文章编号:1000-5862(2013)03-0268-05

基于遍历序列恢复二叉树的新解法及其证明

化志章
(江西师范大学计算机信息工程学院,江西 南昌　330022)

摘要:提出了一种基于前序和中序遍历序列恢复二叉树的解法,算法以数学公式形式呈现,反映了建树过程中相关数据变化的一般规律,具备数学上的引用透明性,由此能机械获得非递归程序和循环不变式,并进行了正确性证明.通过简单变换,获得了后序+中序、前序+后序恢复二叉树的可信算法.实验效果表明了该解法的有效性.

关键词:状态变迁;二叉树遍历;恢复二叉树;循环不变式
中图分类号:TP 311.1　　**文献标志码**:A

0　引言

二叉树是一种简单的非线性结构,在编译器构造、数据挖掘、人工智能等领域有着广泛应用.二叉树相关算法的形式化开发及证明,对基于非线性结构的可信算法构造有着重要的借鉴意义.遍历是二叉树最基本操作,而基于遍历序列恢复二叉树常被视为遍历的逆过程,并有着与遍历操作类似的特性.构造递归算法很容易,构造非递归算法则较困难.

D. E. Knuth[1] 将基于遍历恢复二叉树视为有趣的算法问题,对其可行性进行了简单解释.研究程序变换的学者则将其作为经典的程序问题进行处理[2],产生较多非递归算法. H. A. Burgdorff 等[3] 给出了时间为 $O(n^2)$ 的非递归算法,Gen-Huey Chen 等[4] 从时间优先和空间优先角度给出 2 个恢复二叉树的算法,其中时间优先算法需要借助哈希存储,而空间优先算法需要 $O(N\log N)$ 时间和 $O(N)$ 空间;A. Andersson 等[5] 对 Gen-Huey Chen 的时间优先算法进行了改进,用栈替代哈希存储,使时间和空间均为 $O(n)$;E. Mäkinen[6] 提出的新算法侧重于减少恢复过程中比较次数;唐自立[7] 较为系统地分析了基于遍历序列恢复二叉树的各种可能,用数学归纳法证明其可行性,并在文献[8]给出了基于前序和后序序列恢复无 1 度结点二叉树的实现算法.但上述研究提出的算法均由分析获得,并未对算法本身的正确性进行证明,结果难以令人信服. N. Arora 等[9] 系统性地研究了现有的恢复二叉树算法,发现有些算法存在错误,并进行了修正.遗憾的是他同样未对所提算法正确性进行证明.有别于传统的命令式算法形式,S. C. Mu 等[2] 使用 Bird-Meertens 演算给出了函数式风格的前序+中序恢复树算法.开发过程中需要引入大量的专业数学知识,属于数学家的开发.

基于状态变迁的思想构造了二叉树后序遍历非递归算法[10],并通过参数变换获得二叉树前序遍历、中序遍历以及 K 叉树前序遍历和后序遍历的非递归算法,算法以数学公式方式呈现,比现有算法简洁.本文将状态变迁思想应用于基于遍历序列恢复二叉树,同样收到了较好效果.

1　基于状态变迁思想的可信算法构造

基于非线性结构的数据应用有较多隐含信息需要考虑,如二叉树结点 t 包含数据域和左右孩子等信息.在设计非递归的基于遍历恢复二叉树算法时,t 的数据域已知、左孩子已知或右孩子已知等代表着不同的含义,必须要明确区分.基于上述考虑,根据问题求解时结点使用的不同情形,将结点分作不同状态.这些状态的变迁,是问题求解从初始情形向最终情形变迁的基本操作.进一步拓展,以结点状态及影响结点状态变迁的所有相关数据为基础构造程

收稿日期:2013-01-12
基金项目:江西省教育厅科技课题(GJJ09142)资助项目.
作者简介:化志章(1972-),男,安徽萧县人,副教授,硕士,主要从事算法形式推导和软件验证等方面的研究.

序状态. 程序状态反映了求解过程中任一中间时刻所有数据的取值,初始程序状态体现了问题求解的已知条件,终止状态则包含了问题求解的最终结果. 算法则可看作是从初始状态向终止状态变迁的有限自动机,算法开发的关键就是发现状态变迁函数(State Transformation Function,STF). STF 是用数学公式描述的算法,由 STF 可机械产生抽象程序和该程序的循环不变式.

由于算法开发过程中会使用非形式化的领域知识,算法正确性需要证明. 证明步骤可分作 2 步走:(ⅰ)证明 STF 满足问题需求;(ⅱ)证明所得程序满足 STF. 前者常采用数学归纳法,后者则依照 Dijkstra-Gries[1-2] 的最弱前置谓词法实施. 由于 STF 涉及求解问题的初始条件和最终结果,因此可作为抽象程序的形式规约(即前置断言和后置断言). 众所周知,使用最弱前置谓词法的最大难点在于构造合适的循环不变式,而本文的循环不变式可从 STF 机械变换得到. 因此,第(ⅱ)步证明过程变得非常容易,常省略.

2　构造前序 + 中序恢复二叉树的可信算法

本小节将首先介绍将使用的一些记号; 继而定义建树过程中二叉树的结点状态,并基于遍历定义获得结点状态变迁规则; 之后简单变换,获得体现建树规则的状态变迁函数,进而机械获得非递归抽象程序和循环不变式,并证明其正确性.

2.1　记号说明

序列是特定类型元素的线性排列, $\mathbf{I}$ 表示空序列; $\uparrow$ 是序列的连接运算符,如 $[a,b]\uparrow[c,d]=[a,b,c,d]$; 对任意序列 S,有 $S\uparrow\mathbf{I}=\mathbf{I}\uparrow S=S$. 对序列 S,$S[0]$ 表示序列的首个元素,$S[1\cdots]$ 表示 S 中由第 2 至序列末尾元素组成的子序列.

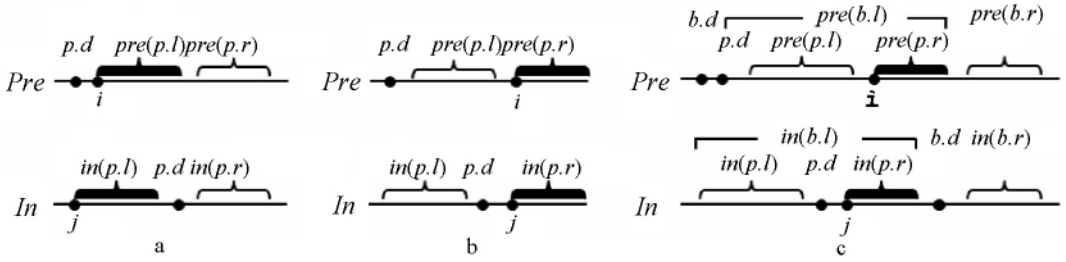
Pre 和 In 是给定的前序和中序序列,Pre^i、In^j 表示当前 Pre、In 的位置分别为 i、j; $pre(t)$、$in(t)$ 分别表示 t 的前序和中序遍历序列,用 % 表示空树; 用 $t^{<t.d,t.l,t.r>}$ 的形式表示非空结点 t,其中 $t.d$、$t.l$、$t.r$ 分别表示 t 数据域和左右孩子. 另外,用"▲"表示信息已知,用"?"表示信息未知,如 $t^{<▲,q,?>}$ 表示 $t.d$ 已知,$t.l=q$,$t.r$ 未知.

2.2　构造可信算法

定义 1　对非空结点 t,根据建树过程中 $t.d$、$t.l$ 和 $t.r$ 的知晓顺序定义 0、1、2 共 3 种状态. 0 态表示 $t.d$ 已知,$t.l$ 和 $t.r$ 未知; 1 态表示 $t.d$ 已知和 $t.l$、$t.r$ 未知; 2 态表示 $t.d$、$t.l$ 和 $t.r$ 均已获知.

定义 2　将结点 t 从 0 态转变到 1 态,或从 1 态转变到 2 态,称作结点状态的一次变迁.

为方便推导,本文约定对当前结点 p,当其处于 0 态时,i、j 分别位于 $pre(p.l)$ 和 $in(p.l)$ 的起始位置处; 当其处于 1 态时,i、j 分别位于 $pre(p.r)$ 和 $in(p.r)$ 的起始位置处. 图 1 是 p 在 0 态或 1 态时 i、j 所处位置的示意图. 其中根据 $p.r$ 是否位于遍历序列的尾部将 1 态的位置情况分成图 1(b) 和图(c) 2 种.



(a) p 处于 0 态; (b) p 处于 1 态且 $p.r$ 在遍历序列尾部; (c) p 处于 1 态且 $p.r$ 不在序列尾部.

图 1　当前结点 p 在 0 态或 1 态时 i、j 所处位置的示意图

由图 1(a) 易知,若 $p.l$ 为空,则有 $In[j]=p.d$; 否则,$p.l$ 指向新结点 $q^{<Pre[i],?,?>}$. 由于 $pre(p)$ 和 $in(p)$ 与 p 的树形间能够相互确定,故上述推理是充要的,即有如下定理:

定理 1　p 处于 0 态 $\wedge$ (Pre^i,In^j) $\wedge$ $p.d=In[j]\Rightarrow p.l=\%$; p 处于 0 态 $\wedge$ (Pre^i,In^j) $\wedge$ $p.d\neq In[j]\Rightarrow p.l=q^{<Pre[i],?,?>}$.

当 p 处于 1 态时,若 $p.r$ 为空,则可能有 $j\geqslant n$(见图 1(b)) 或 $j<n\wedge In[j]=b.d$(见图 1(c)),其中 b 是距离 p 最近且为 1 态的祖先; 否则,$p.r=q^{<Pre[i],?,?>}$,即有如下定理:

定理 2　p 处于 1 态 $\wedge$ (Pre^i,In^j) $\wedge$ $(j\geqslant n\vee In[j]=b)\Rightarrow p.r=\%$; p 处于 1 态 $\wedge$ (Pre^i,In^j) $\wedge$ (b 不存在 $\vee$ $In[j]\neq b)\Rightarrow p.r=q^{<Pre[i],?,?>}$,其中,$b$

为距离 p 最近且状态为 1 态的祖先.

初始时,有 $p = root^{<Pre[1],?,?>}$, $i = 1, j = 0$. 之后,建树就是反复运用上述规则进行状态变迁的过程. 当状态变迁过程中产生新结点(即孩子结点)时(如定理1中的 $q^{<Pre[i],?,?>}$),根据树的遍历定义,必须优先新结点的变迁,因此需要使用序列 S 保存尚未变至2态的结点. 若当前结点变迁完毕(即变至2态),则需从 S 中剔除,并取出新结点开始变迁. 当 S 中无待变迁结点时,整个建树过程结束.

为精确描述建树过程,用 $\langle X, S, Pre^i, In^j, tag\rangle$

$$F(X,S,Pre^i,In^j,tag) = \begin{cases} \langle X, [^{<\blacktriangle,\%,?>}] \uparrow S[\cdots], Pre^i, In^{j+1}, 1\rangle, & \text{if } tag = 0 \wedge In[j] = t.d, \\ \langle X, [q^{<Pre[i],?,?>}] \uparrow [^{<\blacktriangle,q,?>}] \uparrow S[\cdots], Pre^{i+1}, In^j, 0\rangle, & \text{if } tag = 0 \wedge In[j] \neq t.d, \\ \langle X \uparrow [^{<\blacktriangle,\blacktriangle,\%>}], S[\cdots], Pre^i, In^{j+1}, 1\rangle, & \text{if } tag = 1 \wedge (j \geqslant n \vee In[j] = S[1].d), \\ \langle X \uparrow [^{<\blacktriangle,\blacktriangle,q>}], [q^{<Pre[i],?,?>}] \uparrow S[\cdots], Pre^{i+1}, In^j, 0\rangle, & \text{if } tag = 1 \wedge (S[\cdots] = \blacksquare \vee In[j] \neq S[1].d), \\ out(root), & \text{if } S = \blacksquare \wedge (j \geqslant n \vee i \geqslant n), \end{cases} \quad (1)$$

令 $t = S[0]$,初始条件: $X = \blacksquare \wedge S = [root^{<Pre[0],?,?>}] \wedge i = 1 \wedge j = 0 \wedge tag = 0 \wedge n \geqslant 1$,公式(1)用前序序列 + 中序序列恢复二叉树的状态变迁公式. 用 Apla 语言描述前序 + 中序建树的非递归算法如下:

```
function btree( char)  F( Pre: list( btree( char) ) , in: list( btree( char) ) )
        /* 注意: F( X,S,Prei,Inj,tag) 刻画的是程序执行期间任意时刻的全部数据信息,
           部分参数将以函数的局部变量形式出现  * /
    var   q: btree( char) ;   X: list( btree( char) ) ;   S: list( btree( char) ) ;   tag: int;
    begin
      root = new( btree* ) ;    root. d = Pre [0] ;   X,S,i,j,tag,: = ■, [root] ,1,0,0;
                do ¬ ( S = ■ ∧ j ≥ n ∧ i ≥ n) → t = S [0] ;
        if ( tag = 0 ∧ In [j] = t. d) →t. l = % ;   X,S,i,j,tag = X,S,i,j + 1,1;
         ■( tag = 0 ∧ In [j] ≠t. d) → q = new( btree* ) ; q. d = Pre [i] ; t. l = q;
           X,S,i,j,tag = X,q ↑ S,i + 1,j,0;
         ■( tag = 1 ∧( j ≥ n ∨ In [j] = S [1] . d) ) → t. r = % ; X,S,i,j,tag = X ↑ t,
           S [1 ⋯] ,i,j + 1,1;
         ■( tag = 1 ∧ ( S [1 ⋯] = ■ ∨ In [j] ≠S [1] . d) ) →
              q = new( btree* ) ; q. d = Pre [i] ; t. r = q; X,S,i,j,tag = X ↑ t,q ↑ S,i + 1,j,0;
        fi;
      od;
      return root;
    end.
```

用 Apla 语言描述公式(1),可近乎机械地获得非递归抽象程序. Apla 是一种抽象程序设计语言,有强大的抽象数据类型描述机制. 借助 Apla → C + + 自动转换工具和可重用部件库支持,可由 Apla 程序生成 C + + 程序,这里略. 将公式(1)书写成逻辑形式,即可得到

ρ: $X = \blacksquare \wedge S = [root^{<Pre[0],?,?>}] \wedge i = 1 \wedge j = 0 \vee S = [^{<\blacktriangle,\%,?>}] \uparrow S[\cdots] \wedge tag = 0 \wedge In[j] = t.d \vee S = [q^{<Pre[i],?,?>}] \uparrow [^{<\blacktriangle,q,?>}] \uparrow S[\cdots] \wedge tag = 0 \wedge In[j] \neq t.d \vee X = X \uparrow [^{<\blacktriangle,\blacktriangle,\%>}] \wedge S = S[\cdots] \wedge tag = 1 \wedge (j \geqslant n \vee In[j] = S[1].d) \vee X = X \uparrow [^{<\blacktriangle,\blacktriangle,q>}] \wedge S = [q^{<Pre[i],?,?>}] \uparrow S[\cdots] \wedge tag = 1 \wedge (S[\cdots] = \blacksquare \vee In[j] \neq S[1].d)$.

该程序的循环不变式 ρ,它反映了在创回复二叉树过程中任一时刻的程序状态.

2.3　算法的正确性证明

该算法的正确性证明分 2 步:(ⅰ)证明公式(1)能正确建树;(ⅱ)证明由公式(1)机械产生的公式 ρ,是上述 Apla 程序的循环不变式,即确保所得程序满足公式(1).

(ⅰ)证明公式(1)能正确建树(用数学归纳法)

① 当 $n=1$ 时,公式(1)能正确建树. $n=1$ 时有 $Pre[0]=In[0]$,且初始时,有 $root=q^{<Pre[0],?,?>}\wedge X=\Lambda\wedge S=[root]\wedge \mathrm{tag}=0\wedge i=1\wedge j=0$,根据公式(1),有

$F(X,[root^{<Pre[0],?,?>}]\uparrow\Lambda,Pre^1,In^0,0)=F(\Lambda,[root^{<Pre[0],\%,?>}]\uparrow\Lambda,pre^1,in^1,1)=F([root^{<Pre[0],\%,\%>}],\Lambda,Pre^1,In^1,0)=root^{<Pre[0],\%,\%>}$;得证.

② 假设当 $n\leqslant k$ 时可以正确地建树,现证明则当 $n=k+1$ 时,公式(1)仍能正确建树. 不失一般性,假定新增叶子 q,使得 $n=k+1$,其中 $p.l=q$,$p.d=a$,$q.d=b$. 令原 Pre 和 In 中,有 $Pre[i]=In[j]=a$,易知插入后,有 $Pre[i]=a\wedge Pre[i+1]=b\wedge In[j]=b\wedge In[j+1]=a$. 现证明基于此序列,$q$ 作为叶子可被插入到 $p.l$ 处.

$F(X,[p^{<a,?,?>}]\uparrow S[1\cdots],Pre^{i+1},In^j,0)=F(X,S,Post^i,In^j,tag)=$

$$\begin{cases}\langle X,[^{<\blacktriangle,?,\%>}]\uparrow S[1\cdots],Post^i,In^{j-1},1\rangle, & \text{if } tag=0\wedge In[j]=t.d,\\ \langle X,[q^{<Post[i],?,?>}]\uparrow[^{<\blacktriangle,?,q>}]\uparrow S[1\cdots],Post^{i-1},In^j,0\rangle, & \text{if } tag=0\wedge In[j]\neq t.d,\\ \langle X\uparrow[^{<\blacktriangle,\%,\blacktriangle>}],S[1\cdots],Post^i,In^{j-1},1\rangle, & \text{if } tag=1\wedge(j<0\vee In[j]=S[1].d),\\ \langle X\uparrow[^{<\blacktriangle,q,\blacktriangle>}],[q^{<Post[i],?,?>}]\uparrow S[1\cdots],Post^{i-1},In^j,0\rangle, & \text{if } tag=1\wedge(S[1\cdots]=\Lambda\vee In[j]\neq S[1].d),\\ out(root), & \text{if } S=\Lambda\wedge(j<0\vee i<0),\end{cases}\tag{2}$$

令 $t=S[0]$,初始条件:$X=\Lambda\wedge S=[root^{<Post[n-1],?,?>}]\wedge i=n-2\wedge j=n-1\wedge tag=0\wedge n\geqslant 1$,公式(2)后序 + 中序建立二叉树的递推公式.

另外,对无 1 度结点的二叉树 t,可通过前序 Pre 和后序 $Post$ 遍历创建. $Pre[0]$ 为 $t.d$,$Post[n-2]$ 为 $t.r.d$,可区分左右子树. 创建 $p.l$ 时,若 $Post[j]=F(X,S,Pre^i,Post^j,tag)=$

$$\begin{cases}\langle X\uparrow[^{<\blacktriangle,\%,\%>}],S[1\cdots],Pre^i,Post^{j+1},1\rangle, & \text{if } tag=0\wedge Post[j]=t.d,\\ \langle X,[q^{<Pre[i],?,?>}]\uparrow[^{<\blacktriangle,q,?>}]\uparrow S[1\cdots],Pre^{i+1},Post^j,0\rangle, & \text{if } tag=0\wedge Post[j]\neq t.d,\\ \langle X\uparrow[^{<\blacktriangle,\%,\%>}],S[1\cdots],Pre^i,Post^{j+1},1\rangle, & \text{if } tag=1\wedge Post[j]=t.d,\\ \langle X,[q^{<Pre[i],?,?>}]\uparrow[^{<\blacktriangle,\blacktriangle,q>}]\uparrow S[1\cdots],Pre^{i+1},Post^j,0\rangle, & \text{if } tag=1\wedge Post[j]\neq t.d,\\ out(root), & \text{if } S=\Lambda\wedge(j\geqslant n\vee i\geqslant n),\end{cases}\tag{3}$$

$F(X,[q^{<b,?,?>}]\uparrow[p^{<a,q,?>}]\uparrow S[1\cdots],Pre^{i+2},In^j,0)=F(X,[q^{<b,\%,?>}]\uparrow[p^{a,q,?}]\uparrow S[1\cdots],Pre^{i+2},In^{j+1},1)=F(X\uparrow[q^{<b,\%,\%>}],[p^{a,q,?}]\uparrow S[1\cdots],Pre^{i+2},In^{j+2},1)$.

由此可知,b 被正确地创建成叶子(即 $q^{<b,\%,\%>}$),且被正确地插入到 $p.l$ 处(即 $p^{<a,q,?>}$). 类似地,易得 b 可以被正确插入 a 的右孩子位置. 得证.

(ⅱ)证明 ρ 是表 1 所示程序的循环不变式(最弱前置谓词法)

证明依照 Dijkstra-Gries 的最弱前置谓词法实施,主要包括 4 个步骤:(a)证明 ρ 在执行循环前为真;(b)证明 ρ 在每次执行循环体后仍为真;(c)证明在循环终止时,$\rho\wedge(S=\Lambda\wedge j\geqslant n\wedge i\geqslant n)\Rightarrow F(Pre,In)$;(d)证明循环必在有限步内终止.

证明过程简单,这里略.

3　方法推广

前序、中序遍历的通式可写作. bLR、LbR,将后序和中序遍历通式倒置,可得 bRL 和 RbL. 因此,从尾至首扫描中序和后序序列,将左右子树建树规则互换,则可得到用中序 + 后序建树的递推公式.

$p.d$,表明 $p.l$ 为空,相应地 $p.r$ 也为空,之后转入 p 双亲的右子树处理;否则 $p.l=q^{<Pre[i],?,?>}$,继续 $p.l$ 左子树的创建. 创建 $p.r$ 时,若有 $Post[j]=p.d$,表明 $p.r$ 处理完毕,之后转入 p 双亲的右子树处理;否则 $p.r=q^{<Pre[i],?,?>}$,继续 $p.r$ 左子树的创建. 由此,容易得到用前序 + 后序建树的递推公式为

令 $t = S[0]$,初始条件:$X = \Pi \wedge S = [root^{<Pre[0],?,?>}] \wedge i = 1 \wedge j = 0 \wedge tag = 0 \wedge n \geqslant 1$,公式(3)前序 + 后序建立二叉树的递推公式(注:树中无 1 度结点).

4 总结和讨论

本文使用状态变迁思想,经过简单推导获得了基于前序 + 后序序列恢复二叉树的非递归算法,算法以数学公式(即状态变迁函数)的形式呈现,该公式反映了建树过程中相关数据变化的一般规律,具备数学上的引用透明性,由此公式能机械获得非递归抽象程序和循环不变式,并进行了正确性证明. 与文献[3-9]的工作相比,本工作不仅开发步骤更为简单,而且用数学函数形式描述的算法也更简洁、精确,正确性也得到了保证. 因此有理由相信,本文的方法更能揭示问题求解的本质规律,结果也更令人信服.

5 参考文献

[1] Knuth D E. The art of computerprogramming volume 1: fundamental algorithms [M]. 3rd Edition. New York: Addison Wesley, 1997.

[2] Mu S C, Bird R S. Rebuilding a tree from its traversals: a case study of program inversion [EB/OL]. [2012-10-11]. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.1166.

[3] Burgdorff H A, Jojodia S, Springsteel F N, et al. Alternative methods for the reconstruction of tree from their traversals [J]. BIT, 1987, 27(2): 134-140.

[4] Gen-Huey Chen, Yu M S, Liu L T. Two algorithms for constructing a binary tree From its traversals [J]. Information Processing Letters, 1988, 28(6): 297-299.

[5] Andersson A, Carlsson S. Construction of a tree from its traversals in optimal time and space [J]. Information Processing Letters, 1990, 34(1): 21-25.

[6] Mäkinen E. Constructing a binary tree efficiently from its traversals [J]. International Journal of Computer Mathematics, 2000(1): 75.

[7] 唐自立. 基于遍历序列的唯一确定树或二叉树的方法 [J]. 小型微型计算机系统, 2001, 22(8): 985-988.

[8] 唐自立. 基于遍历序列的构造严格二叉树的算法 [J]. 苏州大学学报: 自然科学版, 2010, 26(3): 40-43.

[9] Arora N, Tamta V K, Kumar S. Modified non-recursive algorithm for reconstructing a binary tree [J]. International Journal of Computer Applications, 2012, 43(10): 25-28.

[10] 化志章, 揭安全, 杨庆红. 树非递归遍历统一的新解法及其形式证明 [J]. 江西师范大学学报: 自然科学版, 2010, 34(2): 123-127.

[11] Gries D. The science of programming [M]. New York: Springer-Verlag, 1981.

[12] Dijkstra E W, Scholten C S. Predicate calculus and program semantics [M]. New York: Springer-Verlag, 1990.

A New Solution with Proof for Reconstructing a Binary Tree from Its Traversals

HUA Zhi-zhang

(College of Computer Information and Engineering, Jiangxi Normal University, Nanchang Jiangxi 330022, China)

Abstract: A new solution for reconstructing a binary tree based on the pre-order and in-order traversal is proposed. Its algorithm is mathematical formula, which reflecting the data change rule about the reconstructing binary tree process, and with referential transparency on the mathematical. Then, non-recursive abstract program and its loop invariant are obtained mechanically according to the formula, and their correctness are proved formally. And through a simple transformation, the trusted algorithms using post-order + in-order traversal, and pre-order + post-order traversal, are obtained. Experimental results show the effectiveness of this solution.

Key words: state transformation; binary tree traversal; reconstructing tree; loop invariant

(责任编辑: 冉小晓)