介绍 | 2 |
| 2.1 所用方法的一般介绍 | 2 |
| 2.2 算法伪代码 | 2 |
| 3 算法实现 | 3 |
| 3.1 实验环境与问题规模 | 3 |
| 3.1.1 实验环境 | 3 |
| 3.1.2 问题规模 | 3 |
| 3.2 数据结构 | 3 |
| 3.3 实验结果 | 3 |
| 3.4 系统中间及最终输出结果 | 3 |
| 4 总结及讨论 | 5 |
| 参考文献 | 6 |
| 附录 A 实验代码 | 7 |

1 简介/问题描述

1.1 解决问题的解释

查阅资料 [1]，编写程序，使用产生式系统、框架系统、语义网络等解决猴子吃香蕉的问题。

1.2 问题的形式化描述

通过给出当前系统的状态与最终状态，计算所需的操作能够使猴子达到最终状态，输出这个操作序列。

1.3 解决方案介绍（原理）

规则库中有五条规则，而综合数据库中只有初始状态。

接下来我们使用程序，对综合数据库中的每个状态，使用规则库中的规则产生新的状态，然后将新状态加入到综合数据库中。重复这一步骤，直到找到目标状态。根据存储的状态转移序列，即可得到所需的操作序列。

2 算法介绍

2.1 所用方法的一般介绍

我们使用了广度优先搜索算法。使用队列来存储综合数据库中的新状态，直到找到目标状态。

2.2 算法伪代码

算法 1 通用搜索算法

输入: *initialState* 初始状态, *finalState* 最终状态

输出: 从初始状态转移到最终状态所需的操作序列

- 1: 将开始状态放在综合数据库
 - 2: **while** 最终状态不在综合数据库中 **do**
 - 3: **for** $x \in$ 综合数据库找到所有没有扩展过的状态 **do**
 - 4: 对这些状态使用规则库中的五个规则进行状态转移
 - 5: 将转移得到的状态 y 加入到综合数据库之中
 - 6: 并且记录这个状态 y 的父亲状态是 x
 - 7: **end for**
 - 8: **end while**
 - 9: 根据最终状态的父亲节点递归寻找, 直到找到最初状态, 打印状态序列. 并且根据状态序列找到猴子的操作序列.
-

3 算法实现

3.1 实验环境与问题规模

3.1.1 实验环境

- CPU: Intel E3-1230 V5
- IDE: Code::Blocks
- 操作系统: Windows 10

3.1.2 问题规模

当前系统状态数目是一个五元组, 合法的状态数目很少, 所以数据规模很小。

3.2 数据结构

在定义状态时, 我们使用了五元组来表示 $((M, B, Box, On, H))$ 。

其中 M 表示猴子的位置, B 表示香蕉的位置, Box 表示箱子的位置, $On = 0$ 表示猴子在地板上; $On = 1$ 表示猴子在箱子上; $H = 0$ 表示猴子没有抓到香蕉; $H = 1$ 表示猴子抓到了香蕉。

为了处理方便, 我们将状态转化成了一个五位 *int* 整形数据, 我们使用一个长为 99999 的数组来表示综合数据库, 当数组中下标为 i 的元素不是 0 的时候, 表示整数 i 对应的状态在综合数据库之中, 同时我们使用 *state_father* 数组来记录状态的父亲状态。如果我们使用状态 i 利用规则库之中的规则产生了状态 j , 那么数组中下标为 j 的位置存储的就是 i 。

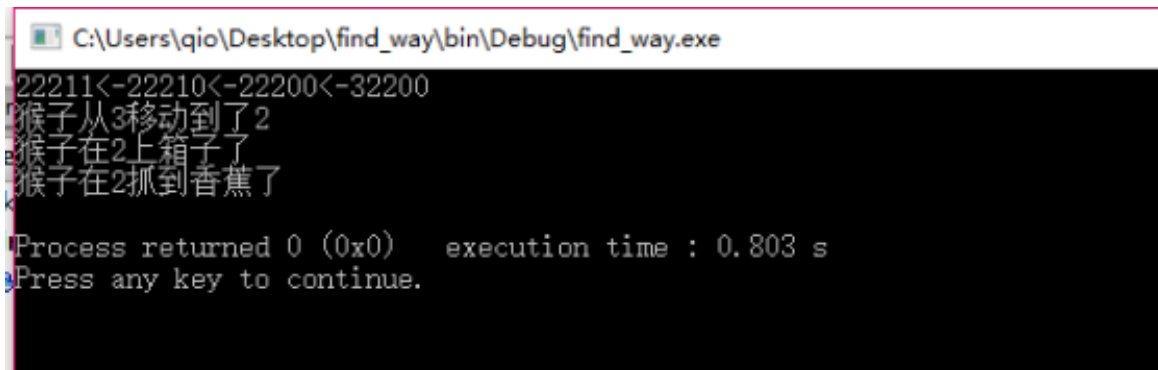
3.3 实验结果

我们的程序能够根据猴子的最初状态和最终状态找到一个合法的状态转移序列, 并且能够根据这个状态转移序列给出猴子每一步需要作出什么操作。

3.4 系统中间及最终输出结果

如图3.1所示, 系统最初状态为 32200, 此时猴子从 3 移动到了 2, 状态变为 22200。

接着猴子在 2 上箱子, 状态又变成了 22210, 最后猴子在 2 抓到香蕉了, 系统到达了最终的状态 22211。



```
C:\Users\qio\Desktop\find_way\bin\Debug\find_way.exe
22211<-22210<-22200<-32200
猴子从3移动到了2
猴子在2上箱子了
猴子在2抓到香蕉了

Process returned 0 (0x0)   execution time : 0.803 s
Press any key to continue.
```

图 3.1 结果截图

4 总结及讨论

通过本次实验，我们认识到了产生式系统的工作原理，进一步了解了规则库、控制系统和综合数据库的概念。

本次实验中，我与其他组员分工明确，合作无间，完成了本次实验。通过本次实验，我们的友谊得到了升华，对人工智能导论课程内容的理解进一步提高。

参考文献

- [1] 人工智能原理及其应用. 高等学校教材. 电子工业出版社, 2000.

附录 A 实验代码

```
#include <iostream>
#include <string.h>
#include <stdio.h>
using namespace std;
int state_set[99999]; //记录综合数据库
int state_father[99999]; //记录产生该状态的父亲状态
int state_son[99999]; //记录产生正确结果的过程之中，某一个状态的后继状态

int next_state; //记录由状态和规则产生的下一个状态

const int start_state=32200; //用全局变量来表示开始的节点和结束的节点
const int end_state=22211;

int mat[5];
int mat2[5];
void get_value(int i) //将输入的状态 i 存储到 mat 数组之中
{
    mat[0]=i/10000;
    mat[1]=(i%10000)/1000;
    mat[2]=(i%1000)/100;
    mat[3]=(i%100)/10;
    mat[4]=(i%10);
}
void get_value2(int i) //将输入的状态存储到 mat2 数组之中
{
    mat2[0]=i/10000;
    mat2[1]=(i%10000)/1000;
    mat2[2]=(i%1000)/100;
    mat2[3]=(i%100)/10;
    mat2[4]=(i%10);
}

void print(int a,int b) //打印从状态 a 到状态 b 需要执行的操作
```

```

{
    get_value(a);
    get_value2(b);
    //根据状态 a 与状态 b 的区别找到猴子需要执行什么操作
    if(mat[4]!=mat2[4])
    {
        printf(" 猴子在%d 抓到香蕉了\n",mat[0]);
        return;
    }
    if(mat[3]!=mat2[3])
    {
        if(mat2[3]==0)
        {

            printf(" 猴子在%d 下箱子了\n",mat[0]);return;
        }
        if(mat2[3]==1)
        {

            printf(" 猴子在%d 上箱子了\n",mat[0]);return;
        }

    }
    if(mat[2]!=mat2[2])
    {
        printf(" 猴子从%d 把箱子推到了%d\n",mat[2],mat2[2]);return;
    }
    if(mat[0]!=mat2[0])
    {
        printf(" 猴子从%d 移动到了%d\n",mat[0],mat2[0]);return;
    }
}
void addadd(int i)
{
    //printf("next_state=%d\n",next_state);
    if( state_set[next_state]==0)
    {
        state_set[next_state]=1;
        state_father[next_state]=i;
    }
}

```

```

}
int main()
{
    state_set[start_state]=1;//将最初的状态加入到综合数据库
    while(state_set[end_state]==0)//一直循环直到找到最终状态为止
    {
        for(int i=0;i<99999;i++)// 遍历综合数据库中的每一个状态，
            // 并且用规则库之中的状态产生新的状态
        {
            if(state_set[i]==1)
            {
                next_state=0;
                get_value(i);//将状态进行解码
                //IF (x, y, z, 0, 0) THEN (w, y, z, 0, 0)
                //下面利用规则集中的第一条规则进行状态扩展，后面的同理
                if(mat[3]==0&&mat[4]==0)
                {
                    next_state=1*10000+i%10000;
                    addadd(i);
                    next_state=2*10000+i%10000;
                    addadd(i);
                    next_state=3*10000+i%10000;
                    addadd(i);
                }
                //IF (x, y, x, 0, 0) THEN (z, y, z, 0, 0)
                if(mat[3]==0&&mat[4]==0 &&mat[0]==mat[2])
                {
                    Next_state=mat[1]*1000+1*100+1*10000;
                    addadd(i);
                    next_state=mat[1]*1000+2*100+2*10000;
                    addadd(i);
                    next_state=mat[1]*1000+3*100+3*10000;
                    addadd(i);
                }

                //IF (x, y, x, 0, 0) THEN (x, y, x, 1, 0)
                if(mat[3]==0&&mat[4]==0 &&mat[0]==mat[2])
                {
                    next_state=(i/100)*100+10;
                    addadd(i);
                }
            }
        }
    }
}

```

```

    }
    //IF (x, y, x, 1, 0) THEN (x, y, x, 0, 0)
    if(mat[3]==1&&mat[4]==0 &&mat[0]==mat[2])
    {
        next_state=(i/100)*100;
        addadd(i);
    }
    //: IF (x, x, x, 1, 0) THEN (x, x, x, 1, 1)
    if(mat[0]==mat[1]&&mat[1]==mat[2]&& mat[3]==1&&mat[4]==0)
    {
        next_state=i+1;
        addadd(i);
    }
    //当前状态已经扩展完成了，我们使用 2 来防止以后扩展该状态
    state_set[i]=2;
}
}

//打印状态序列，利用 state_father 中存储的信息，从终点逆序找到最初开始的节点。
//在打印正确结果的同时，我们也将信息存储于在 state_son 之中，方便我们接下来顺序打印操作序列
printf("%d<-",end_state);
int father=state_father[end_state];
state_son[father]=end_state;
while(father!=start_state){
    printf("%d<-",father);
    state_son[state_father[father]]=father;
    father=state_father[father];
}
printf("%d\n",start_state);
//下面将猴子需要进行的操作逐行打印出来
int index=start_state;
while(index!=end_state)
{
    print(index,state_son[index]);//打印从当前状态到下一个状态需要进行的操作
    index=state_son[index];
}
return 0;
}

```