模仿学习做证明题

实验内容：运用模仿学习算法，让计算机在人的指导下学习证明奇数X奇数=奇数、偶数X偶数=偶数，然后自己自动证明奇数X偶数=偶数。

实验编号：17

实验开始日期：2016年8月7日

实验成功日期：2016年8月8日

编程语言：C++

编程环境：Visual Studio 2015

讨论帖：<http://tieba.baidu.com/p/4804596283>

关于模仿学习算法的介绍，见纪录片：<http://tieba.baidu.com/p/4760009643>

相关实验：

【模仿学习】计算机学习求导f(x)=x^a

　　　　　　http://tieba.baidu.com/p/4409768742

【模仿学习】让计算机通过“模仿学习”学会解一类小学推理题

http://tieba.baidu.com/p/4485754083

【定理证明】自编程证明“正弦平方加余弦平方等于一”

http://tieba.baidu.com/p/4561469152

使用方法：

1.打开程序，看到一行文字：现在开始证明公式：奇数\*奇数=奇数。这时用户需要向程序输入证明过程。

用户可以输入以下六种命令：

用代数式表示左边第一项

用代数式表示左边第二项

相乘

等号右边除以二

余零则为偶数

余一则为奇数

我输入命令：用代数式表示左边第一项

程序提示：输入一个代数式来表示左边第一项（格式如：2X 或 2X+1）

根据数学常识，奇数被表示为2乘以一个整数加1的形式，于是我就输入2m+1表示等号左边第一个奇数。

接着，我输入命令：用代数式表示左边第二项。

跟刚才一样，这次我用2n+1表示左边第二个奇数。

输入命令：相乘。

程序对(2m+1)和(2n+1)做乘法运算，显示：经计算，等号右边等于4mn+2m+2n+1。

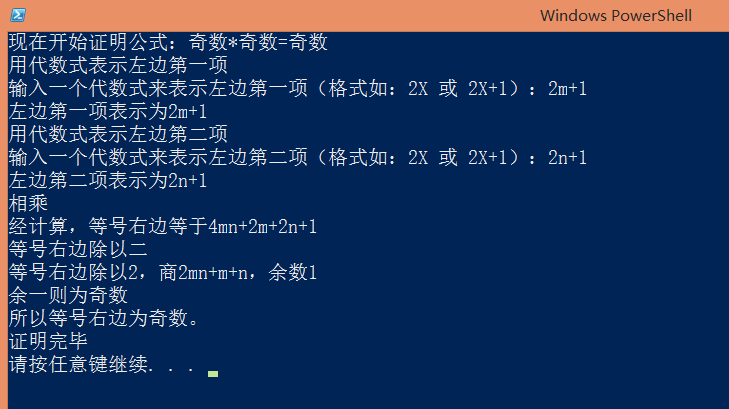
输入命令：等号右边除以二

程序对4mn+2m+2n+1做“除以二并求余数”运算，显示：等号右边除以2，商2mn+m+n，余数1。

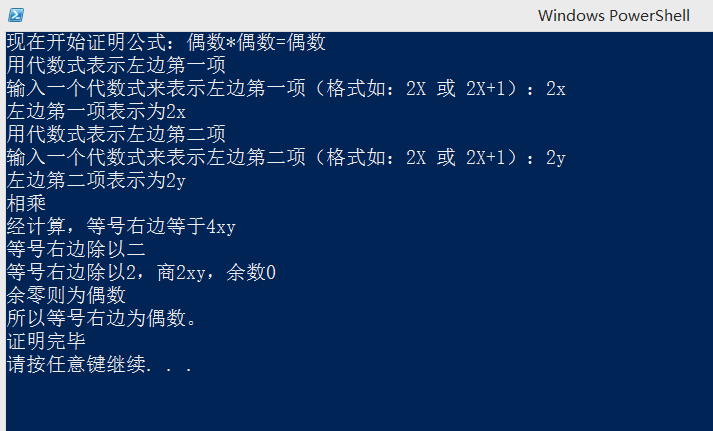
现在我们需告诉程序，一个整数除以二，余数为1则是奇数，没有余数则是偶数。输入命令：余一则为奇数。

程序显示：所以等号右边为奇数。

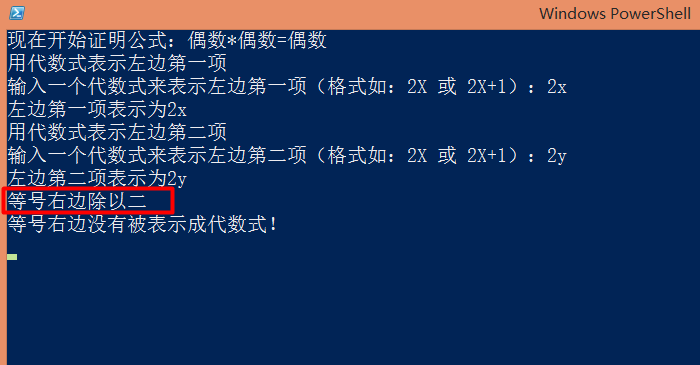
此时“奇数X奇数=奇数”证明完毕。



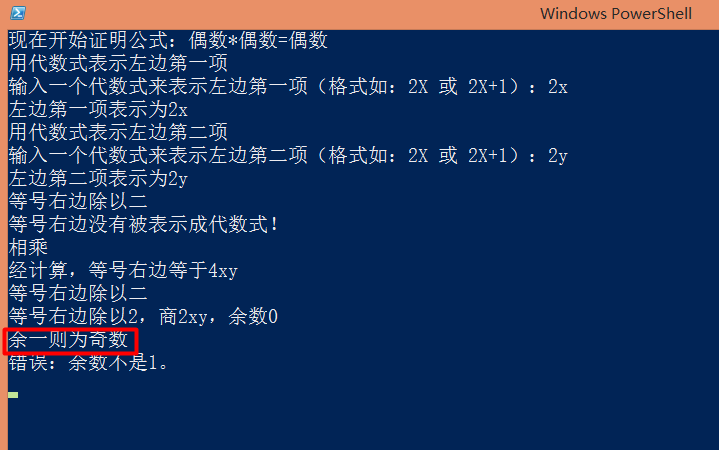
2.按下任意键，程序清屏，开始进行第二个公式：偶数X偶数=偶数 的证明。方法跟刚才类似，不再赘述。



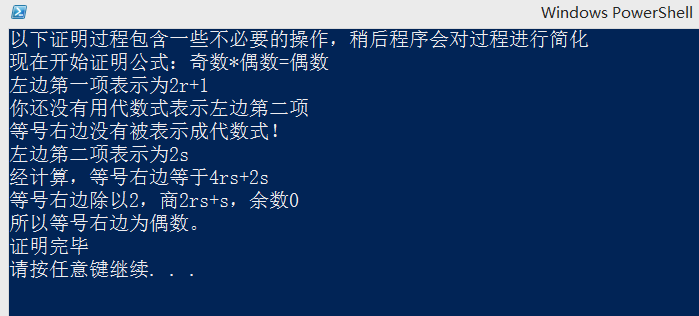
注意：用户输入指令顺序颠倒，或者输入了错误的指令，都会导致程序不能正确地学习定理证明方法。为了避免错误的发生，我引入了“报错机制”。例如我在没有对等号左边两项做乘法的情况下，直接让程序对等号右边做“除以2”运算（此时等号右边是未知的，不能进行任何运算），程序就会输出错误信息。



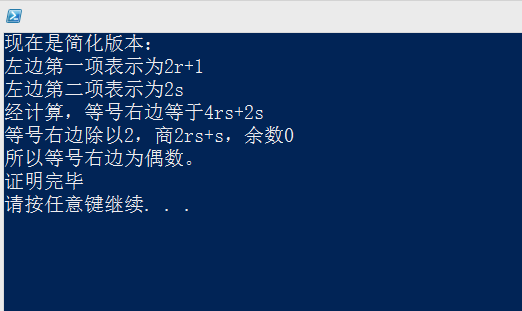
与当前问题无关的指令也是不被接受的，例如除以二以后余数为0，而我告诉程序“余一则为奇数”，程序也会报错：余数不是1 。



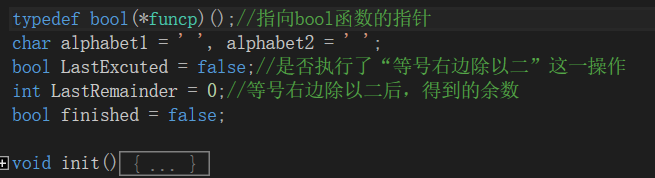
3.在学习了“奇数X奇数=奇数”“偶数X偶数=偶数”的证明过程以后，程序开始尝试套用学习到的方法，来证明“奇数X偶数=偶数”这个公式。这张图显示了程序的尝试过程：



4．可以看出刚才的证明过程包含了一些多余的操作。程序每进行一次操作，都会记录下这个操作是成功了还是失败了（出错或者没有任何效果，则被判定为失败）。成功的操作被保留下来，失败的则被剔除。现在，让我们来看一下简化后的证明过程。



先发数据结构定义，稍后讲解原理。



typedef bool(\*funcp)();//指向bool函数的指针

class item //运算项

{

public:

int coef=1;//系数

bool var1=false;

bool var2 = false;

【构造函数略】

string tostring()

{

string result;

if (var1 || var2)

{

if (coef > 1) result += int2str(coef);

}

else

{

if(coef!=0) result += int2str(coef);

}

if (var1) result += alphabet1;

if (var2) result += alphabet2;

return result;

}

void display()

{

cout << this->tostring();

}

item div2()

{

item result(coef / 2, var1, var2);

return result;

}

};

class expression //表达式

{

public:

vector<item> items; //表达式所包含的运算项，每一项之间用加号连接

【构造函数略】

bool available() //是否有效

{

return (items.size()>0);//一个不包含运算项的表达式是无效的

}

void add(item it) //添加运算项

{

this->items.push\_back(it);

}

string tostring() 【用字符串表示表达式，此处省略】

void display()

{

cout << this->tostring();

}

int mod2()

{

int remainder = 0;//余数

if (items.size() == 0) return 0;

for (int i = 0; i < items.size(); i++)

{

remainder += items[i].coef % 2;

}

return remainder;

}

expression div2()

{

if (items.size() == 0) return \*this;

else

{

expression result = \*this;

for (int i = 0; i < result.items.size(); i++)

{

result.items[i].coef /= 2;

}

return result;

}

}

};

item ItemMul(item it1, item it2) //乘法

{

item result;

result.coef = it1.coef\*it2.coef;

result.var1 = it1.var1 || it2.var1;

result.var2 = it1.var2 || it2.var2;

return result;

}

expression ExprMul(expression ex1, expression ex2) //乘法

{

expression result;

for (int i = 0; i < ex1.items.size(); i++)

for (int j = 0; j < ex2.items.size(); j++)

result.items.push\_back(ItemMul(ex1.items[i], ex2.items[j]));

return result;

}

原理：

本程序用到了模仿学习算法。程序在运行的时候不仅记录下用户的操作，还记录了当时的环境条件——等号左边第一个变量的奇偶性、等号左边第二个变量的奇偶性、等号右边是已知的还是未知的、等号右边是否已经除以二、余数是0还是1。其中前两个条件属于题设套件，跟题目内容有关；后面的都是运行状态条件，跟用户的操作有关。

程序在进行“奇数X偶数=偶数”的自动证明时，会用上记录过的所有命令。当然，程序不是无原则地盲目尝试。程序在运行一个命令时，会调取出用户键入此命令时的所有条件（题设条件与运行状态条件），与现在的条件相匹配，如果完全一致则执行当前命令。下面给出条件判断函数的代码。

//题设条件判断

bool ProblemOdd0()//等号左边第一个变量是奇数

{

return CurrentFormula.odd[0];

}

bool ProblemOdd1()//等号左边第二个变量是奇数

{

return CurrentFormula.odd[1];

}

bool ProblemOdd2()

{

return CurrentFormula.odd[2];

}

bool ProblemEven0()//等号左边第一个变量是偶数

{

return !CurrentFormula.odd[0];

}

bool ProblemEven1()//等号左边第二个变量是偶数

{

return !CurrentFormula.odd[1];

}

bool ProblemEven2()

{

return !CurrentFormula.odd[2];

}

funcp ProblemOdd[3] = { ProblemOdd0,ProblemOdd1,ProblemOdd2 };

funcp ProblemEven[3] = { ProblemEven0,ProblemEven1,ProblemEven2 };

//运行状态条件判断

bool ResultRightOdd() //公式等号右边（运算结果）为奇数

{

if (!CurrentFormula.exprs[2].available())

{

//cout << "假：公式等号右边（运算结果）为奇数\n";

return false;

}

else

{

//if (CurrentFormula.exprs[2].mod2() == 1&&LastExcuted) cout << "真：";

//else cout << "假：";

//cout << "公式等号右边（运算结果）为奇数\n";

return CurrentFormula.exprs[2].mod2() == 1 && LastExcuted;

}

}

bool Remainder0()

{

//cout << ((LastExcuted && (LastRemainder == 0)) ? "真" : "假") << "：余数为零\n";

return LastExcuted&&(LastRemainder == 0);

}

bool Remainder1()

{

//cout << ((LastExcuted && (LastRemainder == 1)) ? "真" : "假") << "：余数为一\n";

return LastExcuted&&(LastRemainder == 1);

}

我在程序里面定义了两个平行全局数组变量（vector容器）：

vector<funcp> commands;//用户输入的命令

vector<vector<funcp>> conditions;//每个指令的执行都是有一定前提条件的（前提条件不止一个）

其中commands用来记录命令，conditions用于记录执行某条命令时的条件。注意conditions是一个二维数组，内层的vector<funcp>用于记录多个指向bool函数的指针。我来解释一下：程序每次执行一个命令的时候，会连带执行所有的条件判断函数（就是我刚才列举的那些），如果返回值为true，则记录之。以后程序自动完成定理证明的时候，每执行一个命令，都会先判断：当时为true的条件，现在仍然为true吗？如果有一个false，则表明当前的条件与原先的条件不符，不执行该命令。

判断的代码如下：

bool ConditionsAllTrue(int index) //判断：所有必须的条件都符合（当前条件==原先条件）

{

if (conditions[index].size() == 0)

return true;

else

{

for (int i = 0; i < conditions[index].size(); i++)

{

if (conditions[index][i]() == false)

{

//cout << conditions[index][i] << "不为真\n";

return false;

}

}

}

return true;

}

接下来的几个函数，体现了本试验程序的“记录机制”：

int GetCommandIndex(funcp cmd) //获取命令下标

{

if (commands.size() == 0) return -1;

else

{

for (int i = 0; i < commands.size(); i++)

{

if (commands[i] == cmd) return i;

}

}

return -1;

}

//记录命令

bool RecordCommand(int VarIndex,funcp cmd)

{

bool exist = false;

if (commands.size() > 0)

{

for (int i = 0; i < commands.size(); i++)

{

if (commands[i] == cmd)

{

exist = true; return false;

}

}

}

if (!exist)

{

commands.push\_back(cmd);

conditions.push\_back(GetTrueConditions(VarIndex));

return true;

}

}

//命令解释，在执行命令前先记录

bool interpret(string command)

{

funcp cmd;//要执行的命令

bool cmdSucceed = false;//命令是否执行成功

int VarIndex = -1;

if (command == "用代数式表示左边第一项") cmd=&Var0Represent;

else if (command == "用代数式表示左边第二项") cmd = &Var1Represent;

else if (command == "相乘") cmd = &FormulaMultiply;

else if (command == "等号右边除以二") cmd = &Mod2;

else if (command == "余一则为奇数") cmd = &Remainder1Odd;

else if (command == "余零则为偶数") cmd = &Remainder0Even;

else cmd = 0;

if (cmd)//如果指向bool型函数的指针不为零

{

bool HasRecorded = false;

if(GetCommandIndex(cmd)==-1 && //如果这个函数没有被记录过

cmd != &Var0Represent&&cmd != Var1Represent)

HasRecorded=RecordCommand(-1, cmd);//先记录

cmdSucceed = cmd();//再执行函数

if (cmdSucceed==false&& //如果执行失败

HasRecorded)//并且刚才记录了

{

commands.pop\_back();//就把刚才记录过的命令“吐”出来

conditions.pop\_back();

}

}

return cmdSucceed;

}

总结：

编写“模仿学习”的人工智能程序，需要注意以下几点：

1. 程序不仅要记录用户的操作，还要记录下当时的环境条件。
2. 当时的环境条件，可以用一个bool函数来表示。Bool函数的返回true表示条件符合，返回false表示不符合。
3. 执行某个命令时，应当用一个或一组变量来存储当时返回true的“环境条件判断函数”，我建议用指向bool函数的指针来存储。
4. 用函数指针存储用户调用过的函数。建议把命令函数的返回值设为bool，表示执行成功/失败。如果函数还要返回其他类型的值，可以用“传递实参”的方法进行。
5. 学习完毕以后，程序自主进行某项智能活动时，会调用前面记录的所有命令。调用时应检查是否符合当时的环境条件。检查方法如下：程序调出与该命令相关的所有bool函数指针，逐个执行，如果有一个返回false，则判定条件不符合，不执行此函数。