ON BACKTRACKING AND GREATEST FIXPOINTS

by

Willem P. de Roever *)

Queen's University at Belfast/Mathematisch Centrum, Amsterdam.

Contents: Descriptions and correctness proofs are discussed of three algorithms involved with iterative processing of tree-structured computations: iterative traversal of a binary tree, a backtracking algorithm, and a marking algorithm of a binary directed graph which constructs a spanning tree of that graph (the Deutsch-Schorr-Waite marking algorithm). The backtracking algorithm is believed to be novel. Intuitively these proofs are complicated by the fact that one does not know beforehand whether the processed space of computations is finite or infinite (; in the finite case a proof is simple). The complication due to the possibility of an infinite (search) space is dealt with by introducing an induction principle which asserts that a given computation is necessarily infinite, and therefore yields an undefined result. This principle, *greatest fixpoint induction*, is both in its actual mechanics and in spirit complementary to Burstall's structural induction.

1. MOTIVATION

1.1 *Practical motivation*

(i) Eventually, to obtain machine-checkable proofs of graph processing algorithms of the kind studied in concrete complexity.

(ii) To find an informal formulation of Hitchcock & Park's analysis of termination.

---

*) The author wrote this paper while supported by a senior visiting fellowship related to SRC Contract No. B/RG/74082 (Program Proving Techniques).

The research described was also carried out:
at the Mathematisch Centrum, Amsterdam;
at the Programming Research Group, Oxford, on a scholarship of the Netherlands Organization for the Advancement of Pure Research (Z.W.O.);
at the Département d'Informatique, University of Rennes, France.

1.2 *Formal motivation*

Equivalence proofs involving recursive procedures are sometimes at first sight hard to find when only Scott induction is available. This may be due to the following situation:

In a naive setting, all induction arguments in such proofs concern induction on the recursion depth of some quantity, and are therefore carried out within the framework of the natural numbers.

However, in the context of program proofs there is no reason for the preponderant role of the natural numbers since one would like equally well to (1) induct directly on the complexity of lists, trees, directed graphs, etc. (structural induction of some sort), or, (2) carry out one's proofs in a system in which the natural numbers are no part of one's underlying domain of data. One therefore looks for natural-number-free induction principles.

Scott induction is such a principle; it generalizes the following induction scheme over the natural numbers, ***n-step-n+1 induction,*** to formal systems in which no natural numbers are explicitly present:

n-step-n+1 induction scheme: If (A1) $A(0)$ can be proved,
and (A2) if one assumes $A(n)$ as hypothesis, then $A(n+1)$ can be proved,
then (A3) $\forall n.\ A(n)$ holds.

Again, if the natural numbers are an explicit part of one's formal system, then n-step-n+1 induction is equivalent to

Course-of-values induction scheme: If (B1) $A(0)$ can be proved,
and (B2) for every m, if for some $0 \le m_1 < m_2 < \ldots < m_k \le m$ one assumes $A(m_1) \wedge A(m_2) \wedge \ldots \wedge A(m_k)$ as hypothesis, then $A(m+1)$ can be proved,
then (B3) $\forall m.\ A(m)$ holds.

However in natural-number-free formalisms there exists no direct analogue of course-of-values induction, because its formulation seems to require explicit knowledge of the natural numbers; therefore only Scott-induction is at one's disposal in such formalisms.

Yet many equivalence proofs - especially those between a recursive procedure and some form of iterative implementation - require induction steps for which (B2), and not (A2), is the natural naive setting.

Transposing such proofs to formalisms in which only Scott's equivalent of (A2) is available as induction step requires sometimes ingenuity. I shall argue, by way of

example, that in such cases greatest fixpoint induction may provide a solution. The resulting proofs display a combination of least and greatest fixpoint reasoning.

However, as observed independently by Robert Milne and Robin Milner, the proofs so obtained also suggest in their turn a n-step-n+1 argument, i.e., application of Scott-induction.

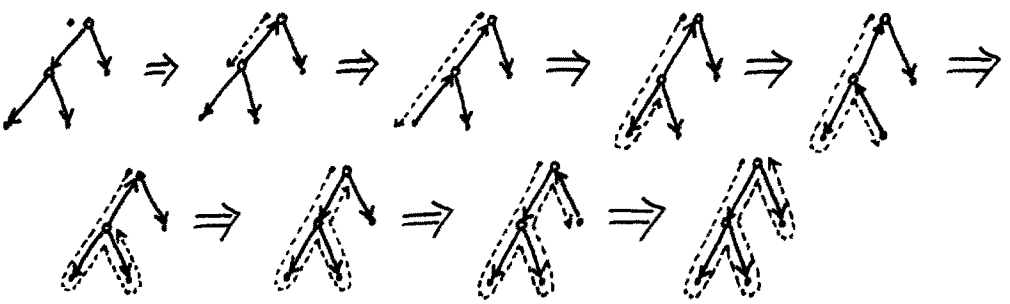
Consequently, from a formal point of view, greatest fixpoint induction is in the considered applications of *heuristic* value.

2. ITERATIVE PROCESSING OF TREE-STRUCTURED COMPUTATIONS

First we discuss iterative traversal of an a priori given binary tree, then iterative processing of tree-structured computations, - i.e., *backtracking* - and finally iterative traversal and construction of spanning trees of binary directed graphs - the Deutsch-Schorr-Waite marking algorithm. The outline of the correctness proof of the last algorithm shows how proofs about program schemes can be relevant to higher-order programs.

2.1 *Iterative versus recursive tree-traversal*

2.1.1 Iterative traversal of a given binary tree can be illustrated by the following picture:



Since each arrow changes exactly twice of direction, the net result of the traversal is *no change at all* : at the end the original tree is recovered.

2.1.2 The above picture illustrates the idea behind the following algorithm:

$$
\begin{aligned}
S(\ell,r) \Leftarrow\ & \underline{\text{if}}\ \text{at}\ (\ell)\ \underline{\text{then}} \\
& \quad \underline{\text{if}}\ \ell=\text{NIL}\ \underline{\text{then}}\ r\ \underline{\text{else}}\ S(r,A(\ell))\ \underline{\text{fi}} \\
& \underline{\text{else}}\quad S(\text{car}(\ell),\text{cons}(\text{cdr}(\ell),r))\ \underline{\text{fi}} \qquad \ldots\ \text{I}
\end{aligned}
$$

Here at(ℓ),car(ℓ),cdr(ℓ),cons(ℓ) denote the usual binary tree processing primitives, call-by-value is used as parameter mechanism, i.e., S is ***strict***, and at(ℓ) implies that A(ℓ) is defined.

2.1.3 Let

$$N(\ell) \Leftarrow \underline{\text{if}}\ \text{at}(\ell)\ \underline{\text{then}}\ \neg(\ell=\text{NIL})\ \underline{\text{else}}\ N(\text{car}(\ell))\wedge N(\text{cdr}(\ell))\ \underline{\text{fi}}$$

define a boolean procedure which determines whether NIL occurs or does not occur amongst the leaves of ℓ. Then correctness of S is expressed by

$$\forall \ell[N(\ell) \rightarrow S(\ell,\text{NIL}) = P(\ell)],$$

where P(ℓ) is given by

$$P(\ell) \Leftarrow \underline{\text{if}}\ \text{at}(\ell)\ \underline{\text{then}}\ A(\ell)\ \underline{\text{else}}\ \text{cons}(P(\text{car}(\ell)),P(\text{cdr}(\ell)))\ \underline{\text{fi}}.$$

A direct proof of this assertion by structural induction on ℓ fails.

By proving

$$\forall \ell,r[N(\ell) \rightarrow S(\ell,r) = S(r,P(\ell))] \qquad \ldots \quad \text{II}$$

instead, the previous assertion follows, since if N(ℓ) holds then

$$S(\ell,\text{NIL}) = (\text{by II})\ S(\text{NIL},P(\ell)) = P(\ell).$$

A proof of II using structural induction is straightforward (; the difficult step is realizing that correctness of S requires proving II):

Assume N(ℓ). There are two cases:

a. at(ℓ): Then $S(\ell,r) = (\text{since } \neg(\ell=\text{NIL}))S(r,A(\ell)) = S(r,P(\ell))$.

b. ¬at(ℓ): Assume the result by hypothesis for car(ℓ) and cdr(ℓ).
Then S(ℓ,r)=S(car(ℓ),cons(cdr(ℓ),r))=(hyp.)
S(cons(cdr(ℓ),r),P(car(ℓ)))=
S(cdr(ℓ),cons(r,P(car(ℓ))))=(hyp.)
S(cons(r,P(car(ℓ))),P(cdr(ℓ)))=
S(r,cons(P(car(ℓ)),P(cdr(ℓ))))=S(r,P(ℓ)),
since N(ℓ) implies N(car(ℓ)) and N(cdr(ℓ)). □

2.1.4 *Generalization to infinite trees*

Without the presence of NIL in ℓ, S(ℓ,r) = S(r,P(ℓ)) also holds if ℓ is *infinite*; then S(ℓ,r) does not terminate, and neither does P(ℓ), and a forteriori S(r,P(ℓ)) since the parameters of S are called by value. (At this stage divergence with call-by-name (or call-by-need) based **non-strict** formalisms occur; in such formalisms P(ℓ) may very well represent for infinite ℓ a well-defined value obtained

as result of a limiting process, while for S(ℓ,r) this limiting process "never takes off", and hence the value of S(ℓ,r) is represented by the "undefined" value ⊥.)

Lifting the finiteness condition off ℓ implies that no inductive argument upon the structure of ℓ is anymore available.

Yet II can still be proved by mathematical induction on the recursion depth of N(ℓ). (This possibility is eliminated in the next section.)

Note that the value of N(ℓ) may now be either true, false, or 'undefined'. Consequently our propositional logic should be adapted [9]. The kind of intricacies involved in proving II for the case N(ℓ) = 'undefined' are illustrated in the section 2.2.4.

2.2 *Backtracking*

2.2.1 *Introduction : representing the 4 queens' problem.*

The 4 queens' problem requires a quaternary tree representation. Therefore a recursive solution has the following form:

$$P'(\ell) \Leftarrow \underline{\text{if}}\ s(\ell)\ \underline{\text{then}}\ P'(S_1(\ell)) \cup P'(S_2(\ell)) \cup P'(S_3(\ell)) \cup P'(S_4(\ell))\ \underline{\text{else}}\ \text{Sol}(\ell)\ \underline{\text{fi}}.$$

The intended interpretation of P'(ℓ) is given by

$$P'(\text{col},s) \Leftarrow \underline{\text{if}}\ \text{col}<4 \wedge \text{free}(\text{col},s)\ \underline{\text{then}}\ \bigcup_{i=1}^{4} P'(\text{col}+1, i \frown s)\ \underline{\text{else}}$$
$$\underline{\text{if}}\ \text{col}=4 \wedge \text{free}(4,s)\ \underline{\text{then}}\ \{s\}\ \underline{\text{else}}\ \phi\ \underline{\text{fi}},$$

where

(1) col∈{1,2,3,4}, s denotes a linear list with 1,2,3,4 as atoms, and the value of P(col,s), if defined, is a set consisting of linear lists,

(2) free is a total predicate checking the particular constraints dictated by the 4-queens' problem,

(3) "∪" denotes set-theoretic union, "⌣" concatenation of an atom to a linear list, "ϕ" the empty set, and {s} the set with element s.

The call P(0,Λ), with Λ denoting the empty linear list, has as value a set consisting of all solutions to the 4-queens' problem [5].

An equivalent iterative solution is obtained by Q(ℓ,endmarker), with Q defined by

$$Q(\ell,r) \Leftarrow \underline{\text{if}}\ \text{is-probl-repr}(\ell)\ \underline{\text{then}}$$
$$\underline{\text{if}}\ s(\ell)\ \underline{\text{then}}\ Q(S_1(\ell),\text{cons}(S_2(\ell),S_3(\ell),S_4(\ell),r))\ \underline{\text{else}}\ Q(r,\text{Sol}(\ell))\ \underline{\text{fi}}$$
$$\underline{\text{else if}}\ \ell=\underline{\text{endmarker}}\ \underline{\text{then}}\ r\ \underline{\text{else}}$$
$$Q(\text{car}(\ell),\text{cons}(\text{cdr}_1(\ell),\text{cdr}_2(\ell),\text{cdr}_3(\ell),r))\ \underline{\text{fi}},$$

where if-probl-repr(ℓ)$\wedge$ ℓ=endmarker implies ℓ=cons(car(ℓ),$cdr_1(\ell)$,$cdr_2(\ell)$,$cdr_3(\ell)$).

This transformation of a recursive solution into an iterative equivalent results in a ***backtracking*** algorithm.

2.2.2 This example motivates our study of the scheme Q, defined by

$$Q(\ell,r) \Leftarrow \underline{\text{if}}\ ipr(\ell)\ \underline{\text{then}}\ \underline{\text{if}}\ s(\ell)\ \underline{\text{then}}\ Q(S_1(\ell),cons(S_2(\ell),r))\ \underline{\text{else}}\ Q(r,A(\ell))\ \underline{\text{fi}}\ \underline{\text{else if}}\ em(\ell)\ \underline{\text{then}}\ r\ \underline{\text{else}}\ Q(car(\ell),cons(cdr(\ell),r))\ \underline{\text{fi}},$$

where

(i) ipr is a total predicate s.t. ipr(ℓ) implies that s(ℓ) is defined, and $\neg$ipr(ℓ) implies that em(ℓ) is defined,

(ii) s(ℓ) iff $S_1(\ell)$ and $S_2(\ell)$ are defined, and $\neg$ s(ℓ) iff A(ℓ) defined,

(iii) $\neg$em(ℓ) implies car(ℓ) and cdr(ℓ) defined,

(iv) S_1,S_2 and A denote functions, $S_1(\ell)$ defined iff $S_2(\ell)$ defined and then ipr($S_1(\ell)$) and ipr($S_2(\ell)$) are satisfied, too, and if A(ℓ) is defined then ipr(A(ℓ)) is satisfied.

(v) car,cdr,cons operate on finite or infinite binary trees over atoms ℓ satisfying either ipr(ℓ) or em(ℓ),*)

(vi) cons and Q are strict, i.e., use call-by-value.

Let T(ℓ) be defined by

$$T(\ell) \Leftarrow \underline{\text{if}}\ s(\ell)\ \underline{\text{then}}\ cons(T(S_1(\ell)),T(S_2(\ell)))\ \underline{\text{else}}\ A(\ell)\ \underline{\text{fi}}.$$

Then we have

$$\forall \ell,r[ipr(\ell)\rightarrow Q(\ell,r) = Q(r,T(\ell))] \quad \ldots \quad \text{III}$$

2.2.3 A proof of III can be split into two parts:

$$\forall \ell,r[ipr(\ell)\rightarrow Q(\ell,r) \sqsubseteq Q(r,T(\ell))] \quad \ldots \quad \text{III.1}$$

$$\forall \ell,r[ipr(\ell)\rightarrow Q(r,T(\ell)) \sqsubseteq Q(\ell,r)] \quad \ldots \quad \text{III.2}$$

Here $\sqsubseteq$ is defined as follows:

Let D denote a set, $\perp$ denote an element not contained in D, and x and y denote elements of D$\cup\{\perp\}$. Then $x \sqsubseteq y$ iff either $x=\perp$ or $x=y$.
Moreover, for n-tuples $\langle x_1,\ldots,x_n\rangle \sqsubseteq \langle y_1,\ldots,y_n\rangle$ iff $x_i \sqsubseteq y_i$ holds coordinatewise, i=1,...,n, and for functions $f_1 \sqsubseteq f_2$ iff, for all x, $f_1(x) \sqsubseteq f_2(x)$ holds pointwise.

*) Note added in proof: and em(cons(ℓ_1,ℓ_2))=ipr(cons(ℓ_1,ℓ_2))= false.

Assertion III.2 can be proved by induction on the recursion depth of $T(\ell)$, and is straightforward.

Proof of assertion III.1, at first sight, seems to require course-of-values induction, also called truncation induction, on the recursion depth of Q.

Let $\sigma[Q]$ denote the procedure body of Q. Then Q_n, the *restriction* of Q to recursion-depth $\leq n$, is defined by $Q_0=\sigma[\lambda\ell,r.\perp/Q]$, $Q_{n+1}=\sigma[Q_n/Q]$, and we have $Q_n \underline{\sqsubset} Q_{n+1}$ for all $n\in N$.

Proof of III.1 using course-of-values induction:

Assume $ipr(\ell)$. There are two cases:

a. $\neg s(\ell)$ *holds:* $Q_0(\ell,r) = \perp \underline{\sqsubset} Q_0(r,T(\ell))$.

For all $k>0$, $Q_k(\ell,r) = Q_{k-1}(r,A(\ell)) = Q_{k-1}(r,T(\ell)) \underline{\sqsubset} Q_k(r,T(\ell))$.

b. $s(\ell)$ *holds:* Assume $\forall \ell,r.(ipr(\ell)\rightarrow Q_j(\ell,r) \underline{\sqsubset} Q_j(r,T(\ell)))$

for $j=n$ and $j=n-1$.

If $n\geq 2$ then

$Q_{n+1}(\ell,r) = Q_n(S_1(\ell),cons(S_2(\ell),r)) \underline{\sqsubset}$ (hyp., since $ipr(S_1\ell)$ follows from (i),(ii), (iv) above)

$Q_n(cons(S_2(\ell),r),T(S_1(\ell)))=$

$Q_{n-1}(S_2(\ell),cons(r,T(S_1(\ell)))) \underline{\sqsubset}$ (hyp., since $ipr(S_2\ell)$ holds)

$Q_{n-1}(cons(r,T(S_1(\ell))),T(S_2(\ell)))=$

$Q_{n-2}(r,cons(T(S_1(\ell)),T(S_2(\ell)))) = Q_{n-2}(r,T(\ell)) \underline{\sqsubset}$

$Q_{n+1}(r,T(\ell))$.

If $n=1$ then $Q_2(\ell,r) \underline{\sqsubset} Q_2(r,P(\ell))$ follows from

$Q_2(\ell,r)=Q_0(cons\ \ldots)$, see above, and $Q_0(cons\ \ldots) = \perp$. □

2.2.4 In mechanized proof systems, proofs using course-of-values induction are undesirable since these involve unnecessary operations on the natural numbers; in such systems one prefers proofs using n-step-n+1 (- Scott -) induction.

Two such proofs are presented.
The first one is given below and applies greatest fixpoint induction, a principle for reasoning about infinite computations.
The second one is given in section 3.4 and amounts to a translitteration of the

course-of-values proof given above.

Proof of III.1 using n-step-n+1 and greatest fixpoint induction:

Introduce the auxiliary boolean procedure k(ℓ):

k(ℓ) <= if s(ℓ) then k(S_1(ℓ))∧k(S_2(ℓ)) else true fi,

assume ipr(ℓ), and distinguish between two cases:

a. k(ℓ) = true: *proof:* by n-step-n+1 induction on the recursion depth of k(ℓ). The proof is similar to that of section 2.1.3 and therefore omitted.

b. k(ℓ) = 'undefined' : *proof:* By greatest fixpoint induction. Since s(ℓ)=false implies k(ℓ)=true, *necessarily* k(ℓ)=k(S_1ℓ)∧k(S_2ℓ)='undefined'; hence either k(S_1ℓ)='undefined' or k(S_1ℓ)=true and k(S_2ℓ)='undefined'.

b1. k(S_1ℓ)='undefined' : Then Q(ℓ,r)=Q(S_1ℓ,cons(S_2ℓ,r)).

b2. k(S_1ℓ)=true and k(S_2ℓ)='undefined' : Then

Q(ℓ,r)=Q(S_1ℓ,cons(S_2ℓ,r))$\underline{\underline{C}}$(by (a) above)Q(cons($S_2$ℓ,r),T($S_1$ℓ))

=Q(S_2ℓ,cons(r,T(S_1ℓ))).

Thus k(ℓ)='undefined' implies in both subcases that computation of Q(ℓ,r) requires computation of an inner call Q(ℓ',...) with k(ℓ')='undefined'. Moreover the two subcases are *exhaustive*. Therefore computation of Q(ℓ,r) has to proceed *infinitely* if k(ℓ)='undefined' is not to be violated, and hence assertion III.1 is trivially fulfilled since the value of Q(ℓ,r) is undefined. □

In section 3 the induction principle behind the infinite-sequences argument used above - greatest fixpoint induction - is formally introduced, and the proof given above is justified.

2.2.5 The reader may object that:

(1) he considers the example too contrived,

(2) he is perfectly satisfied with the proof of III.1 using course-of-values induction,

(3) the example is too trivial since the course-of-values proof at level n+1 uses the hypothesis at levels n and n-1 only, and that he would rather see an algorithm in which the hypotheses are dynamically rather than statically determined.

Objection (1) is met in the next section.

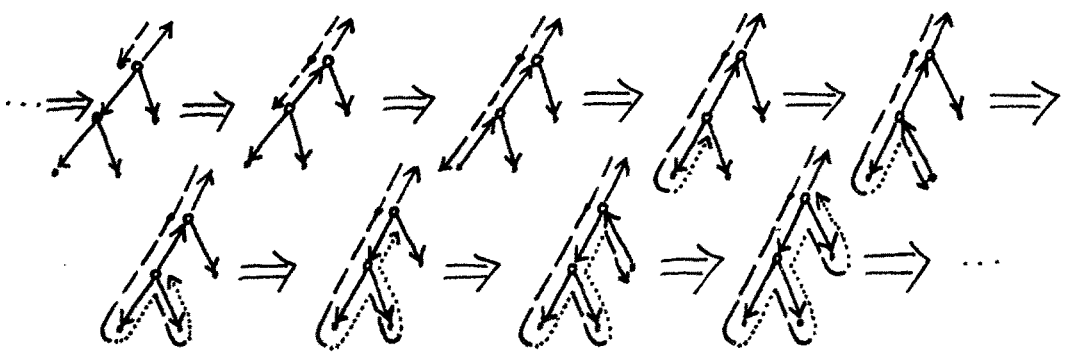
I do not know of an example to answer objection (3) above; it would be very interesting to see. *If anybody finds such an example, please inform me*!

2.3 *The Deutsch-Schorr-Waite marking algorithm.*

2.3.1 *Version for binary trees*

The idea behind the Deutsch-Schorr-Waite marking algorithm is described in two steps:

First we refine the figure in section 2.1.1 as follows:



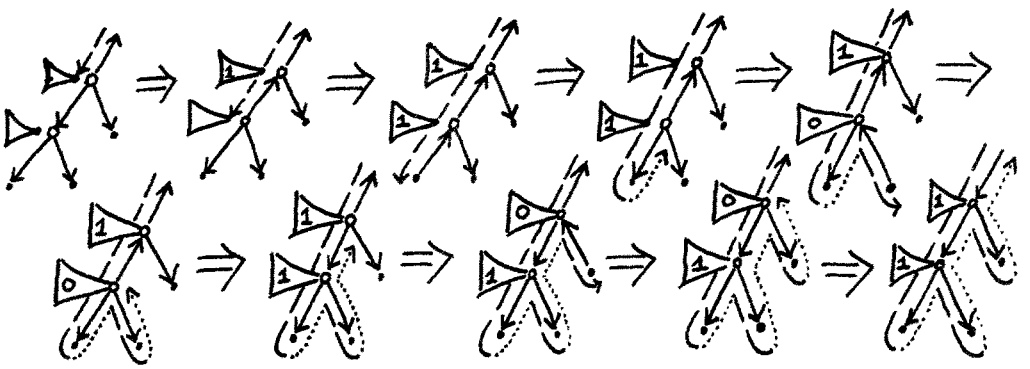
That is, we distinguish during traversal of the tree between a *leftdown* phase and a *backup* phase.

Traversal in leftdown phase ends upon encountering an atom, upon which traversal in backup phase is initiated.

Traversal in backup phase changes into leftdown phase upon encountering a (not yet visited) righthand subtree of the original tree, and remains in backup phase, otherwise; a bit is introduced in each interior node to distinguish between these two cases.

An interior node met in leftdown phase - this is the first time that node is met - is marked by 1 to encode that the righthand subtree must still be traversed. Upon encountering that node (for the first time) in backup phase - this is the second time that node is met - the marking bit is set to 0 to encode preservation of that phase after the righthand tree has been traversed. Finally, the third time that node has been met, its bit has served its purpose as far as marking phase differences is concerned, and it is systematically set to, e.g., 1.

This reflected in the following figure:



Hence one obtains the following marking algorithm for binary trees (with one bit in each interior node):

```
Left(ℓ,r) <= if at(ℓ) then Back(ℓ,r) else Left(car(ℓ),cons(cdr(ℓ),r,1)) fi,

Back(ℓ,r) <= if r=NIL then ℓ else
                 if bitfield(r)=1 then Left(car(r),cons(cdr(r),ℓ,0))
                    else Back(cons(cdr(r),ℓ,1),car(r)) fi,
```

where NIL serves as endmarker, and bitfield(r) isolates the marking-bit of r.

Let

```
M(ℓ) <= if at(ℓ) then ℓ else cons(M(car(ℓ)),M(cdr(ℓ)),1) fi
```

denote the obvious recursive version of the marking algorithm.
Then the following assertion holds both for finite and infinite trees ℓ (- cons calls its arguments by value -).

$\forall \ell, r.[\text{Left}(\ell,r) = \text{Back}(M(\ell),r)]$.

This algorithm has been investigated independently in Burstall[10] and de Roever[11]; its correctness proof is similar to that of section 2.2.4.

2.3.2 *Version for directed binary graphs.*

In the above algorithm new nodes of the tree are encountered for the first time in leftdown phase, and are then submitted to a test of the form at(ℓ).

Therefore, in the case of traversal of a cycle of a binary directed graph, that test is the spot in the algorithm where newly encountered nodes have to be distinguished from already visited ones in order to prevent infinite repetition in traversal of

that cycle.
This distinction is made by introducing a second marking bit which enables marking a node upon encountering that node for the first time, and by replacing the test at(ℓ) by a test at(ℓ)∨m(ℓ) with m(ℓ) checking whether or not a node has been visited already.

In order to express the change in underlying formalism required to express binary directed graphs, a new formalism is introduced.

New formalism (based on [8]):

Let ***At*** and ***Loc*** denote two disjoint sets, and ***at*** denote a total predicate over these sets satisfying at(α) = true iff α∈At.

A ***memory*** for representing binary directed graphs with two marking bits is a total function $\alpha:Loc\to\{0,1\}^2\times(Loc\cup At)^2$.

Changes of such a memory are described as follows:

Let $a_1,a_2\in\{0,1\}$, and $a_3,a_4\in(Loc\cup At)$, then $\sigma[\alpha\to a_1,a_2,a_3,a_4]$ is for α∈Loc defined by

$$\lambda\beta\in Loc.\ \underline{if}\ \alpha=\beta\ \underline{then}\ \langle a_1,a_2,a_3,a_4\rangle\ \underline{else}\ \sigma(\beta)\ \underline{fi}.$$

For α∈Loc, elements of the quadruple σ(α) are accessed by the (total) functions
m,f: Loc×Mem→{0,1} and
hd,tl: Loc×Mem→Loc ∪ At, with Mem denoting the collection of memories as defined above, and ***m,f,hd,tl*** defined by:

If α∈Loc, σ∈Mem, *and* $\sigma(\alpha)=\langle a_1,a_2,a_3,a_4\rangle$, *then*
$m(\alpha,\sigma)=a_1, f(\alpha,\sigma)=a_2, hd(\alpha,\sigma)=a_3, tl(\alpha,\sigma)=a_4$.

In view of the above, the Deutsch-Schorr-Waite marking algorithm of binary directed graphs, expressed by LEFT(α,β,σ) and BACK(α,β,σ) below, should now be obvious:

```
LEFT(α,β,σ) <= if at(α) ∨ m(α,σ) = 1 then BACK(α,β,σ)
               else
               LEFT(hd(α,σ),α,σ[α→1,1,tl(α,σ),β]) fi,

BACK(α,β,σ) <= if β = NIL then <α,σ> else
               if f(β,σ) = 1 then
               LEFT(hd(β,σ),β,σ[β→m(β,σ),0,tl(β,σ),α])
               else
               BACK(β,hd(β,σ),σ[β→1,1,tl(β,σ),α]) fi.
```

Let

```
M(α,σ) <= if at(α) ∨ m(α,σ) = 1 then σ else
          M(tl(α,σ),M(hd(α,σ),σ[α→1,f(α,σ),hd(α,σ),tl(α,σ)])) fi,
```

denote the obvious recursive version of the marking procedure.

Then *correctness of the Deutsch-Schorr-Waite marking algorithm for binary directed graphs* is expressed by

$$\forall\alpha,\sigma.\left[\text{LEFT}(\alpha,\text{NIL},\sigma) = \langle\alpha,\text{M}(\alpha,\sigma)\rangle\right],\ldots$$

a special case of

$$\forall\alpha,\beta,\sigma.\left[\text{LEFT}(\alpha,\beta,\sigma) = \text{BACK}(\alpha,\beta,\text{M}(\alpha,\sigma))\right] \qquad \ldots \quad \text{IV}$$

The only interesting part in the proof of assertion IV is that of

$$\forall\alpha,\beta,\sigma.\left[\text{LEFT}(\alpha,\beta,\sigma) \sqsubseteq \text{BACK}(\alpha,\beta,\text{M}(\alpha,\sigma))\right]$$

This assertion is proved by

- introducing the boolean procedure $k(\alpha,\beta,\sigma)$:

$k(\alpha,\beta,\sigma)$ <= if at(ℓ) ∨ m(α,σ) = 1 then true else
k (hd(α,σ),α,σ[α→1,1,tl(α,σ),β]) ∧
k (tl(α,σ),α,M(hd(α,σ),σ[α→1,0,β,hd(α,σ)])) fi,

- distinguishing between the cases $k(\alpha,\beta,\sigma)$ = true and

$k(\alpha,\beta,\sigma)$ = 'undefined',

cf. de Roever[6]; this proof was inspired by Topor[8], and derives from an attempt to elucidate the inductive structure underlying Topor's proof.

Remark: Michael Patterson observed that the idea behind algorithm S as expressed in section 2.1.1 applies also to directed graphs *without marking bits*. This can be understood by translitterating S into a formalism such as described above (but deleting the part dealing with marking bits).

Open problem: Prove correctness of this new version of S (, for traversal of unmarked directed graphs).

3. FORMAL JUSTIFICATION

First, greatest fixpoint induction is formally introduced as a valid principle in its own right.

Then our use of it in section 2 is justified.

Finally, we present Milne's and Milner's observation that the use of greatest fixpoint induction in section 2 also suggests a proof using n-step-n+1 (- Scott -) induction.

3.1 *Greatest fixpoint induction*

Greatest fixpoints, and their associated operators, were described by David Park in [4]; their first application in programming - expression of *fairness* of a merge of two infinite sequences - was also formulated by him.

Just as in the context of complete partial orders the least fixpoint of an equation $X = \sigma(X)$ is defined by $\mu X[\sigma] \underset{DEF}{=} \bigsqcap\{X \mid \sigma(X) \sqsubseteq X\}$, in dual fashion its *greatest* fixpoint is defined by

$$\nu X[\sigma] \underset{DEF}{=} \bigsqcup\{X \mid X \sqsubseteq \sigma(X)\}.$$

It can be checked that $\nu X[\sigma]$ is a fixpoint, indeed.

Consequently the following rule, *greatest fixpoint induction*, is valid:

$$\tau \sqsubseteq \sigma(\tau) \vdash \tau \sqsubseteq \nu X[\sigma].$$

Obviously greatest fixpoint induction is dual to the following property - least fixpoint induction - of least fixpoints:

$$\sigma(\tau) \sqsubseteq \tau \vdash \mu X[\sigma] \sqsubseteq \tau$$

As usual, these definitions and this rule can be generalized to finite or infinite systems of equations.

Existence of these fixpoints is guaranteed by *monotonicity* of $\sigma(X)$ in X (; $\{X \mid X \sqsubseteq \sigma(X)\}$ is then obviously a directed set).

Expression of the relationship between least and greatest fixpoints requires complete partial orders equipped with a complementation, or negation, operator "$\bar{\ }$". Examples of such structures are complete algebras of relations, or complete boolean algebras (in the latter case the "$\neg$" sign will be used).

In such structures

$$\nu X[\sigma(X)] = \overline{\mu X[\overline{\sigma(\bar{X})}]} \qquad \ldots \quad V.$$

holds, as originally observed by Park, and proved below:

Proof of V:

$\sqsubseteq$: Since $\nu X[\sigma]$ is a fixpoint of $X=\sigma(X)$,

$\nu X[\sigma] \sqsubseteq \sigma(\nu X[\sigma])$ <=>

$\overline{\sigma(\overline{\nu X[\sigma]})} \sqsubseteq \overline{\nu X[\sigma]}$ => (by least fixpoint induction)

$\mu X[\overline{\sigma(\bar{X})}] \sqsubseteq \overline{\nu X[\sigma]}$ <=>

$\nu X[\sigma] \sqsubseteq \overline{\mu X[\overline{\sigma(\bar{X})}]}$

$\sqsupseteq$: Since $\mu X[\overline{\sigma(\bar{X})}]$ is a fixpoint of $\overline{\sigma(\bar{X})}=X$,

$\overline{\sigma(\overline{\mu X[\overline{\sigma(\bar{X})}]})} \sqsubseteq \mu X[\overline{\sigma(\bar{X})}]$ <=>

$\overline{\mu X[\overline{\sigma(\bar{X})}]} \sqsubseteq \sigma(\overline{\mu X[\overline{\sigma(\bar{X})}]})$ => (by greatest fixpoint induction)

$\overline{\mu X[\overline{\sigma(\bar{X})}]} \sqsubseteq \nu X[\sigma]$. □

3.2 *Formal application*

3.2.1 Consider again the recursive procedure T(ℓ) of section 2.2 over (finite or infinite) binary trees: T(ℓ) <= if s(ℓ) then T(S_1(ℓ))∧T(S_2(ℓ)) else A(ℓ) fi, assuming that s(ℓ) implies that A(ℓ) is defined.

It can be proved [1],[2] that T(ℓ) terminates iff k(ℓ) - recall

k(ℓ) <= if s(ℓ) then k(S_1(ℓ))∧k(S_2(ℓ)) else true fi -

terminates, i.e. k(ℓ) = true.

Let L be the complete lattice of total predicates p from finite or infinite binary trees to the complete lattice $\emptyset$ defined by {false,true|false $\sqsubseteq$ true}.

Define σ_1 by

$$\sigma_1 \underset{\text{DEF}}{=} \lambda p.\lambda \ell.\ \text{if } s(\ell) \text{ then } p(S_1(\ell))\vee p(S_2(\ell)) \text{ else false fi.}$$

σ_1 is a continuous functional over L, and hence its greatest fixpoint (denoted by) $\nu p[\sigma_1(p)]$ exists.

Now k is characterized by the least fixpoint $\mu p[\tau_1(p)]$ of the dual transformation

$$\tau_1 \underset{\text{DEF}}{=} \lambda p.\lambda \ell.\ \text{if } s(\ell) \text{ then } p(S_1(\ell))\wedge p(S_2(\ell)) \text{ else true fi.}$$

It follows from V that

$$\nu p[\sigma_1(p)](\ell) \leftrightarrow \mu p[\tau_1(p)](\ell), \qquad \ldots \quad \text{VI}$$

since $\neg\sigma_1(\neg p) = \lambda \ell.$ if s(ℓ) then ¬(¬p(S_1(ℓ))∨¬p(S_2(ℓ))) else true fi = $\tau_1(p)$.

Consequently, $\nu p[\sigma_1(p)](\ell)$ = true iff T(ℓ) does not terminate.

3.2.2 A similar argument applies to

$$Q(\ell,r) \Leftarrow \underline{\text{if}}\ \text{ipr}(\ell)\ \underline{\text{then}}\ \underline{\text{if}}\ \text{s}(\ell)\ \underline{\text{then}}\ Q(S_1(\ell),\text{cons}(S_2(\ell),r))\ \underline{\text{else}}\ Q(r,A(\ell))\ \underline{\text{fi}}\ \underline{\text{else}}\ \underline{\text{if}}\ \text{em}(\ell)\ \underline{\text{then}}\ r\ \underline{\text{else}}\ Q(\text{car}(\ell),\text{cons}(\text{cdr}(\ell),r))\ \underline{\text{fi}}.$$

Introduce the functional σ_2 over L:

$$\sigma_2 \underset{\text{DEF}}{=} \lambda p.\lambda \ell,r.\ \underline{\text{if}}\ \text{ipr}(\ell)\ \underline{\text{then}}\ \underline{\text{if}}\ \text{s}(\ell)\ \underline{\text{then}}\ p(S_1(\ell),\text{cons}(S_2(\ell),r))\ \underline{\text{else}}\ p(r,A(\ell))\ \underline{\text{fi}}\ \underline{\text{else}}\ \underline{\text{if}}\ \text{em}(\ell)\ \underline{\text{then}}\ \underline{\text{false}}\ \underline{\text{else}}\ p(\text{car}(\ell),\text{cons}(\text{cdr}(\ell),r))\ \underline{\text{fi}}.$$

Then $\nu p[\sigma_2(p)](\ell,r) = \underline{\text{true}}$ iff $Q(\ell,r)$ does not terminate.

3.2.3 Using these notions, our proof in section 2.2.4 of

$$\forall \ell,r[\text{ipr}(\ell) \rightarrow Q(\ell,r) = Q(r,T(\ell))]$$

involving informal use of greatest fixpoint induction, can be formally justified using the principle defined in section 3.1:

Assume $\text{ipr}(\ell)$. By VI,

$$\mu p[\tau_1(p)](\ell) \vee \nu p[\sigma_1(p)](\ell) = \underline{\text{true}}.$$

Hence the proof splits into:

a. $\forall \ell,r[\mu p[\tau_1(p)](\ell) \rightarrow Q(\ell,r) = Q(r,T(\ell))]$. Proof: by simultaneous Scott induction on $\mu p[\tau_1(p)]$ and T. (Cf. section 2.2.4).

b. $\forall \ell,r[\nu p[\sigma_1(p)](\ell) \rightarrow \nu p[\sigma_2(p)](\ell,r)]$. I.e., $k(\ell)$ = 'undefined' implies that $Q(\ell,r)$ does not terminate. This is sufficient to prove, since $\nu p[\sigma_1(p)](\ell) = \underline{\text{true}}$ iff $T(\ell)$, and a forteriori $Q(r,T(\ell))$, does not terminate.
By introducing the predicate transformer $R \circ p = \lambda x.\ y[xRy \wedge p(y)]$ and the strict projection function π_i defined by $\pi_i(\langle \ell_1,\ell_2 \rangle) \underset{\text{DEF}}{=} \ell_i, i = 1,2$,
$\forall \ell,r[\nu p[\sigma_1(p)](\ell) \rightarrow \nu p[\sigma_2(p)](\ell,r)]$ is expressible by the inclusion

$$\pi_1 \circ \nu p[\sigma_1(p)] \sqsubseteq \nu p[\sigma_2(p)].$$

In this formulation, Park's rule of greatest fixpoint induction is applicable:

$$\underbrace{\pi_1 \circ \nu p[\sigma_1(p)] \sqsubseteq \sigma_2(\pi_1 \circ \nu p[\sigma_1(p)])}_{\equiv \text{ LHS}} \vdash \pi_1 \circ \nu p[\sigma_1(p)] \sqsubseteq \nu p[\sigma_2(p)].$$

LHS is formally proved in [7].

Thus our informal appeal to greatest fixpoint induction in section 2.2 can be

underpinned by formal application of the principle introduced in section 3.1. Similar arguments apply to other informal appeals to this principle in section 2. This justifies our use of the term greatest fixpoint induction in section 2.

Remark. Robert Milne observed that by introducing the lattice T = {⊥,false,true | ⊥ ⊑ false, ⊥ ⊑ true}, and by considering least fixpoints of σ_1 and τ_1 w.r.t. T, VI can be proved by Scott induction. Then one could probably prove the above results using least fixpoints only.

3.2.4 Obviously, our use of this rule depends critically on finding auxiliary predicates - such as k - describing the domain of (non) termination of recursive procedures. In general, existence of such predicates follows from Hitchcock & Park[2]. Our particular versions of these predicates are obtained by applying a theorem of de Bakker's[1].

3.3 *An alternative proof*

Robert Milne and Robin Milner observed independently of each other that the informal use of greatest fixpoint induction in section 2 suggests proofs by n-step-n+1 induction. This is illustrated by the following proof of

$$\forall \ell,r[ipr(\ell)\rightarrow Q(\ell,r) \sqsubseteq Q(r,T(\ell))] \qquad \ldots \qquad \text{III.1}$$

In this proof ⊥ denotes either false in $\mathcal{O}$, or the undefined value representing nontermination.

III.1 follows from proving

$$(Q_i \sqsubseteq Q_{i+1}) \wedge \forall \ell,r[ipr(\ell)=true \rightarrow \{(k(\ell)=\perp \rightarrow Q_i(\ell,r)=\perp) \wedge (k(\ell)=true \rightarrow Q_i(\ell,r) \sqsubseteq Q_i(r,T(\ell)))\}].$$

by induction on i :

Proof: n=0: Immediate.

n=i+1: Assume the result by hypothesis for n=i.

(a) Assume $ipr(\ell)=true$ and $k\ell=\perp$. Then of course $s(\ell)=true$, $Q_{i+1}(\ell,r)=Q_i(S_1\ell,cons(S_2\ell,r))$, $S_1\ell$ is defined and $ipr(S_1\ell)=ipr(S_2\ell)=true$.
We have either $k(S_1\ell)=\perp$ *or* $k(S_1\ell)=true$ and $k(S_2\ell)=\perp$.
In the first case, $Q_i(S_1\ell,\ldots)=\perp$, by hypothesis.
In the second case,
$Q_i(S_1\ell,cons(S_2\ell,r)) \sqsubseteq$ (hyp.,2nd conjunct) $Q_i(cons(S_2\ell,r),T(S_1\ell)) \sqsubseteq$
(hyp.) $Q_{i+1}(cons(S_2\ell,r),T(S_1\ell))=Q_i(S_2\ell,\ldots)=$(hyp.)$\perp$.

(b) Assume $ipr(\ell)=\underline{true}=k\ell$. Of course, $s(\ell)$ is defined; we proceed analogous to section 2.2.4 case (a):

(i) $s(\ell)$ = $\underline{false}$:

Then $Q_{i+1}(\ell,r)=Q_i(r,A(\ell))=Q_i(r,T(\ell))\sqsubseteq Q_{i+1}(r,T(\ell))$.

(ii) $s(\ell)=\underline{true}$: Then since $k\ell=\underline{true}, kS_1\ell=kS_2\ell=\underline{true}$.

Then $Q_{i+1}(\ell,r)=Q_i(S_1\ell,cons(S_2\ell,r))$

$\sqsubseteq$(hyp.)$Q_i(cons(S_2\ell,r),T(S_1\ell))$

$\sqsubseteq$(hyp.)$Q_{i+1}(cons(S_2\ell,r),T(S_1\ell))$

$= Q_i(S_2\ell,cons(r,T(S_1\ell)))$

$\sqsubseteq$(hyp.)$Q_i(cons(r,T(S_1\ell)),T(S_2\ell))$

$\sqsubseteq$(hyp.)$Q_{i+1}(cons(r,T(S_1\ell)),T(S_2\ell))$

$=Q_i(r,cons(T(S_1\ell),T(S_2\ell)))=Q_i(r,T(\ell))$

$\sqsubseteq$(hyp.)$Q_{i+1}(r,T(\ell))$.

(c) $Q_{i+1}\sqsubseteq Q_{i+2}$ follows by monotonicity of the transformations induced by procedure bodies. □

3.4 *In retrospect*

In the meantime more proofs of III.1 have been found, displaying the rich variety which Scott induction allows in this kind of examples.

The simplest of these proofs is the following one:

Proof of III.1 using n-step-n+1 induction:

Assume $ipr(\ell)$. We prove for all n:

$$\forall \ell,r[ipr(\ell)\rightarrow Q_n(\ell,r)\sqsubseteq Q_n(r,T\ell)].$$

n=0: $Q_0(\ell,r)=\perp\sqsubseteq Q(r,T\ell)$.

n=k+1: Assume the result by hypothesis for n=k.

a. $\neg s(\ell)$ holds: $Q_{k+1}(\ell,r)=Q_k(r,A\ell)=Q_k(t,T\ell)\sqsubseteq Q_{k+1}(r,T\ell)$.

b. $s(\ell)$ holds: $Q_{k+1}(\ell,r)=Q_k(S_1\ell,cons(S_2\ell,r))\sqsubseteq$(hyp.)
$Q_k(cons(S_2\ell,r),T(S_1\ell))\sqsubseteq Q_{k+1}(cons(S_2\ell,r),T(S_1\ell))$ =
$Q_k(S_2\ell,cons(r,T(S_1\ell)))\sqsubseteq$(hyp.)$Q_k(cons(r,T(S_1\ell)),T(S_2\ell))\sqsubseteq$
$Q_{k+1}(r,T\ell)$. □

I tried to tame the variety mentioned above by simulating a structural-induction-type argument by

(1) introducing auxiliary predicates, and

(2) applying greatest fixpoint induction in order to reason about nontermination.

Acknowledgement: I am grateful to Bruno Courcelle, Robert Milne, Robin Milner, Michael Patterson and Christopher Wadsworth for their interest. Robin Milner's criticism was justified. Robert Milne generated several alternative proofs of III of varying complexity, invented systems [3] which tackled the general problem raised in section 1.2 of this note, and wrote me many helpful letters.

Belfast, 7th April, 1977.

4. REFERENCES.

[1] de Bakker, *Termination of nondeterministic programs,* in 3rd colloquium on automata, languages & programming, Michelson & Milner (eds.), Edinburgh University Press, 1976.

[2] Hitchcock & Park, *Induction rules and proofs of termination,* in Proc. IRIA symp. on automata, formal languages, and programming, M. Nivat (ed.), North-H., 1972.

[3] Milne, *Private communication.*

[4] Park, *Fixpoint induction and proof of program semantics,* in Machine Intelligence V, pp 59-78, Edinb. Univ. Press, 1970.

[5] de Roever, *On proofs about backtracking: a tutorial,* report no.51, Dép. d'Informatique, Univ. de Rennes, France, 1976.

[6] de Roever, Lecture notes for *"Introduction to programme correctness"* unpublished notes, Univ. of Oxford, 1975.

[7] de Roever, *Maximal fixpoints solve some of the problems with Scott induction,* Université de Rennes, France, 1976.

[8] Topor, *Correctness of the Schorr-Waite list marking algorithm,* Memo MIP - R-104, School of Artificial Intelligence, Univ. of Edinburgh, 1974.

[9] Manna & McCarthy, *Properties of programs and partial function logic,* in M.I.V., Meltzer & Michie (eds.), Edinb. Univ. Press, 1970.

[10] Burstall, *Program proving as hand simulation with a little induction,* IFIP 74, North-Holland.

[11] de Roever, *Recursion and parameter mechanisms: an axiomatic approach,* Proc. 2nd coll. on Automata, Languages, and Programming, Springer C. Sc. 14 (1974).