鲁宾逊归结原理报告

--- 李晓晨 21217010

目录

1 基本内容

本实验环境为：Myeclipse8.5 + java

输入文件为“clauseset.txt”，在该文件中输入子句集中的所有子句，每个子句集用“---”断开。对于子句的输入，用“-”表示补运算，用“#”表示析取运算。

输出文件为“result.txt”，里面包含每个子句集的归结过程

2 利用水平浸透法的鲁滨逊归结原理

2.1 水平浸透法基本原理

鲁滨逊归结原理是通过对子句集中的子句做多次的归结产生空子句，从而证明字句集的不可满足性，从而结果得以证明。

其基本思想：首先把欲证明的问题的结论否定，并加入字句集，得到一个扩充的子句集。然后设法检查该子句集中是否含有空子句，若没有空子句，则继续使用归结法，直到产生空子句为止。

归结的一般过程是水平浸透法，即对子句集中遍历每个子句，对每个子句遍历每个谓词公式，进行归结。在第 n 次归结时，生成 n+1 级子句集，利用 1…n 的子句集对 n+1 级子句集进行归结，直到新生成的子句集为 NIL（归结成功）或不能继续归结（归结失败）。

2.2 程序结构说明

本程序使用 Java 编写，分为 5 个包。

org.oscar.ai.lxc.data 包为基本的数据结构，其中 AIPrediFormula.java 用于存储一个原子谓词公式；AIClause.java 用于存储一个子句，即原子谓词公式的析取；AIClauseSet.java 用于存储子句集，即鲁滨逊归结原理用于归结的对象；AISubstitution.java 用于存储一对差异集，进行最一般合一的替换。

org.oacar.xc.tools 包为工具类，其中 OFileTool.java 是一般的文件读写操作。

org.oscar.ai.lxc.unifier 包是最一般合一包，其中 AISimpleUnifier.java 进行最一般合一操作。

org.oscar.ai.lxc.robinson 包是鲁滨逊归结原理包，其中 Resolution.java 是使用一般归结方法进行的鲁滨逊归结原理；DeleteSetResolution.java 是增加的删除策略的鲁滨逊归结原理。

org.oscar.ai.lxc.main 包是主函数，其中 AllMain.java 用于启动整个程序。

2.3 水平浸透法伪代码及说明

归结的一般过程在 Resolution.java 中实现，伪代码如下：

```
public void ResolutionClauseSet(AIClauseSet clauseSet){
        初始化
        newClauseSet = firstResolution(origClauseSet);
        while(isResolution){
            生成新的子句集
            newClauseSet = nResolution(origClauseSet, tmpClauseSet);
            newClauseSet.rmSameClause();
            if (newClauseSet == null){
                System.out.println("不能继续归结，归结失败");
            } else if (newClauseSet.getSize() == 0){
                System.out.println("出现NIL，归结成功");
            }
        }
}
```

首先进行初始化，进行第一次归结操作 firstResolution();第一次归结操作为对原始子句集中 C1 与其余的 C2、C3…进行归结，C2 与其余的 C3、C4…进行归结，最终生成 1 级子句集 newClauseSet。

然后进行 while 循环，进行余下的归结操作，每一次都是把 n-1 级的所有子句添加到原始子句集中，生成新的子句集，利用这个子句集中的所有子句，与 n 级子句进行归结，生成 n+1 级子句集。这些归结调用 nResolution()函数。如果本次没有进行归结，则 n+1 级子句集为 null，表示归结失败；如果本次成功的进行了归结，且生成的新的子句集大小为 0（表示，所有的归结都是互补文字对的归结，即生成了 NIL），则归结成功。如果不是这两种情况，

则进行下一次循环。

下面是第一次归结的伪代码

```
private AIClauseSet firstResolution(AIClauseSet clauseSet){
    AIClauseSet newClauseSet = new AIClauseSet();
    for (int i = 0; i < clauseSet.getSize(); ++i){
        for (int j = i+1; j < clauseSet.getSize(); ++j){
            对每两个子句进行归结尝试ResolutionTwoClause()
            如果归结成功，则把归结后的结果添加到newClauseSet中
        }
    }
    return;
}
```

第二次以后的归结伪代码如下

```
private AIClauseSet nResolution(AIClauseSet n_1clauseSet, AIClauseSet nclauseSet){
    AIClauseSet newClauseSet = new AIClauseSet();
    for (int i = 0; i < n_1clauseSet.getSize(); ++i){
        for (int j = 0; j < nclauseSet.getSize(); ++j){
            对每两个子句进行归结尝试ResolutionTwoClause()
            如果归结成功，则把归结后的结果添加到newClauseSet中
        }
    }
    return;
}
```

第一次归结和 n>=2 的归结的不同之处在于对子句集操作的范围不同。

在每次归结中，调用 ResolutionTwoClause()对两个子句进行实际的归结操作，伪代码如下：

```
private boolean ResolutionTwoClause(AIClause aic1, AIClause aic2, AIClause tmpClause){
    把两个子句aic1和aic2放到一起形成tmpClause
    for (int i = 0; i < tmpClause.getSize(); ++i){
        for (int j = i+1; j < tmpClause.getSize(); ++j){
            if 两个谓词公式互补
                归结（同时删除）这两个谓词公式
            } else if 两个谓词公式完全相等
                直接合并成为一个谓词公式
            } else if 适合进行归结
                对两个谓词公式调用最一般合一操作，进行归结（同时删除）
                利用得到的差异集，对子句中的其它谓词公式进行替换
            }
        }
    }
    return;
```

}

利用上述几步操作，完成了鲁滨逊归结原理的一般过程。

3 利用删除策略的鲁滨逊归结原理

一般的归结过程是不断的寻找可归结的子句，生成新的子句集，子句越多，付出的代价越大。删除策略是在归结过程中，删除无用的子句，以减少寻找范围。删除策略包括三个：

（1）纯文字删除法：如果某文字 L 在子句集中不存在可与之互补的文字-L，则称改文字为纯文字，可以删除这个子句。

（2）重言式删除法：如果一个子句中同时包含互补文字对，则称该子句为重言式，可以删除这个子句。

（3）包孕删除法：如果存在一个替换，使得两个子句具有包含关系，则可以把子句集中包孕的子句删除。

这里实现了纯文字删除法和重言式删除法，以减少寻找范围。具体的过程在 DeleteSetResolution.java 中实现。伪代码如下：

```
public void ResolutionClauseSet(AIClauseSet clauseSet){
    初始化
    doDelete();
    newClauseSet = firstResolution(origClauseSet);
    while(isResolution){
        生成新的子句集
        doDelete();
        newClauseSet = nResolution(origClauseSet, tmpClauseSet);
        newClauseSet.rmSameClause();
        if (newClauseSet == null){
            System.out.println("不能继续归结，归结失败");
        } else if (newClauseSet.getSize() == 0){
            System.out.println("出现NIL，归结成功");
        }
    }
}

private void doDelete(AIClauseSet curClauseSet){
    deletePureWord(curClauseSet);
    deleteTautology(curClauseSet);
}
```

与一般的方法不同指出在于，在得到原始子句集或生成新的子句集时，调用 doDelete() 函数，删除子句集中的纯文字和重言式。纯文字和重言式的寻找为对子句集的遍历过程。如果一个谓词公式在子句集中找不到候选的进行归结的公式，则该谓词公式为纯文字，删除该

谓词公式所属的子句；如果对于一个子句，里面包含了互补文字，则认为该子句为重言式，删除该子句。

4 结果截图

```
原始子句集为：
P(A)#P(B)#P(C)
-P(A)#P(B)#P(C)
-P(B)#P(C)
-P(C)

1级归结式中子句为：
P(B)
P(C)
P(C)
-P(C)

删除子句-P(B)#P(B)#P(C)存在重言式
删除子句-P(C)#P(C)存在重言式
2级归结式中子句为：
P(B)

3级归结式中子句为：
P(B)

4级归结式中子句为：
出现NIL，归结成功

---
```

原始子句集为：
-A(x,y)#-B(y)#C(f(x))
-A(x,y)#-B(y)#D(x,f(x))
-C(z)
A(a,b)
B(b)

删除子句-A(x,y)#-B(y)#D(x,f(x))存在纯文字
1级归结式中子句为：
-A(a,b)#-B(b)
-B(b)#C(f(a))
-A(a,b)#C(f(a))

2级归结式中子句为：
-B(b)
-A(a,b)
C(f(a))

3级归结式中子句为：
出现NIL，归结成功
