Inorder traversal of a binary tree and its inversion

David Gries
Jan L.A. van de Snepscheut

87-876
November 1987

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501

Inorder traversal of a binary tree and its inversion

David Gries, Cornell University
Jan L. A. van de Snepscheut, Groningen University

28 October 1987

This paper presents the derivation of an algorithm for producing the inorder traversal of a binary tree and then shows how to invert the algorithm. Given a sequence of values, the inversion produces, nondeterministically, any binary tree whose inorder traversal is that sequence.

The inorder traversal algorithm was presented by the first author at the Institute on Formal Development of Programs and Proofs in the 1987 UT Year of Programming. The second author derived the inversion, with some help from the first author. The paper is written by the authors in alternating sentences.

The inorder traversal algorithm

Consider a finite binary tree t (say), which for our purposes is best defined recursively by

$t = \emptyset$ (i.e. t is empty) or
$t = (t.l,\ t.d,\ t.r)$

where $t.d$ is an integer and $t.l$ and $t.r$ are binary trees. By $\#t$ we denote the number of nodes in tree t.

We shall also be dealing with sequences of elements. The empty sequence is denoted by ϵ. Catenation of sequences and elements is denoted by juxtaposition. For S a sequence, $\#S$ denotes the number of elements in S.

The inorder traversal $in.t$ of t is a sequence of integers defined by

$$in.\emptyset = \epsilon$$
$$in.(t.l,\ t.d,\ t.r) = in.(t.l)\ \ t.d\ \ in.(t.r)\ .$$

We note that $\#t = \#in.t$.

We derive an iterative algorithm for storing the inorder traversal of a given finite binary tree t in a sequence variable Z, thus establishing

$$R:\ \ in.t = Z\ .$$

From the postcondition we obtain invariant P' by replacing Z with $Z\ in.t$ (so that Z can be initialized to the identity ϵ of catenation) and then replacing constant t with a fresh variable p

$$P'\colon \quad in.t = Z \;\; in.p \;\; .$$

P' can be established using $Z, p := \epsilon, \emptyset$.

Determining the body of the loop requires investigation of the invariant. If $p \neq \emptyset$, using the definition of in , P' expands to

$$in.t = Z \;\; in.(p.l) \;\; p.d \;\; in.(p.r)$$

and, if $p.l \neq \emptyset$, to

$$in.t = Z \;\; in.(p.l.l) \;\; p.l.d \;\; in.(p.l.r) \;\; p.d \;\; in.(p.r) \quad .$$

Each replacement of the term $in.(\_)$ following Z by its definition introduces the expression $\_.d \;\; in.(\_.r)$ into the righthand side. We therefore introduce a fresh sequence variable S to contain the trees occurring in these pairs and generalize the loop invariant to

$$P\colon \quad in.t = Z \;\; in.p \;\; x.S$$

where x is defined by

$$x.\epsilon = \epsilon$$
$$x.(p \;\; S) = p.d \;\; in.(p.r) \;\; x.S \quad .$$

The new invariant P is established by
$Z, p, S := \epsilon, t, \epsilon$.

We now develop the loop. Under the condition $p \neq \emptyset$, P can be manipulated as follows:

$$
\begin{array}{lll}
in.t & = & Z \;\; in.p \;\; x.S \\
& = & \quad \{\text{definition of } in\} \\
& & Z \;\; in.(p.l) \;\; p.d \;\; in.(p.r) \;\; x.S \\
& = & \quad \{\text{definition of } x\} \\
& & Z \;\; in.(p.l) \;\; x.(p \;\; S) \;\; .
\end{array}
$$

Hence P is maintained and $\#p$ (i.e. the number of nodes in tree p) is reduced by execution of

$$p, S := p.l, \; p \;\; S \;\; .$$

Under the condition $p = \emptyset \wedge S \neq \epsilon$, we introduce u and U that satisfy $S = u \;\; U$ and manipulate P:

$$
\begin{array}{lll}
in.t & = & Z \;\; in.p \;\; x.(u \;\; U) \\
& = & \quad \{\text{definition of } in\} \\
& & Z \;\; x.(u \;\; U) \\
& = & \quad \{\text{definition of } x\} \\
& & Z \;\; u.d \;\; in.(u.r) \;\; x.U \;\; .
\end{array}
$$

Hence P is maintained and $\#t - \#Z$ is reduced by execution of

```
let u, U sat  S = u U ;
Z, p, S := Z u.d,  u.r,  U      .
```

The algorithm is therefore

```
A: var p: tree;
   var S: seq(tree);
   Z, p, S := ϵ, t, ϵ;
   do p ≠ ∅                  → p, S := p.l, p S
    ▯ p = ∅ ∧ S ≠ ϵ          → Let u, U sat S = u U;
                                Z, p, S := Z u.d, u.r, U
   od
   {P ∧ p = ∅ ∧ S = ϵ; hence R}  .
```

From P it follows that $\#t \geq \#Z$, which implies that the set of pairs assumed by $(\#t - \#Z, \#p)$ is well founded under lexicographic ordering. Since both statements in the loop decrease $(\#t - \#Z, \#p)$, the algorithm terminates.

The simplicity of this algorithm and its description is a result of several design decisions: the use of a tree instead of its representation using pointers; the introduction of S as a sequence instead of a stack, with its operations push and pop; the decision to use the let statement, instead of referring to parts of Z and S using subscripting; and the recursive definition of x, which makes the derivations of the guarded commands and proofs of invariance of P almost trivial.

Inversion of the algorithm

We now invert algorithm A, thus producing an algorithm $\mathcal{U}$ that, given a sequence Z, nondeterministically stores in p any tree satisfying $in.p = Z$. Algorithms A and $\mathcal{U}$ have the same invariant P. Constant t occurs in P and, in the case of $\mathcal{U}$, we take t to be any tree whose inorder traversal is the given sequence Z.

Algorithm A terminates with $p = \emptyset$ and $S = \epsilon$. Hence, its inverse $\mathcal{U}$ begins by setting $p = \emptyset$ and $S = \epsilon$; note that this establishes invariant P. We invert the loop by inverting each of the two guarded commands in isolation. In inverting a guarded command with its associated postcondition, the guard becomes the postcondition and vice versa. Consider the second guarded command:

$$\{p = \emptyset \wedge S = u\ U\}\ Z, p, S := Z\ u.d, u.r, U\ \{Z \neq \epsilon\}\ .$$

Denote the final value of Z by $Z = V\,v$. Inverting the command requires storing into Z, p, and S their initial values. For p this amounts to the assignment $p := \emptyset$, and for Z it is stripping off the last element v, i.e. $Z := V$. The inverse of $S := U$ is $S := u\ S$, provided we can construct tree u. We have $u.d = v$ and $u.r = p$ but we have no value available for $u.l$, so we leave it open. Thus the inverse command is

$$\{ Z \neq \epsilon \} \ \underline{\text{Let}} \ V, v \ \underline{\text{sat}} \ Z = V\, v; \quad Z, p, S := V, \emptyset, (\_, v, p)\ S \ .$$

We verify that the command maintains P, no matter what tree is chosen for $\_$:

$$\begin{aligned}
& wp(\text{“}Z, p, S := V, \emptyset, (_, v, p)\ S\text{”}, P) \\
= & \quad \{ \text{definition of } P \text{ and } wp \} \\
& in.t = V\ in.\emptyset\ x.((_, v, p)\ S) \\
= & \quad \{ \text{definition of } in \} \\
& in.t = V\ x.((_, v, p)\ S) \\
= & \quad \{ \text{definition of } x \} \\
& in.t = V\ v\ in.p\ x.S \\
= & \quad \{ Z = V\, v \ ; \ \text{definition of } P \} \\
& P \ .
\end{aligned}$$

The precondition $Z \neq \epsilon$ ensures that the Let statement is well defined.

Consider the first guarded command in isolation :

$$\{ p \neq \emptyset \} \quad p, S := p.l, p\ S \quad \{ S \neq \epsilon \} \ .$$

Denote the final value of S by $S = u\ U$. Inverting the command requires storing into p and S their initial values. Inverting $S := p\ S$ requires simply deleting the first element of S -using $S := U$. In terms of the values in p and S upon termination of the above command, into p is to be stored the tree

$$(p,\ u.d,\ u.r) \quad .$$

Thus the inversion of this command is

$$\{S \neq \epsilon\}\ \underline{\text{Let}}\ u, U\ \underline{\text{sat}}\ S = u\ U;$$
$$p,\ S := (p,\ u.d,\ u.r),\ U\ .$$

We verify that the command maintains invariant P:

$$\begin{array}{ll} & wp(\text{“}p, S := (p, u.d, u.r),\ U\text{”},\ P) \\ = & \quad \{\ \text{definition of } P \text{ and } wp\ \} \\ & in.t = Z\ \ in.(p, u.d, u.r)\ \ x.U \\ = & \quad \{\ \text{definition of } in\ \} \\ & in.t = Z\ \ in.p\ \ u.d\ \ in.(u.r)\ \ x.U \\ = & \quad \{\ \text{definition of } x\ \} \\ & in.t = Z\ \ in.p\ \ x.(u\ U) \\ = & \quad \{\ S = u\ U\ ;\ \text{definition of } P\ \} \\ & P\ . \end{array}$$

The precondition $S \neq \epsilon$ ensures that the Let statement is well defined. Hence, the inverted program is

```
Ψ: var  p: tree ;
   var  S: seq (tree) ;
   p, S := ∅, ϵ ;
   do  Z ≠ ϵ  →  Let  V, v  sat  Z = V v ;
                 Z, p, S := V, ∅, (_, v, p) S
    ▯  S ≠ ϵ  →  Let  u, U  sat  S = u U ;
                 p, S := (p, u.d, u.r), U
   od
   { P ∧ Z = ϵ ∧ S = ϵ ;  hence  in.t = in.p }   .
```

Each guarded command reduces the pair $(\#Z, \#t - \#p)$, lexicographically speaking. Investigation of the invariant shows that the pair is bounded from below by $(0, 0)$. Hence the algorithm terminates.

Any tree whose inorder traversal equals the given sequence can be produced by an execution of algorithm Ψ. The sole source of nondeterminism is in the selection of a guarded command if both guards are true.

Remark We neglected to state in invariant P that all elements of sequence S are nonempty trees. In both A and Ψ this is required by the second guarded command; it is ensured by the initialization and both guarded commands. End of remark