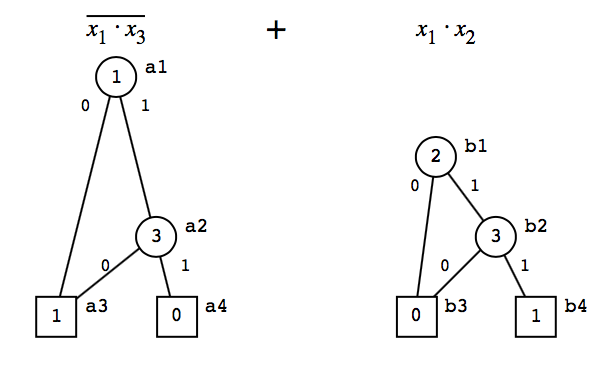
1. (20分) 图1表示¬(*x*1·*x*3) 与*x*2·*x*3进行“或（or）”操作的过程，请给出该运算的**过程图（Evaluation Graph）与最终化简图，并给出化简后的表达式。**



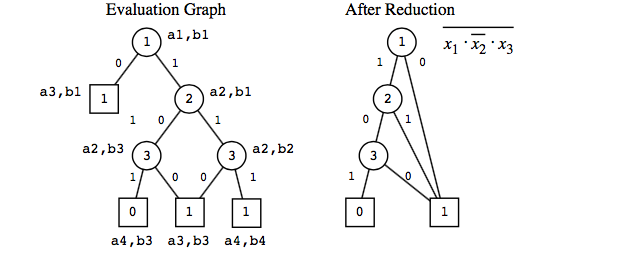


图1

1. (20分)子句（Clause）数据结构设计

2.1 设计一个基于数组的数据结构存储子句数据（clause data）中的文字、子句与公式。（15分）

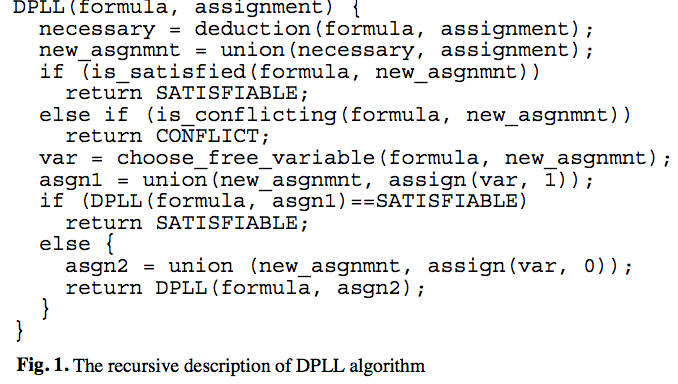
用数组存储子句的文字，对于每个数组设置两个指针，分别为头指针与尾指针，然后对于每个变量v设置4个指针，分别装有句头是 v 的子句，句头是非 v 子句， 句尾是 v 的子句，句尾是非 v 的子句，这样存储的子句，无论子句顺序及其中变量顺序的改变都不会影响所有变量的四个链表中的子句数量。取所有变量v是句头的子句与非v是句头的子句或者v是句尾的子句与非v是句尾的子句合取可以得到公式

2.2 列举出相比于指针（pointer-heavy）数据结构，用数组存储子句的优点。（5分）

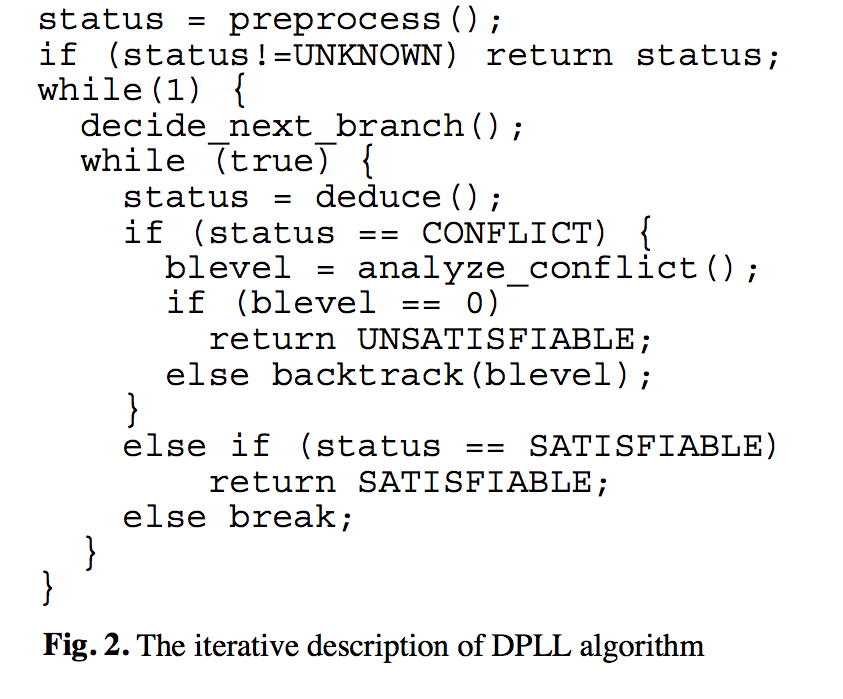
1、数组采用连续内存空间存储，内存利用率更高，访问局部变量速度更快。

2、对于cache命中而言，由于数组在内存上连续存储，相比于链表的不连续存储能够提高cache命中率，从而能够增加计算速度。

3.（10分）简述为什么在DPLL算法的实现中往往采用迭代（图3）实现而不是递归实现（图2）。



**图2 DPLL 算法的递归实现**



**图3 DPLL 算法的迭代实现**

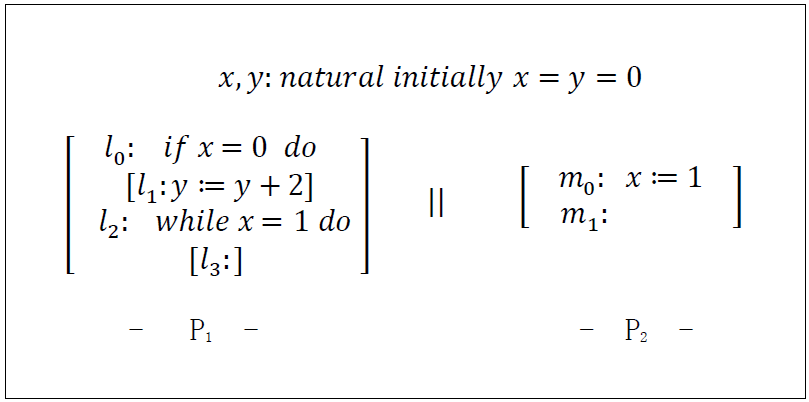
（1）递归速度慢且容易发生溢出，相对于迭代就有很多自身的劣势。

（2）迭代具有非时间顺序回溯（智能回溯）的优势。

4. （15分）基于计数器的BCP算法是一种容易理解与实现的 BCP 算法。假设每个子句（clause）拥有两个计数器（counter），一个用于子句中的值1字面量（literal）的计数，一个用于子句中的值0字面量的计数。每个变量（variable）都有两个列表，其中包含所有子句，其中该变量分别显示为正值和负值。当为变量分配一个值时，包含此字面量的所有子句将更新其计数器。设实例具有*m*个子句（clause）和*n*个变量（variable），并且平均每个子句具有*l*个字面值（literal）。**那么每当给一个变量赋值时，有多少个计数器****（counter）需要更新。简要概述分析过程。**l/n表示每个变量平均在每个子句中出现的次数，然后乘以子句的数量m，所以，有一个变量被赋值的时候，会有平均 ml/n 个计数器需要更新，在回溯的时候，每取消一个变量赋值，也会平均有 ml/n 个计数器的更新。

5. (15分) 将下列程序转换成逻辑公式。

Program:

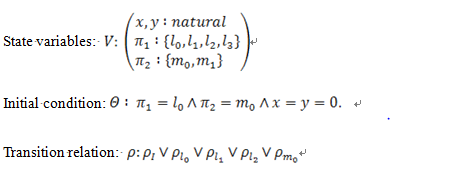


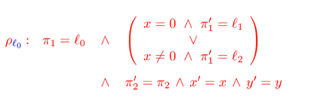
State variables:

Initial condition:

Transition relation:

Write the disjoints and





(For example:)

6. (20分) “天花板与地板”模型（the Alloy Analyzer）

sig Platform {}

*there are “Platform” things*

sig Man {ceiling, floor: Platform}

*each Man has a ceiling and a floor Platform*

pred Above(m, n: Man) {m.floor = n.ceiling }

*Man m is “above” Man n if m's floor is n's ceiling*

fact {all m: Man | some n: Man | Above (n,m)}

*"One Man's Ceiling Is Another Man's Floor"*

**assert BelowToo**

**{all m: Man | some n: Man | Above (m,n)}**

*"One Man's Floor Is Another Man's Ceiling"?*

现在假设有2个Platform实例，2个 Man 实例，找出至少一个不满足断言“BelowToo”的反例，并用图形表示。

