Transformational programming and forests

A. Bijlsma
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Introduction

In an earlier paper [0], we introduced the concept of constructs. These may be regarded as a common generalization of finite and infinite sequences, finite and infinite bags, trees of arbitrary arity and forests. In this paper we attempt to show how constructs may be used as an alternative foundation for the transformational programming method proposed by Bird [1] and Meertens [2]. In our view, the main advantage of such an approach would be the fact that generally applicable operations like "map" and "reduce" can be given a simple definition at the level of constructs, so that the need for referring to polymorphism or type hierarchies disappears. The later sections of this paper introduce an operator on constructs that generalizes both "map" and "reduce" and may be used to express in a succinct way many properties of trees and forests. In the last sections it is shown how the properties of this operator can be exploited to transform some specifications involving trees and forests into efficient functional programs.

Constructs

In this section we list some facts about constructs. Proofs are omitted, but can be found in [0].

Consider mappings from (partially) ordered sets to arbitrary sets. Two such mappings, say $f \in A \to X$ and $g \in B \to Y$, are called equivalent if there exist an isomorphism σ of A onto B such that $f = g \circ \sigma$. (Note that this implies $X = Y$). With every mapping f we associate an object $C(f)$ in such a way that $C(f) = C(g)$ if and only if f and g are equivalent; the objects $C(f)$ are called constructs.

A finite sequence is a construct, say $C(f)$ with $f \in A \to X$, such that f is surjective and A is a finite, linearly ordered set. In case A is the set $\{1, 2, ..., n\}$ with its usual ordering, it is common practice to denote $C(f)$ by $\langle f(1), f(2), ..., f(n)\rangle$. Every finite sequence can be written as $\langle x_1, x_2, ..., x_n\rangle$ in precisely one way.

A bag is a construct, say $C(f)$ with $f \in A \to X$, such that f is surjective and A is discretely ordered. In case A is the set $\{1, 2, ..., n\}$ with the discrete ordering, we shall denote C(f) by [f(1), f(2), ..., f(n)] . Every finite bag can be written as $[x_1, x_2, ..., x_n]$. The elements x_1, x_2, ..., x_n are uniquely determined but for permutations. Two bags, say C(f) and C(g) with $f \in A \to X$ and $g \in B \to Y$, are equal if and only if X = Y and, for all x in X ,

$$(\underline{N}\ i: i \in A: f(i) = x) = (\underline{N}\ j: j \in B: g(j) = x) \quad .$$

For a construct C(f) with $f \in A \to X$ and a mapping $g \in X \to Y$ we define

$$g*(C(f)) = C(g \circ f) \quad .$$

For instance, if C0 is the infinite bag that contains every integer value twice, and g is the mapping defined by g(x) = 2x + 1 , then g*(C0) is the infinite bag that contains every odd integer value twice. (In [1] and [2] the symbol * was introduced for finite sequences and finite bags separately. The pronunciation of * is variously reported as "map" [1] or "applied to all" [2].)

Let X be any set and let A be the ordered set obtained by ordering X discretely. The construct $C(id_A)$ is called the <u>set construct</u> of X : it is a bag in which every element of X occurs precisely once. Conversely, for any construct, say C(f) with f ∈ A – X , we define its <u>base set</u> to be im f . Then, for any set X , the base set of the set construct of X is again X ; in this sense there is a natural correspondence between sets and set constructs. We shall exploit this correspondence in the following way: if f is some mapping from constructs to constructs and X is a set, we denote by f(X) the base set of the image under f of the set construct of X . For instance, if **Z** is the set of integers and g is defined on **Z** by $g(x) = x^2$, then g*(**Z**) is the set of natural numbers.

Let A and B be ordered sets. The <u>concatenation</u> of A and B , denoted A ++ B , is the set

$$(0) \qquad \{(0, a) \mid a \in A\} \cup \{(1, b) \mid b \in B\} \quad ,$$

ordered in such a way that

$$(i0,\ c0) \le_{A{+}{+}B} (i1,\ c1)$$
$$\equiv (i0 = 0 \wedge i1 = 1)$$
$$\vee\ (i0 = i1 = 0 \wedge c0 \le_A c1) \vee (i0 = i1 = 1 \wedge c0 \le_B c1) \quad .$$

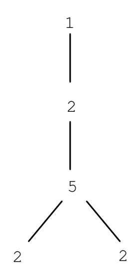
Consider constructs $C(f)$ and $C(g)$, where $f \in A \to X$ and $g \in B \to Y$. The <u>concatenation</u> of $C(f)$ and $C(g)$, denoted $C(f) ++ C(g)$, is the construct $C(h)$, where $h \in (A ++ B) \to X \cup Y$ is defined by

(1) $h(0,\ a) = f(a) \wedge h(1,\ b) = g(b)$

for $a \in A$, $b \in B$. Thus

$$\langle 1, 2, 5\rangle ++ \langle 5, 3, 0\rangle = \langle 1, 2, 5, 5, 3, 0\rangle$$

and $\langle 1, 2, 5\rangle ++ [2, 2]$ is the partially ordered construct



2

Concatenation of constructs reduces to the normal concatenation operator when the constructs are sequences. There is also an operation on constructs that generalizes the normal operator on bags:

Let A and B be ordered sets. The <u>sum</u> of A and B , denoted $A + B$, is the set (0), ordered in such a way that

$$(i0,\ c0) \le_{A+B} (i1,\ ci)$$
$$\equiv (i0 = i1 = 0 \wedge c0 \le_A c1) \vee (i0 = i1 = 1 \wedge c0 \le_B c1) \quad .$$

Consider constructs $C(f)$ and $C(g)$, where $f \in A \to X$ and $g \in B \to Y$. The <u>sum</u> of $C(f)$ and $C(g)$, denoted $C(f) + C(g)$, is the construct $C(h)$, where $h \in (A + B) \to X \cup Y$ is defined by (1).

We shall call a construct $C(f)$, with $f \in A \to X$, <u>empty</u> if A is empty. It follows that there is only one empty construct, namely $\{\emptyset\}$.

We shall call a nonempty construct <u>irreducible</u> if it cannot be written as the concatenation or sum of nonempty constructs. Clearly, every finite nonempty construct may be built from irreducible constructs with the operators ++ and + . The question naturally occurs whether this decomposition is in some sense unique. Obviously, it cannot be completely unique, since both ++ and + are associative and + is commutative (as operators between constructs; they are so up to isomorphism as operators between ordered sets). Therefore, we must allow rearrangement of terms and introduction of parentheses that exploit these properties; for instance, $[2] + ([1] + [0]) = [0] + [1] + [2]$. However, apart from this the decomposition turns out to be unique.

Let A be an ordered set. The relation $(<_A)$ is defined on A by

$$a\ (<_A)\ b \equiv a <_A b \wedge \neg\ (\underline{E}\ x\colon x \in A\colon a <_A x <_A b) \quad .$$

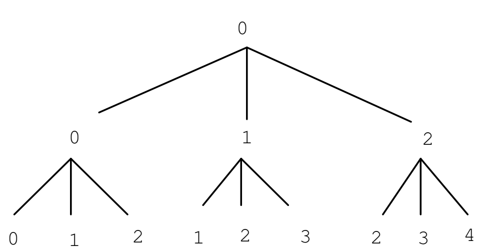
If a and b satisfy $a\ (<_A)\ b$, we say that a is a <u>predecessor</u> of b, and that b is a <u>successor</u> of a.

An ordered set A is called <u>tree-like</u> if A is finite and there exists an r in A such that for all a in A,

$$(\underline{N}\ x\colon x \in A\colon x\ (<_A)\ a) = \underline{\text{if}}\ a = r \to 0\ \square\ a \neq r \to 1\ \underline{\text{fi}} \quad .$$

If such an r exists, it is obviously unique; we call it the <u>root</u> of A. In a tree-like ordered set, the root is the least element.

A <u>tree</u> is a construct $C(f)$, say with $f \in A \to X$, such that A is tree-like and f is surjective. For example, let A denote the set of words of length at most 2 over the alphabet $\{0, 1, 2\}$. For words t and u we let $t \leq_A u$ mean that t is a prefix of u. Define f on A as the sum of the numerical values of the symbols in a word. Then $C(f)$ is the ternary tree



Note that the left-right ordering of the successors of any given node is purely arbitrary: interchanging, for instance, the leftmost 0 and 1 on the bottom row would not change C(f) . Note also that this tree could equally well have been described with the aid of the operators ++ and + , since it is exactly equal to

```
[0] ++ (  ([0] ++ ([0] + [1] + [2]))
        + ([1] ++ ([1] + [2] + [3]))
        + ([2] ++ ([2] + [3] + [4]))
       )    .
```

In fact, every tree may be built from singleton bags with the operators ++ and + . As singleton bags are obviously irreducible, it follows from the above that this decomposition is essentially unique.

(It is not true that every finite construct can be decomposed into singleton bags. As a counterexample, consider the ordered set A consisting of {2, 3, 6, 9} with the divisibility ordering. If f is the identity function on A , then C(f) is neither a concatenation nor a sum.)

A sum of trees will be called a <u>forest</u>. Then forests have the following properties: every finite sequence, finite bag, or tree is a forest. The sum of any two forests is again a forest. The concatenation of a singleton bag and a forest is a tree. Every forest may be built from singleton bags by means of the operators ++ and + in such a way that the left hand operand of ++ is always a singleton bag.

In the rest of this paper, we let X denote an arbitrary set and we let Co(X) denote the set of all nonempty constructs that can be built from singleton bags over X by means of ++ and + , and Fo(X) the set of all nonempty forests among these. Due to the essential uniqueness of the canonical decomposition, we may define functions on Co(X) by describing their values on singleton bags and their behaviour under ++ and + .

DEFINITION. Let $ denote a commutative, associative operator on X . Then $/ is the X-valued function on Co(X) that satisfies

```
$/([x]) = x  for all  x  in  X ,
$/(C0 ++ C1) = $/(C0) $ $/(C1)  for all  C0 , C1  in  Co(X) ,
$/(C0 + C1) = $/(C0) $ $/(C1)  for all  C0 , C1  in  Co(X) .
```

(In [1] and [2] the symbol / was introduced for finite sequences and finite bags separately. The pronunciation is "reduce" [1] or "inserted in" [2].)

Thus, if · denotes multiplication in **Z** , then

·/(<2, 3, 5>) = ·/([2] ++ [3] ++ [5]) = 2 · 3 · 5 = 30 ;

·/([2, 3, 5]) = ·/([2] + [3] + [5]) = 2 · 3 · 5 = 30 ;

·/(<2, 3> ++ [2, 3, 5])

= ·/([2] ++ [3] ++ ([2] + [3] + [5])) = 2 · 3 · 2 · 3 · 5 = 180 .

It is worth mentioning that, for f X – Y , the mapping f* introduced earlier may be defined on Co(X) by

f*([x]) = [f(x)] for all x in X ,

f*(C0 ++ C1) = f*(C0) ++ f*(C1) for all C0 , C1 in Co(X) ,

f*(C0 + C1) = f*(C0) + f*(C1) for all C0, C1 in Co(X) .

A generalization of the Bird-Meertens symbols

Comparison of the formulae given for $/ and f* on Co(X) immediately leads to the following common generalization, which turns out to be well suited for expressing various properties of trees and forests.

DEFINITION. Let X and Y be sets; assume f ∈ X → Y . Let $0 and $1 be associative operators on Y such that $1 is also commutative. Then (f, $0, $1)! denotes the mapping F ∈ Co(X) → Y that satisfies

F([x]) = f(x) for x ∈ X ,

F(C0 ++ C1) = F(C0) $0 F(C1) for C0, C1 ∈ Co(X) ,

F(C0 + C1) = F(C0) $1 F(C1) for C0, C1 Co(X) .

EXAMPLES. (a) ([_], ++, +)! = $id_{Co(X)}$.

(b) (id_X, $1, $1)! = $1/ . For instance, ($id_X$, +, +)!(C) is the sum of all values in C , and (id_X, max, max)!(C) is the maximum of all values in C .

(c) ([_] ° f, ++, +)! = f* . Together, (b) and (c) justify our claim that ! is a generalization of / and * .

(d) For p ∈ X → Bool we define

$$p^{\sim}(x) = \underline{if}\ p(x) \rightarrow [x] \ \square\ \neg\ p(x) - [\,]\ \underline{fi} \quad .$$

Then (p˜, ++, +)!(C) is obtained from C by leaving out all values that do not satisfy p . For instance, if p = even , applying (p˜, ++, +)! to the tree

```
              1
         6          2
      3      5    0   6
         2   4  2   4
```

yields the forest

```
         6          2
         2   4  2   4   0   6
```

Generalizing a notation introduced by Bird and Meertens for sequences and bags, we propose to write (p˜, ++, +)! = p◁ .

(e) Note that (f, \$0, \$1)! = (id_Y, \$0, \$1)! ° f* . Hence, at the price of slightly longer formulae, we might have limited the new notation to the case where f is the identity mapping.

(f) ([_], +, +)!(C) is the bag of elements of C . For instance, if C is the tree in Example (d), then ([_], +, +)!(C) is the bag [0, 1, 2, 2, 2, 3, 4, 4, 5, 6, 6].

(g) (1, +, +)!(C) is the number of values in C .

In the examples that follow, we assume C to be a forest. A root path of C is the sequence of values occurring on a path from a root to a leaf of C .

(h) The number (1, +, <u>max</u>)!(C) is the height of C , i.e., the maximal number of nodes on any root path. Similarly, (id_X, +, <u>max</u>)!(C) is the maximal sum of the values on any root path.

(i) (1, +, <u>min</u>)!(C) is the minimal number of nodes on any root path and (id_X, +, <u>min</u>)!(C) is the minimal sum of the values on any root path.

(j) (1, <u>max</u>, +)!(C) is the number of leaves in C .

(k) $(1, \underline{min}, +)!(C)$ is the number of roots in C .

(l) Let the operator $\leftarrow$ be defined on X by $a \leftarrow b = a$. Then $(id_X, \leftarrow, +)!(C)$ denotes the sum of the values in the roots of C .

(m) Similarly, if $\rightarrow$ is defined by $a \rightarrow b = b$, then $(id_X, \rightarrow, +)!(C)$ denotes the sum of the values in the leaves of C .

(n) $(1, \rightarrow, +)!(C)$ is the number of leaves in C (cf. (j)).

(o) $(1, \leftarrow, +)!(C)$ is the number of roots in C (cf. (k)).

(p) $([\_], \rightarrow, +)!(C)$ is the bag consisting of the values in the leaves of C . Similarly, $([\_], \leftarrow, +)!(C)$ is the bag consisting of the values in the roots of C .

(q) $(id_X, \leftarrow, \underline{max})!(C)$ is the maximum value in any root of C . Similarly, $(id_X, \rightarrow, \underline{min})!(C)$ is the minimum value in any leaf of C .

(r) $(\{\_\}, \cup, \cup)!(C)$ is the set of all values in C .

(s) $(\{\_\}, \rightarrow, \cup)!(C)$ is the set of values in the leaves of C .

(t) $(\{\_\}, \cap, \cup)!(C)$ is the set of values x with the property that there exists a root path of C containing only the value x .

(u) $(\{\_\}, \cup, \cap)!(C)$ is the set of values that occur on all root paths of C .

(v) For $x \in X$, we define

$$\chi_x(y) = \underline{if}\ y = x \rightarrow 1\ \square\ y \neq x \rightarrow 0\ \underline{fi} \quad .$$

Then $(\chi_x, +, +)!(C)$ is the number of times that x occurs in C .

(w) $(\chi_x, \rightarrow, +)!(C)$ is the number of times that x occurs in a leaf of C . Replacing $\rightarrow$ by $\leftarrow$ gives the same thing for a root.

(x) Let $\uparrow$ be a commutative, associative operator on the set of finite sequences over X such that $S \uparrow T \in \{S, T\}$ and

$$\#(S \uparrow T) = (\#S)\ \underline{max}\ (\#T) \quad ,$$

where $\#S$ denotes the length of a sequence S . (The existence of such an operator follows from the axiom of choice.) Then $([\_], ++, \uparrow\_)!(C)$ is a root path of maximal length.

(y) $([\_], \leftarrow, \uparrow)!(C)$ is a singleton bag containing a root value of C . Similarly, $([\_], \rightarrow, \uparrow)!(C)$ contains a leaf value.

(z) Let @ denote a commutative, associative operator on the set of finite sequences over X such that $S\ @\ T \in \{S ++ T, T ++ S\}$ for all S and T . (Again, such an operator exists by the axiom of choice.) Then $([\_], ++, @)!C$ is a pre-order traversal of C .

The following property of the symbol ! can be used to transform specifications into efficient functional programs.

THEOREM. Let £0 , £1 be associative operators on X and let $0 , $1 be associative operators on Y . Assume that £1 and $1 are commutative and that f ∈ X → Y satisfies

(2) f(x0 £0 x1) = f(x0) $0 f(x1) ,

f(x0 £1 x1) = f(x0) $1 f(x1)

for all x0, x1 in X . Then, for g ∈ W → X ,

(3) (f ° g, $0, $1)! = f ° (g, £0, £1)! .

PROOF. By induction on the canonical decomposition of the argument. Base:

```
  (f ° (g, £0, £1)!)([w])
= {definition of ! }
  f(g(w))
= {definition of ! }
  (f ° g, $0, $1)!([w])    .
```

Step: assume both functions yield the same value when applied to C0 , and also when applied to C1 . Then

```
  (f ° (g, £0, £1)!)(C0 ++ C1)
= {definition of ! }
  f((g, £0, £1)!(C0) £0 (g, £0, £1)!(C1))
= {(2)}
  f((g, £0, £1)!(C0)) $0 f((g, £0, £1)!(C1))
= {induction hypothesis}
  (f ° g, $0, $1)!(C0) $0 (f ° g, $0, $1)!(C1)
= {definition of ! }
  (f ° g, $0, $1)!(C0 ++ C1)    .
```

Repeating this derivation with ++ , £0 , \$0 replaced by + , £1 , \$1 completes the induction step.

In the canonical decomposition of constructs in Fo(X) , the left hand operand of ++ is always a singleton bag. Hence, on Fo(X) a slightly stronger version of the theorem holds: equality (2) need only be satisfied for $x0 \in \text{im } g$, not necessarily for all $x0 \in X$. It is in fact this stronger version of the theorem that we shall apply.

An application: unique values on root paths

In Example (t) of the previous section, it was stated without proof that for any forest C the value of ({_}, ∩, ∪)!(C) is the set of all x with the property that there exists a root path in C that contains only x . In this section, we shall derive this formula from a formal definition of root paths.

Informally, we defined a root path as the sequence of values occurring on a path from a root to a leaf. In order to formalize this, an inductive definition seems appropriate. Clearly the only root path of [x] is [x] , and a root path of C0 + C1 is either a root path of C0 or one of C1 . But which are the root paths of C0 ++ C1 ? These are of the form P0 ++ P1 , where P0 is a root path of C0 and P1 is a root path of C1. Let us define, for sets S0 , S1 of constructs,

$$S0 +\!+\!+ S1 = \{P0 +\!+ P1 \mid P0 \in S0 \wedge P1 \in S1\} \quad .$$

Now the set of root paths of C0 ++ C1 can be found from the sets of root paths of C0 and C1 respectively by applying the operator +++ . Hence the set of root paths of a forest C may be formally defined as

(4) ({[_]}, +++, ∪)!(C) ,

where {[_]} is the mapping that has value {[x]} in x .

Next we investigate the meaning of the phrase "construct C0 contains only the value x ". Put C0 = C(f) , where $f \in A \to X$; we shall then take the phrase to mean that im f = {x} . This, however, may be transformed as follows:

$$\text{im } f = \{x\}$$

= { C0 $\in$ Co(X) , hence $A \neq \emptyset$ }

$(\underline{A}\ a\colon a \in A\colon f(a) = x)$

= {}

$(\underline{A}\ a\colon a \in A\colon x \in \{f(a)\})$

= {definition of $\cap$ }

$x \in \cap/(C(\{\_\} \circ f))$

= {definition of * }

$x \in (\cap/ \circ \{\_\}^*)(C(f))$.

Hence the phrase "construct C0 contains only x " may also be rendered by

(5) $x \in (\cap/ \circ \{\_\}^*)(C0)$.

Straightforwardly combining (4) and (5), we find that the set of all x with the property that there exists a root path in C that contains only x is

$\{x \mid (\underline{E}\ P\colon P \in (\{[\_]\}, +\!+\!+, \cup)!(C)\colon x \in (\cap/ \circ \{\_\}^*)(P))\}$

= {definition of $\cup$ }

$\{x \mid x \in\ /(\{(\cap/ \circ \{\_\}^*)(P) \mid P \in (\{[\_]\}, +\!+\!+, \cup)!(C)\})\}$

= {}

$\cup/(\{(\cap/ \circ \{\_\}^*)(P) \mid P \in (\{[\_]\}, +\!+\!+, \cup)!(C)\})$

= {definition of * }

$(\cup/ \circ (\cap/ \circ \{\_\}^*)^*)(\{P \mid P \in (\{[\_]\}, +\!+\!+, \cup)!(C)\})$

= {}

(6) $(\cup/ \circ (\cap/ \circ \{\_\}^*)^* \circ (\{[\_]\}, +\!+\!+, \cup)!)(C)$.

This may be regarded as an inefficient functional program or a specification; by means of the (strengthened) theorem of the previous section, we shall transform it into the more efficient program $(\{\_\}, \cap, \cup)!(C)$.

The shape of (6) suggests that we apply the theorem with

f, £0, £1:= $\cup/ \circ (\cap/ \circ \{\_\}^*)^*$, +++, .

Next we determine which choice to make for \$0 and \$1 . Now for any S0 and S1 such that S0 is a singleton set, say {P} ,

$$f(S0 +++ S1)$$

$= \{$definitions of f and $+++\}$

$$(\cup/ \circ (\cap/ \circ \{\_\}*)*)(\{P0 ++ P1 \mid P0 \in S0 \wedge P1 \in S1\})$$

$= \{$definition of $*\}$

$$\cup/(\{(\cap/ \circ \{\_\}*)(P0 ++ P1) \mid P0 \in S0 \wedge P1 \in S1\})$$

$= \{ *$ distributes over $++\}$

$$\cup/(\{\cap/(\{\_\}*(P0) ++ \{\_\}*(P1)) \mid P0 \in S0 \wedge P1 \in S1\})$$

$= \{\cap/(C0 ++ C1) = \cap/(C0) \cap \cap/(C1)\}$

$$\cup/(\{\cap/(\{\_\}*(P0)) \cap \cap/(\{\_\}*(P1)) \mid P0 \in S0 \wedge P1 \in S1\})$$

$= \{S0 = \{P\}\}$

$$\cup/(\{\cap/(\{\_\}*(P)) \cap \cap/(\{\_\}*(P1)) \mid P1 \in S1\})$$

$= \{$definition of $*\}$

$$(\cup/ \circ (\cap/(\{\_\}*(P)) \cap)*)(\{\cap/(\{\_\}*(P1)) \mid P1 \in S1\})$$

$= \{ \cap$ distributes over $\cup\}$

$$\cap/(\{\_\}*(P)) \cap \cup/(\{\cap/(\{\_\}*(P1)) \mid P1 \in S1\})$$

$= \{S0 = \{P\}\}$

$$(\cup/ \circ (\cap/ \circ \{\_\}*)*)(S0) \cap (\cup/ \circ (\cap/ \circ \{\_\}*)*)(S1)$$

$= \{$definition of $f\}$

$$f(S0) \cap f(S1) \quad .$$

As it is easy to see that

$$f(S0 \cup S1) = f(S0) \cup f(S1) \quad ,$$

we can now use the theorem to rewrite (6) in the form

$$(7) \qquad (\cup/ \circ (\cap/ \circ \{\_\}*)* \circ \{[\_]\}, \cap, \cup)!(C) \quad .$$

However,

$$(\cup/ \circ (\cap/ \circ \{\_\}*)* \circ \{[\_]\})(x)$$

$= \{$definition of $\{[\_]\}\}$

$$(\cup/ \circ (\cap/ \circ \{\_\}*)*)(\{[x]\})$$

$= \{$definition of $*\}$

∪/({(∩/ ° {_}*)([x])})

= {definition of * }

∪/{(∩/([{x}]))}

= {definition of / }

∪/({{x}})

= {definition of / }

{x}

= {definition of {_} }

{_}(x) .

Therefore

∪/ ° (∩/ ° {}*)* ° {[_]} = {_}

and so (7) reduces to

({_}, ∩, ∪)!(C)

as promised.

<u>Another application: the maximal sum of a subtree</u>

The concept of a nonempty subtree (of a forest) may be defined recursively as follows. The only nonempty subtree of [x] is [x] itself. A subtree of [x] ++ C1 is either a subtree of C1 or the tree [x] ++ C1 . A subtree of C0 + C1 is either a subtree of C0 or one of C1 . Hence, the set of all nonempty subtrees of a forest C over **Z** is St(C) , where St Fo(**Z**) - P(Fo(**Z**)) is defined by

St([x]) = {[x]} ,

St(C0 ++ C1) = {C0 ++ C1} St(C1) ,

St(C0 + C1) = St(C0) St(C1) .

The definition of St is not one of the sort that can be denoted by ! , since the right hand side of the second line mentions C0 and C1 , not just St(C0) and St(C1). Note that this is an essential objection, since St is not injective: a forest consisting of two copies of a tree has the same set of subtrees as that tree itself. However, there is a way out of this difficulty: we "strengthen the invariant" by computing the pair (C, St(C)) rather than just St(C) . Define £0 , £1 on $(Fo(\mathbf{Z}) \times \mathcal{P}(Fo(\mathbf{Z}))$ by

$$(C0, S0)\ £0\ (C1, S1) = (C0 ++ C1, \{C0 ++ C1\} \cup S1) \quad ,$$
$$(C0, S0)\ £1\ (C1, S1) = (C0 + C1, S0 \cup S1)$$

and $g \in \mathbf{Z} \rightarrow (Fo(\mathbf{Z}) \times \mathcal{P}(Fo(\mathbf{Z}))$ by

$$g(x) = ([x], \{[x]\}) \quad ;$$

then the above calculation shows that

$$(g, £0, £1)!(C) = (C, St(C))$$

for any C in $Fo(\mathbf{Z})$. Hence

(8) $St = snd \circ (g, £0, £1)!$,

where snd(x, y) = y .

Now let us consider the task of computing the maximal sum of any nonempty subtree of a given forest C . With the aid of (8) this may be written as

(9) $(\underline{max} \circ +/* \circ snd \circ (g, £0, £1)!)(C)$.

Again, (9) may be viewed as a specification or an inefficient program and it is our task to transform it into an equivalent but more useful expression. The shape of (9) suggest that we apply the theorem with

$$f := \underline{max} \circ +/* \circ snd \quad .$$

We now determine the proper choice for \$0 and \$1 .Under the hypothesis that (C0, S0) im g , hence for some $x \in \mathbf{Z}$ both C0 = [x] and S0 = {[x]} , we have

$$f((C0, S0)\ £0\ (C1, S1))$$

= {definitions of f and £0 }

$$(\underline{max}/ \circ +/* \circ \mathrm{snd})(C0 ++ C1, \{C0 ++ C1\} \cup S1)$$

= {definition of snd }

$$(\underline{max}/ \circ +/*)(\{C0 ++ C1\} \cup S1)$$

= {definition of * }

$$\underline{max}/(\{+/(C0 ++ C1)\} \cup +/*(S1))$$

= {+/(C0 ++ C1) = +/(C0) + +/(C1) , by the definition of / }

$$\underline{max}/\{+/(C0) + +/(C1)\} \cup +/*(S1))$$

= { $\underline{max}/(A \cup B) = \underline{max}/(A)\ \underline{max}\ \underline{max}/(B)$, by the definition of / }

$$\underline{max}/(\{+/(C0) + +/(C1)\})\ \underline{max}\ \underline{max}/(+/*(S1))$$

= { C0 = [x] , by hypothesis}

$$(x + +/(C1))\ \underline{max}\ \underline{max}/(+/*(S1))$$

= { S0 = {[x]} , by hypothesis}

$$(\underline{max}/(+/*(S0)) + +/(C1))\ \underline{max}\ \underline{max}/(+/*(S1))$$

= {definition of f }

$$(f(C0, S0) + +/(C1))\ \underline{max}\ f(C1, S1) \quad .$$

The application of the theorem presents problems since we do not succeed in expressing $+/(C1)$ in terms of $f(C0, S0)$ and $f(C1, S1)$. Once again, the solution comes from "strengthening the invariant" and computing

$$F(C, S) = (+/(C), f(C, S))$$

instead of $f(C, S)$. This is sufficient for the original problem, since (9) is equal to

$$(\mathrm{snd} \circ F \circ (g, £0, £1)!)(C) \quad .$$

From the above calculation it follows that if we define \$0 on $\mathbf{Z} \times \mathbf{Z}$ by

$$(x0, y0)\ \$0\ (x1, y1) = (x0 + x1, (y0 + x1)\ \underline{max}\ y1) \quad ,$$

we have (under the hypothesis mentioned)

$$F((C0, S0)\ £0\ (C1, S1)) = F(C0, S0)\ \$0\ F(C1, S1) \quad .$$

Moreover,

$$f((C0,\ S0)\ \pounds 1\ (C1,\ S1))$$
$$= \{\text{definitions of } f \text{ and } \pounds 1\}$$
$$(\underline{max} / \circ +/* \circ snd)(C0 + C1,\ S0 \cup S1)$$
$$= \{\text{definition of } snd\}$$
$$(\underline{max}/ \circ +/*)(S0 \cup S1)$$
$$= \{ * \text{ distributes over } \cup \}$$
$$\underline{max}/(+/*(S0) \cup +/*(S1))$$
$$= \{ \underline{max}/(A \cup B) = \underline{max}/(A)\ \underline{max}\ \underline{max}/(B)\ ,\ \text{by the definition of } / \}$$
$$(\underline{max}/ \circ +/*)(S0)\ \underline{max}\ (\underline{max}/ \circ +/*)(S1)$$
$$= \{\text{definition of } f\}$$
$$f(C0,\ S0)\ \underline{max}\ f(C1,\ S1) \quad .$$

Hence, if we define $\$1$ on $\mathbf{Z} \times \mathbf{Z}$ by

$$(x0,\ y0)\ \$1\ (x1,\ y1) = (x0 + x1,\ y0\ \underline{max}\ y1) \quad ,$$

we have

$$F((C0,\ S0)\ \pounds 1\ (C1,\ S1)) = F(C0,\ S0)\ \$1\ F(C1,\ S1) \quad .$$

Application of the (strengthened) theorem now gives

$$F \circ (g,\ \pounds 0,\ \pounds 1)! = (F \circ g,\ \$0,\ \$1)! \quad .$$

For any $x \in \mathbf{Z}$ we have

$$(F \circ g)(x)$$
$$= \{\text{definitions of } F \text{ and } g\}$$
$$(+/[x],\ (\underline{max}/ \circ +/* \circ snd)([x],\ \{[x]\}))$$
$$= \{\text{definitions of } / \text{ and } snd\}$$
$$(x,\ (\underline{max}/ \circ +/*)(\{[x]\}))$$
$$= \{\text{definition of } *\}$$
$$(x,\ \underline{max}/(\{+/([x])\}))$$

= {definition of / }

(x, x) .

Gathering together our results, we find that the maximal sum of any nonempty subtree of C is (snd ° (H, $0, $1)!)(C) , where

H(x) = (x, x) ,

(x0, y0) $0 (x1, y1) = (x0 + x1, (y0 + x1) max y1) ,

(x0, y0) $1 (x1, y1) = (x0 + x1, y0 max y1) .

References

[0] A. Bijlsma, A unified approach to sequences, bags, and trees. Computing Science Notes 88/13. Eindhoven University of Technology, 1988.

[1] R.S. Bird, An introduction to the theory of lists; in: M. Broy (ed.), Logic of programming and calculi of discrete design (lecture notes international summer school, Marktoberdorf, 1986). NATO Adv. Sci. Inst. Ser. F, 36. Springer-Verlag, Berlin, 1987.

[2] L. Meertens, Algorithmics: towards programming as a mathematical activity; in: J.W. de Bakker, M. Hazewinkel, J.K. Lenstra (eds.), Mathematics and computer science. North Holland, Amsterdam, 1986.

March, 1989