Non-recursive Algorithm Derivation and Formal Proof of Binary Tree Traversal Class Problems

Zhengkang Zuo¹, Yue Fang¹, Qing Huang¹*, Yunyan Liao¹, Yuan Wang², and Changjing Wang²*
¹School of Computer Information Engineering, Jiangxi Normal University, Nanchang, 330022, China
²School of Software, Jiangxi Normal University, Nanchang, 330022, China
*Correspondence to Changjing Wang (wcj771006@163.com) and Qing Huang (qh@whu.edu.cn)

Abstract—The development of loop invariants for recursive problems of nonlinear data structures is always difficult problem in formal development. The paper studies the derivation and formal proof of binary tree traversal class non-recursive algorithm. The non-recursive Apla (Abstract Programming Language) algorithms of binary tree traversal class problems and their exact and simple loop invariants are derived. Then, the correctness of the algorithm is proved by Dijkstra-Gries standard proving technique. Finally, the PAR platform is used to automatically generate C++ code, which realizes the complete refinement process from abstract specification to concrete executable program. The results of this study point out the direction for the exploration of the loop invariant of the non-recursive algorithm for recursive problems and have guiding significance for the derivation and formal proof of the algorithm program of the nonlinear data structure.

Keywords—binary tree traversal class problems, PAR method, loop invariant, Dijkstra-Gries standard proving technique, nonlinear data structure

I. INTRODUCTION

The derivation and formal proof of non-recursive algorithm of binary tree is always difficult. In the traversal algorithm, the traversal of binary tree can be recursive or iterative. However, the recursive algorithm code is simple-calculating, long time-consuming and large space-occupying, while the iterative process increases the time and the space overhead. Therefore, we prefer to use efficient non-recursive algorithm to solve the binary tree traversal problem. However, the non-recursive algorithm code is numerous, and the logic relationship is complex and difficult to understand. The proving process is obscure, and the correctness of the proof cannot be guaranteed. Therefore, how to find the loop invariant of binary tree traversal class problems are difficult problems. Therefore, we expect to develop a more effective new algorithm design method to develop and understand problem algorithms. This paper is based on the new definition and the new development strategy of loop invariants, which selects binary tree traversal class problems as research objects. Also, the paper uses the idea of recursive definition to derive and formally prove the non-recursive algorithm of binary tree traversal class problems through further studying the commonality of binary tree recursion problems.

II. RELATED WORKS

In the early, Gries supplemented and explained the construction loop invariant development strategy proposed by Dijkstra. The loop invariant of the non-recursive algorithm for preorder traversing binary tree was given. There are many ways to find loop invariants. The most important ways are loop invariant construction techniques based on parameterized templates and constraints solving and the loop invariant construction techniques based on abstract interpretation.

III. PRELIMINARY KNOWLEDGE

A. PAR method and PAR platform

In order to change creative labor into non creative labor as much as possible, and realize the goal of universal applicability of algorithm program development method. Therefore, the research team of this laboratory proposed the PAR method [1], which is a method based on partitioning and recursion. The PAR platform is a platform developed to support the PAR method, which improves the correctness and simplicity of programming. There are two methods to generate code. The first method is suitable for dealing with quantification problems, that is, the goals or problems are specific and can be clearly measured. The second method is suitable for dealing with non-quantification problems, that is, problems that cannot be expressed as specific statistics. The user can directly design the Apla program and give the formal proof or the theorem proof, and then turn it into an executable program. The algorithms derived in this paper are suitable for the second method.

B. Loop invariant

"A true predicate before and after each loop iteration" is called loop invariant. Loop invariant is an important link to realize the formalized development of algorithm program. Therefore, we propose a new definition of loop invariant and a new development strategy based on PAR method by using the recursive definition technique of loop invariant [1].

IV. THE STRATEGY OF DERIVATION AND FORMAL PROOF OF BINARY TREE TRAVERSAL ALGORITHM

A. Steps

Combined with the derivation and formal proof of various binary tree traversal non-recursive algorithms, the following seven steps are summarized:

Step 1: A formal specification is constructed to clarify the working objectives of the binary tree algorithm:

AQ: Given a finite binary tree T

AR: X=A list of nodes generated by traversing the binary tree T.
Step 2: Divide the binary tree to get a certain number of subtrees. The subtrees meet the two characteristics of the same structure and the smaller size than the binary tree. Then divide the subtrees in the same way until each subtree is solved.

Step 3: The recursive relationship $S_j = F(S)$ of the list is deduced using the quantifier conversion method, the initial values are assigned to functions and variables.

Step 4: Develop loop invariants. According to the relation between the element in the variable and the recursive relation of the subtrees and the description of the solution of the subtree in the recursive relation, the content of a list is defined recursively.

Step 5: Based on the recursive relationship and loop invariant, the Apla algorithm program is derived [2].

Step 6: Prove the correctness of the Apla algorithm program by Dijkstra-Gries standard proving technique.

Step 7: Using PAR platform C++ program automatic generation system, automatically generate C++ programs.

B. Improve the loop invariant

In the early, Gries provided the loop invariant of the preorder traversal binary tree non-recursive algorithm:

$$\rho : 0 \leq c \leq \# p \wedge \text{preorder}(p) = b[0..c-1]\text{preorder}(0) \cdots \text{preorder}(r[\neg 1])$$

It can be seen that the loop invariant is used as "\(\cdots\)". It is very long, unclear and difficult to understand. Therefore, the new definition and the new development strategy of loop invariants based on PAR method are proposed, and a new loop invariant of the preorder traversal binary tree non-recursive algorithm is given:

$$\rho : \text{Pre}(T) = X \uparrow \text{Pre}(q) \uparrow F(S)$$

By comparing the loop invariants of two preorder traversal binary tree non-recursive algorithms, we can clearly find that our loop invariants are more precise. It is convenient for the derivation and formal proof of binary tree traversal class problems non-recursive algorithm.

V. Derivation and Formal Proof of Binary Tree Traversal Class Problems

Traversing binary tree refers to starting from the root node and sequentially visiting all the nodes in the binary tree in a certain order, so that each node is visited in turn and only once. The four traversal methods include: preorder, inorder, postorder and hierarchical traversal. According to the idea of using queue or stack storage structure to simulate recursion, traversal of binary tree can be divided into two categories.

A. Hierarchical traversal binary tree

Based on the definition of hierarchical traversal binary tree, the idea of queues, and the strategy of derivation and formal proof of binary tree traversal algorithm, we can derive the loop invariant and the Apla algorithm program of the hierarchical traversal binary tree non-recursive algorithm:

$$\rho : \text{Lay}(T) = X \uparrow [q. d] \uparrow F[S \uparrow [q. l] \uparrow [q. r]]$$

The definition of $F$ is as follows:

$$F(S) = [S. h] \uparrow F[S[h+1..t]] \uparrow [S[h]. l] \uparrow [S[h]. r]$$

$$F([q \uparrow S]) = [q. d] \uparrow F[S \uparrow [q. l] \uparrow [q. r]]$$

Apla algorithm procedure:

$$\text{do } q = [g \uparrow 0] \rightarrow \text{if } (q. l\neq%) \rightarrow S := S \uparrow [q. l]; \text{fi;}$$

$$\text{if } (q. r\neq%) \rightarrow S := S \uparrow [q. r]; \text{fi;}$$

$$X, q := X \uparrow [q. d], \%;$$

$$\text{[l] } q = [g\uparrow 0] \rightarrow q, s := S[h], S[h+1..t];$$

The above Apla algorithm program is proved by Dijkstra-Gries standard proving technique. Then, it can automatically generate C++ programs through the PAR platform C++ automatic generation system.

B. Preorder, inorder and postorder traversal binary tree

By deducing and proving preorder, inorder, and postorder traversal binary tree non-recursive algorithms [3], we obtain the universal loop invariant:

$$\rho : \text{Order}(T) = X \uparrow \text{Order}(q) \uparrow F(S)$$

Order (T) represents the list of nodes generated by each traversing binary tree $T$ non-recursive algorithm; the list variable $X$ stores part of the node list generated during the traversal process; $q$ is used to store the subtree of the $T$ being accessed; $S$ is a list variable acting together on the stack. $F(S)$ represents the traversal result of the subtree to be traversed. The difference between the loop invariants of the three traversal algorithms lies in the different definitions of the function $F(S)$:

$$F([q \uparrow S]) = \begin{cases} \text{preorder} & \text{order} \\ \text{inorder} & \text{inorder} \\ \text{postorder} & \text{postorder} \end{cases}$$

VI. Conclusion

The paper uses the algorithm design technique PAR method to derive the non-recursive algorithm program for binary tree traversal class problems. The correctness of the algorithm is proved by Dijkstra-Gries standard proving technique [4]. Finally, the PAR platform is used to automatically generate C++ code. The results of this study point out the direction for the exploration of the loop invariant of the non-recursive algorithm for recursive problems and have guiding significance for the derivation and formal proof of the algorithm program of the nonlinear data structure.

REFERENCES


