Synthesizing Programs over Recursive Data Structures

Ankur Gupta
Jayadev Misra[*]
The University of Texas at Austin
Austin, Texas 78712, USA
email: misra@cs.utexas.edu

August 22, 2003

Dedicated to the memory of
Edsger Wybe Dijkstra, 1930 – 2002

0 Abstract

This paper describes a methodology and its theoretical basis for synthesizing a class of programs which operate on recursive data structures. The methodology has appeared in an earlier paper by the author [7]. This paper suggests a theoretical basis for the methodology. The theory rests on some elementary results in fixed point theory over lattices. Most of the required mathematics is developed in the paper.

1 Informal Description and Overview

This paper describes a methodology and its theoretical basis for synthesizing a class of programs which operate on recursive data structures. The methodology has appeared in an earlier paper by the author [7]. This paper suggests a theoretical basis for the methodology. The theory rests on some elementary results in fixed point theory over lattices. Most of the required mathematics is developed in the paper.

To motivate the problem and its solution, we describe a few example problems and develop their solutions using the proposed methodology. For this paper, the data structure we consider is finite linear sequence (also called a *string*) over some alphabet.

[*]Work partially supported by the National Science Foundation grant CCR–0204323.

Computing the maximum and the second maximum First, consider the problem of computing the maximum (max) of a nonempty finite sequence of integers. The function can be computed by a left to right scan of the input sequence. This is because for any sequence xe —whose last element is e and the remaining prefix x— max(xe) can be computed from max(x) and e. That is, there is a function, h, such that $\max(xe) = h(\max(x), e)$, for all x and e (we take the max of the empty string to be $-\infty$). The proposed computation exploits the recursive structure of the input, by first computing the function over the prefix, and then combining the result with the last element, e.

Next, consider the problem of computing the second largest number (max2) in a finite sequence of integers having at least two elements. It is no longer possible to apply the left-to-right computation strategy directly, because max2(xe) is *not* a function of max2(x) and e. Therefore, we must compute a more *general* function g which has the property that (1) max2(x) can be computed from $g(x)$, for every x, (2) $g(xe)$, for every x and e, can be computed from $g(x)$ and e, and (3) g is the least generalization of max2, in the sense that it requires the smallest amount of additional computation. For max2, the desired generalization is the pair of functions (max, max2), whose values can be computed from left to right over the sequence, and from which the value of max2 can be extracted.

Inductive Functions Function f which has the property that $f(xe)$ is a function of $f(x)$ and e, for all x and e, is called *inductive*. More generally, the value of an inductive function over a recursive data structure can be computed from the function values over the components of the structure.

In many cases, as in max2, the given function is not inductive. A suitable inductive generalization must be found in such cases. In some sense, the generalization should be the least possible, so that the additional computation is minimized. This paper shows how the least inductive generalization can be computed.

1.1 A Methodology for Constructing Inductive Generalizations

The following methodology was proposed in Misra [7] to obtain an inductive generalization of a given function, f. Call g a *generalization* of f if $f(xe)$, for any x and e, can be computed (efficiently) from $g(x)$ and e. Let h be the *least* generalization of f; if $f = h$ then "stop", because f is inductive. Otherwise ($f \neq h$), set f to h, and iterate this step.

The methodology just described is admittedly vague, particularly, for terms such as "least", "generalization", and, even, "inductive". The purpose of this paper is to assign exact meanings to these terms and show that the least inductive generalization is the limit of the sequence of approximations, obtained by the iterations.

Before developing the theory, we show the application of the proposed methodology on a small problem which has been treated in the literature. Our method-

x	$mss(x)$	e	xe	$mss(xe)$
4 -5 1	4	3	4 -5 1 3	4
4 -5 2	4	3	4 -5 2 3	5

Table 1: mss is not inductive

ology arrives at the solution systematically and with very little effort.

1.2 Maximum Segment Sum

This problem has been popularized by Bentley [1]. Given is a finite sequence of integers. A *segment* in this sequence is a subsequence of contiguous elements; a segment may be empty. A segment sum is the sum of the elements in that segment; empty segment sum is zero. It is required to find the maximum segment sum (mss) in the given sequence.

A little thought shows that mss is not inductive: given $mss(x)$ and e, for arbitrary x and e, $mss(xe)$ cannot be computed in general. Table 1 shows two cases where $mss(x)$ and e are identical, but $mss(xe)$ are different.

The counterexample in Table 1 also guides us to the generalization that is needed to get an inductive function. To compute $mss(xe)$, it is sufficient to know (in addition to $mss(x)$ and e) the sum of the maximum segment *ending at the last element of* x. Call this quantity $mssf(x)$; for empty x, $mssf(x) = 0$. Then,

$$mss(xe) = \max(mss(x), 0, mssf(x) + e)$$

This equation can be understood as follows. The maximum segment in xe either does not include e —then, it is either the maximum segment within x or the empty segment—, or it includes e —then, it is the maximum segment ending at the last element of x (which could be empty) followed by e.

The next step is to repeat the argument with the pair of functions $(mss, mssf)$. Can both of these function values at xe be computed from those at x and e? We know the answer for mss. For $mssf$ we note,

$$mssf(xe) = \max(0, mssf(x) + e)$$

using arguments similar to those in the last paragraph. Therefore, the pair $(mss, mssf)$ is an inductive generalization of mss. Using our theory, it can be shown that this is the least inductive generalization.

1.3 Number of Iterations to Compute Inductive Generalization

The two examples, max2 in the introduction and maximum segment sum of section 1.2, are easily treated using our methodology, because the inductive generalizations can be computed with a bounded number of iterations. In general, however, the number of iterations is not bounded, and may not even be finite. In many cases, the least inductive generalization is of the form $(f_0, \ldots, f_n)$, where each f_i is a function over the argument and n is the length of the string; see the computation of the longest ascending sequence [7], the prefix function of a string, used in the Knuth-Morris-Pratt string matching algorithm [6], or the span of a sequence, treated in Goodrich and Tamassia [3, section 3.5]. Our theory shows that the least inductive generalization is the limiting value of a sequence of approximations, and the length of this sequence need not be bounded.

1.4 Overview of the Theory and Paper

In section 2, we define an order relation, $\preceq$, over the set of functions whose domains are finite sequences (over a given alphabet). Roughly, $f \preceq g$ means that f can be computed from g; more precisely, it means that $f = h \circ g$, for some function h ($\circ$ denotes function composition). Relation $\preceq$ is a preorder, not a partial order; call g and h equivalent if $g \preceq h$ and $h \preceq g$. The least inductive generalization of a function is not unique; it can be any one of a set of equivalent functions.

In order to construct a unique inductive generalization, in section 3 we look at the set of *partitions* of the domain, where each partition corresponds to an equivalence class of functions. Most of this paper is concerned with partitions. Define $r \leq s$, for partitions r and s, if $f \preceq g$ for functions f and g from the corresponding equivalence classes. Relation $\leq$ is a partial order; moreover, partitions form a lattice under this order.

Section 4 contains a definition of inductive functions (and inductive partitions) and enumeration of some of their properties. It is shown, in particular, that the inductive partitions form a sublattice.

Each step of our methodology applies a function σ mapping a partition to a partition. The definition of σ is given in section 6. We show that σ is monotonic (with respect to $\leq$) and continuous. Additionally, the inductive partitions are precisely the fixed points of σ. The least fixed point of σ at or above r, r^*, is the desired least inductive generalization of partition r; r^* is the lub of the sequence $\sigma^i(r)$, $i \geq 0$. See figure 1 for a pictorial depiction of the lattice structures and the application of the methodology.

The theory allows us to get an explicit characterization of the least inductive generalization; see section 7. Specifically, any least inductive generalization f^* of function f satisfies, for all x and y:

$$f^*(x) = f^*(y) \;\equiv\; \langle \forall z :: f(xz) = f(yz) \rangle$$

2 Function Space

We consider functions from finite sequences over a given alphabet. We define an order relation, $\preceq$, over this function space; $f \preceq g$ means $f(x)$, for any sequence x, can be deduced from $g(x)$, i.e., there is a function p so that $f = p \circ g$.

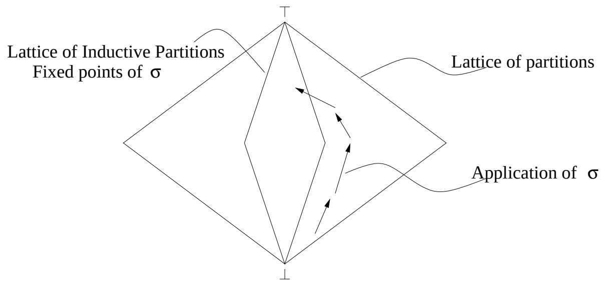

Figure 1: Schematic of the application of the methodology

Notation

Let A be an alphabet, A^* the set of finite sequences (also called *strings*) over A.
x, y, z are arbitrary elements of A^*.
f, g, h are functions whose domains are A^*.
Function composition is denoted by $\circ$.

Definition $f \preceq g$ iff $f = p \circ g$ for some function p (p is a function from the range of g to the range of f). □

Properties of $\preceq$

F2.1 These properties follow directly from the definition of $\preceq$.

1. $\preceq$ is reflexive: $f \preceq f$.
2. $\preceq$ is transitive: $f \preceq g \wedge g \preceq h \;\Rightarrow\; f \preceq h$.
3. $c \preceq f$, where c is a constant function.
4. $f \preceq id$, where $id : A^* \longrightarrow A^*$ is the identity function.
5. $f \preceq g \;\equiv\; \langle \forall x, y :: g(x) = g(y) \Rightarrow\; f(x) = f(y) \rangle$.
6. $f \preceq g \;\Rightarrow\; f \circ h \;\preceq\; g \circ h$, for any h.

3 Partition Space

Relation $\preceq$ is a preorder, not a partial order. That is, $f \preceq g$ and $g \preceq f$ does not imply that $f = g$. For instance, for two different constant functions c and d, $c \preceq d$ and $d \preceq c$. However, for our purposes, if $f \preceq g$ and $g \preceq f$, we may consider f and g equivalent, because either function may be used to compute h, where $h \preceq f$ or $h \preceq g$. So, instead of individual functions, we consider equivalence classes of functions where f and g are defined to be equivalent if $f \preceq g$ and

$g \preceq f$. These equivalence classes correspond precisely to the partitions of A^*, as explained next.

Henceforth, we use the term *partition* to mean an equivalence relation over A^*. For every function f there is a unique partition $\pi(f)$ defined by

π-definition: $x\ \pi(f)\ y \ \equiv\ f(x) = f(y)$

We say that f *induces* the partition $\pi(f)$ and $\pi(f)$ *corresponds to* f. Conversely, every partition r corresponds to some function (perhaps, many functions).

A constant function induces the *coarsest* partition —all elements of A^* are in one equivalence class— and id the *finest* —every element of A^* is in a different equivalence class. We define an order relation, $\leq$, over partitions which formalizes the notions of coarse and fine; it is closely related to the ordering over functions; $f \preceq g$ means that f induces a *coarser* partition than g.

Working with partitions, instead of functions, has the advantage that $\leq$ is a partial order. Further, the partitions form a complete lattice where $\perp$ and $\top$ correspond to a constant function and id, respectively. The lattice structure of partitions is well-known. We enumerate some of its properties, below, for completeness.

Notation

r, s, t denote partitions over A^*.
We write $f \sim g$ for $f \preceq g \ \wedge\ g \preceq f$.

Definition **(Ord)** $r \leq s \ \equiv\ \langle \forall x, y ::\ x\ s\ y \ \Rightarrow\ x\ r\ y \rangle$.

This definition is used extensively in the proofs; so, we have given it a name, (Ord), by which we will refer to it.

3.1 Properties of $\leq$

F3.1 The following propositions follow directly from the definitions.

1. $\leq$ is a partial order, i.e., $\leq$ is reflexive, antisymmetric and transitive.
2. $\pi(f) \leq \pi(g) \ \equiv\ f \preceq g$.
   $\pi(f) = \pi(g) \ \equiv\ f \sim g$.
3. $\pi(c) \leq r$, where c is any constant function.
4. $r \leq \pi(id)$, where id is the identity function over A^*.
5. $x\ \pi(f \circ g)\ y \ \equiv\ g(x)\ \pi(f)\ g(y)$, for all x and y.

We show proofs of two of these propositions, to familiarize the reader with the proof style.

• Part(2), $\pi(f) \leq \pi(g) \ \equiv\ f \preceq g$: For arbitrary x and y,

$$
\begin{array}{ll}
 & \pi(f) \leq \pi(g) \\
\equiv & \{\text{definition of } \leq \text{ from (Ord)}\} \\
 & \langle \forall x, y ::\ x\ \pi(g)\ y \ \Rightarrow\ x\ \pi(f)\ y \rangle \\
\equiv & \{\pi(g) \text{ and } \pi(f) \text{ from } \pi\text{-definition}\} \\
 & \langle \forall x, y ::\ g(x) = g(y) \Rightarrow\ f(x) = f(y) \rangle \\
\equiv & \{\text{from F2.1(part 5)}\} \\
 & f \preceq g
\end{array}
$$

- Part(5), $x\ \pi(f \circ g)\ y \ \equiv\ g(x)\ \pi(f)\ g(y)$:

$$
\begin{array}{ll}
 & g(x)\ \pi(f)\ g(y) \\
\equiv & \{\pi(f) \text{ from } \pi\text{-definition}\} \\
 & f(g(x)) = f(g(y)) \\
\equiv & \{\text{function composition}\} \\
 & (f \circ g)(x) = (f \circ g)(y) \\
\equiv & \{\pi(f \circ g) \text{ from } \pi\text{-definition}\} \\
 & x\ \pi(f \circ g)\ y
\end{array}
$$

Tuples of Functions For a pair of functions f and g, (f, g) is the function whose value is the pair $(f(x), g(x))$ for argument x. The following facts can be proved easily, use the definition of (f, g) for part(1); use part(1) and F3.1(part 2) for part(2).

F3.2 Facts about tuples:

1. $\pi(f, g) = \pi(f) \sqcup \pi(g)$, *i.e.*,
   $x\ \pi(f, g)\ y \ \equiv\ x\ \pi(f)\ y \ \wedge\ x\ \pi(g)\ y$
2. $f \preceq g \ \wedge\ u \preceq v \ \Rightarrow\ (f, u) \preceq (g, v)$

3.2 Lattice Structure of the Partition Space

Partition s is an upper bound of the set of partitions J if for every r in J, $r \leq s$. As usual, s is the least upper bound (*lub*) if $s \leq u$ for every upper bound u. Define the greatest lower bound (*glb*) analogously. We write the lub of J as $\sqcup J$ and the glb as $\sqcap J$. We show that any set of partitions has a lub and a glb; hence, the partitions constitute a complete lattice[2].

Notation $\sqcup$ and $\sqcap$ have the highest binding power over all functions.

Let J^* be the set of finite sequences over J. Any P in J^*, called a *path*, is either empty or of the form rQ, where r is a partition in J and Q is a path. We define $x\ P\ y$ for all x and y:

$$
\begin{array}{lll}
x\ P\ y & \equiv & x = y, \text{ if } P \text{ is empty}, \\
x\ rQ\ y & \equiv & \langle \exists z :: x\ r\ z \ \wedge\ z\ Q\ y \rangle
\end{array}
$$

F3.3 The relations u and v, defined below, are $\sqcup J$ and $\sqcap J$, respectively.

$$\begin{aligned} &\langle \forall x, y :: x\ u\ y \ \equiv\ (\forall r : r \in J : x\ r\ y)\rangle \\ &\langle \forall x, y :: x\ v\ y \ \equiv\ (\exists P : P \in J^* : x\ P\ y)\ \rangle \end{aligned}$$

Proof: See Appendix A. $\square$

It follows from (F3.3) that

$$\sqcup\{\} = \bot,\ \ \sqcap\{\} = \top$$

4 Inductive Functions and Partitions

The material developed in the last two sections provides the basis for definitions of inductive functions and partitions. In sections 4.1 and 4.2, we define *extensions* of functions and partitions, which are used in the subsequent definitions. In section 4.3, we define inductive functions and partitions, and note some of their properties. We show in section 5.2 that inductive partitions form a complete lattice.

Notation and Terminology

ϵ is the empty sequence.
e denotes a symbol in A, and 'e' is the sequence containing e.
Any sequence over A is either ϵ or a sequence x (possibly ϵ) followed by a symbol e. This sequence is written as xe.

4.1 Extensions of Functions

Definition For any $f : A^* \longrightarrow R$, define its *extension* $f' : A^* \longrightarrow R \times A^*$ by

- $f'(\epsilon) = (f(\epsilon), \epsilon)$.
- $f'(xe) = (f(x), \text{'}e\text{'})$.

We give a more elaborate definition of extension, which is useful in establishing its properties. We define two functions, *pre* and *last* on sequences, as follows.

$$\begin{array}{lll} pre :\ A^* \longrightarrow A^*, & \text{and} & last :\ A^* \longrightarrow A^* \\ pre(\epsilon) = \epsilon, & \text{and} & last(\epsilon) = \epsilon \\ pre(xe) = x, & \text{and} & last(xe) = \text{'}e\text{'} \end{array}$$

Then,

$$f' = (f \circ pre, last)$$

Note that $last(x) = \epsilon \ \Rightarrow\ x = \epsilon$, and $\langle x = y\rangle \ \equiv\ \langle pre(x) = pre(y)\ \wedge\ last(x) = last(y)\rangle$.

F4.1 $f \preceq g \;\Rightarrow\; f' \preceq g'$

Proof:

$$\begin{array}{ll}
 & f' \\
= & \{\text{definition}\} \\
 & (f \circ pre, last) \\
\preceq & \{f \preceq g \;\Rightarrow\; f \circ pre \preceq g \circ pre, \text{ from (F2.1, part 6). Also, } last \preceq last. \\
 & \quad \text{Apply (F3.2(part 2))}\} \\
 & (g \circ pre, last) \\
= & \{\text{definition}\} \\
 & g'
\end{array}$$

□

Corollary $f \sim g \;\Rightarrow\; f' \sim g'$ □

4.2 Extensions of Partitions

Definition The extension of partition r, written as r', is defined by:

$$r = \pi(f) \;\equiv\; r' = \pi(f')$$

We show that this is a valid definition, i.e., if $r = \pi(f)$ and $r = \pi(g)$, then $r' = \pi(f')$ and $r' = \pi(g')$. This amounts to proving that $\pi(f) = \pi(g) \;\Rightarrow\; \pi(f') = \pi(g')$, which is shown as a corollary to F4.2 below.

F4.2 $\pi(f) \leq \pi(g) \;\Rightarrow\; \pi(f') \leq \pi(g')$

Proof:

$$\begin{array}{ll}
 & \pi(f) \leq \pi(g) \\
\equiv & \{\text{from F3.1(part 2)}\} \\
 & f \preceq g \\
\Rightarrow & \{\text{from F4.1}\} \\
 & f' \preceq g' \\
\equiv & \{\text{from F3.1(part 2)}\} \\
 & \pi(f') \leq \pi(g')
\end{array}$$

□

By using symmetry, $\pi(g) \leq \pi(f) \;\Rightarrow\; \pi(g') \leq \pi(f')$, and using the antisymmetry of $\leq$,

Corollary $\pi(f) = \pi(g) \;\Rightarrow\; \pi(f') = \pi(g')$ □

The following proposition is a rewriting of F4.2 which makes explicit the monotonicity of the $'$ operator.

F4.3 $r \leq s \;\Rightarrow\; r' \leq s'$ □

F4.4 For arbitrary x and y, and partition r,

$$x\ r'\ y \ \equiv\ \langle pre(x)\ r\ pre(y)\rangle \ \wedge\ \langle last(x) = last(y)\rangle$$

Proof: Let f be such that $r = \pi(f)$.

$$\begin{array}{ll}
 & x\ r'\ y \\
\equiv & \{r' = \pi(f')\} \\
 & x\ \pi(f')\ y \\
\equiv & \{f' = (f \circ pre, last)\} \\
 & x\ \pi(f \circ pre, last)\ y \\
\equiv & \{\text{from F3.2(part 1)}\} \\
 & x\ \pi(f \circ pre)\ y\ \wedge\ x\ \pi(last)\ y \\
\equiv & \{\text{from F3.1(part 5)}\} \\
 & pre(x)\ \pi(f)\ pre(y)\ \wedge\ x\ \pi(last)\ y \\
\equiv & \{r = \pi(f) \text{ and } x\ \pi(last)\ y \text{ is } last(x) = last(y)\} \\
 & pre(x)\ r\ pre(y)\ \wedge\ last(x) = last(y)
\end{array}$$

□

The following proposition is a direct consequence of F4.4.

F4.5 $(\ \forall x, y ::\ x\ r\ y\ \equiv\ \langle \forall e ::\ xe\ r'\ ye\rangle\ )$

4.3 Definitions of Inductive Functions and Partitions

Definition Function f is *inductive* iff $f \preceq f'$. Partition r is *inductive* iff $r \leq r'$.

The definition of inductive function is motivated by the following considerations. Function f's value can be computed by a left to right scan of its argument string iff for any x and e, $f(xe)$ is a function of $f(x)$ and e, i.e., $f(xe) = g(f(x), \text{‘}e\text{’})$, for some g. We show that in this case f is inductive.

Suppose $f : A^* \longrightarrow R$, and there is a function $g : R \times A^* \longrightarrow R$ such that $f(xe) = g(f(x), \text{‘}e\text{’})$, for all x and e. Define $h : R \times A^* \longrightarrow R$ by

$$h(u, v) = \begin{cases} u & \text{if} \quad v = \epsilon \\ g(u, v) & \text{if} \quad v \neq \epsilon \end{cases}$$

Then,

$$\begin{array}{ll}
 & h(f'(\epsilon)) \\
= & \{\text{definition of } f'\text{: } f'(\epsilon) = (f(\epsilon), \epsilon)\} \\
 & h(f(\epsilon), \epsilon) \\
= & \{\text{definition of } h\} \\
 & f(\epsilon)
\end{array}$$

And,

$$\begin{array}{ll}
 & h(f'(xe)) \\
= & \{\text{definition of } f'\text{: } f'(xe) = (f(x), \text{‘}e\text{’})\} \\
 & h(f(x), \text{‘}e\text{’}) \\
= & \{\text{definition of } h\} \\
 & g(f(x), \text{‘}e\text{’}) \\
= & \{\text{definition of } g\} \\
 & f(xe)
\end{array}$$

In all cases, therefore, $h(f'(x)) = f(x)$. Hence, $f \preceq f'$, i.e., f is inductive.

A similar proof shows that an inductive function's value can be computed by a left to right scan of its argument. More generally, if $f \preceq g'$, then $f(xe)$ is a function of $g(x)$ and e.

F4.6 Let $f \sim g$. Then,

$$f \text{ inductive } \equiv g \text{ inductive}$$

Proof: We show that f inductive $\Rightarrow$ g inductive. The converse follows by symmetry.

$$\begin{array}{ll}
 & f \text{ inductive} \\
\Rightarrow & \{\text{definition of inductive}\} \\
 & f \preceq f' \\
\Rightarrow & \{\text{from } f \sim g,\ g \preceq f\} \\
 & g \preceq f' \\
\Rightarrow & \{\text{from } f \sim g,\ f \preceq g, \text{ and applying F4.1, } f' \preceq g'\} \\
 & g \preceq g' \\
\Rightarrow & \{\text{definition of inductive}\} \\
 & g \text{ inductive}
\end{array}$$

F4.7 f inductive $\equiv$ $\pi(f)$ inductive

Proof:

$$\begin{array}{ll}
 & f \text{ inductive} \\
\equiv & \{\text{definition of inductive}\} \\
 & f \preceq f' \\
\equiv & \{\text{from F3.1(part 2)}\} \\
 & \pi(f) \leq \pi(f') \\
\equiv & \{\text{definition of inductive partition}\} \\
 & \pi(f) \text{ inductive}
\end{array}$$

□

5 Intension and Continuity

By now we have set up the machinery to address the main topic of this paper. The machinery consists of partitions, ordering over them, and the notions of extension and inductive partitions; the main topic is to show how the least inductive partition for a given partition can be obtained through successive approximation. The answer can be given in terms of the concepts we have defined so far. However, a considerably more elegant solution is obtained by introducing another operator, called *intension* (and written as ‘), which is the lower adjoint of $'$, the extension operator. That is (intension, extension) is a Galois connection.

Using the intension operator, we prove a number of continuity results. One typical continuity result is: for a set of partitions J, let $J' = \{r' |\ r \in J\}$; then,

$\sqcap(J') = (\sqcap J)'$. The direct proof, using the definition of glb based on paths, is quite ugly. Using the intension operator, we get a very succinct proof.

For a partition r define its intension $`r$ by:

$$(\forall x, y ::\ x\ `r\ y \ \equiv\ \langle \forall e ::\ xe\ r\ ye \rangle)$$

It can be shown that $`r$ is a partition. Also,

F5.1 intension satisfies:

1. $`(r') = r$
2. $r \leq (`r)'$
3. (Monotonicity of intension) $r \leq s \ \Rightarrow\ `r \leq `s$
4. (Galois connection) $(`,')$ is a Galois connection: $`r \leq s \ \equiv\ r \leq s'$

Proof of F5.1, part(1); $`(r') = r$: For any x and y

$$\begin{array}{ll}
 & x\ `(r')\ y \\
\equiv & \{\text{definition of } `\} \\
 & \langle \forall e ::\ xe\ r'\ ye \rangle \\
\equiv & \{\text{from F4.5}\} \\
 & x\ r\ y
\end{array}$$

$\Box$

Proof of F5.1, part(2); $r \leq (`r)'$: For any x and y

$$\begin{array}{ll}
 & x\ (`r)'\ y \\
\equiv & \{\text{from F4.4}\} \\
 & \langle pre(x)\ `r\ pre(y) \rangle\ \wedge\ \langle last(x) = last(y) \rangle \\
\equiv & \{\text{definition of } `r\} \\
 & \langle \forall e ::\ pre(x)e\ r\ pre(y)e \rangle\ \wedge\ \langle last(x) = last(y) \rangle \\
\Rightarrow & \{\text{If either of } x \text{ or } y \text{ is } \epsilon, \text{ both are } \epsilon, \text{ from } last(x) = last(y) \\
 & \quad \text{So, } x\ r\ y, \text{ from the reflexivity of } r. \\
 & \quad \text{If neither of } x \text{ and } y \text{ is } \epsilon, \text{ let } c \text{ be the symbol in } last(x). \\
 & \quad \text{Instantiate } e \text{ by } c \text{ to get } pre(x)c\ r\ pre(y)c. \\
 & \quad \text{That is, } x\ r\ y, \text{ because } x = pre(x)c, \text{ and } y = pre(y)c\} \\
 & x\ r\ y
\end{array}$$

$\Box$

Proof of F5.1, part(3); $r \leq s \ \Rightarrow\ `r \leq `s$: For any x and y

$$\begin{array}{ll}
 & x\ `s\ y \\
\Rightarrow & \{\text{definition of } `s\} \\
 & \langle \forall e ::\ xe\ s\ ye \rangle \\
\Rightarrow & \{r \leq s\} \\
 & \langle \forall e ::\ xe\ r\ ye \rangle \\
\Rightarrow & \{\text{definition of } `r\} \\
 & x\ `r\ y
\end{array}$$

$\Box$

Proof of F5.1, part(4); $`r \le s \;\equiv\; r \le s'$: Proof is by mutual implication.

$$\begin{array}{ll} & `r \le s \\ \Rightarrow & \{\text{monotonicity of }{}', \text{ from F4.3}\} \\ & (`r)' \le s' \\ \Rightarrow & \{r \le (`r)', \text{ from F5.1(part 2)}\} \\ & r \le s' \end{array}$$

And,

$$\begin{array}{ll} & r \le s' \\ \Rightarrow & \{\text{monotonicity of }`, \text{ from F5.1(part 3)}\} \\ & `r \le `(s') \\ \Rightarrow & \{`(s') = s, \text{ from F5.1(part 1)}\} \\ & `r \le s \end{array}$$

□

As a simple illustratation of the power of Galois connection, we strengthen F4.3. F4.1 and its corollary can be strengthened similarly.

F5.2 $r \le s \;\equiv\; r' \le s'$

Proof:

$$\begin{array}{ll} & r \le s \\ \equiv & \{\text{from F5.1, part(1), } `(r') = r\} \\ & `(r') \;\le\; s \\ \equiv & \{\text{in Galois connection, F5.1, part(4), substitute } r' \text{ for } r\} \\ & r' \le s' \end{array}$$

□

5.1 Continuity Results

We prove a number of continuity results in this section.

F5.3 Let J be a set of partitions. Let $J' = \{j' |\; j \in J\}$ and $`J = \{`j |\; j \in J\}$.

1. $\sqcap(J') = (\sqcap J)'$
2. $\sqcup(J') = (\sqcup J)'$
3. $\sqcup(`J) = `(\sqcup J)$
4. $`(\sqcap J) \le \sqcap(`J)$

Proof of F5.3, part(1); $\sqcap(J') = (\sqcap J)'$: We prove the result using the method of indirect equality; for any t, $t \le (\sqcap J)' \;\equiv\; t \le \sqcap(J')$.

$$\begin{array}{ll} & t \le (\sqcap J)' \\ \equiv & \{\text{from Galois connection, F5.1, part(4)}\} \\ & `t \le (\sqcap J) \\ \equiv & \{\text{definition of glb}\} \end{array}$$

$$
\begin{array}{ll}
& \langle \forall j:\ j \in J:\ `t \le j \rangle \\
\equiv & \{\text{from Galois connection, F5.1, part(4)}\} \\
& \langle \forall j:\ j \in J:\ t \le j' \rangle \\
\equiv & \{\text{definition of } J'\} \\
& \langle \forall i:\ i \in J':\ t \le i \rangle \\
\equiv & \{\text{definition of glb}\} \\
& t \le \sqcap(J')
\end{array}
$$

Proof of F5.3, part(2); $\sqcup(J') = (\sqcup J)'$: For any x and y

$$
\begin{array}{ll}
& x\ \sqcup(J')\ y \\
\equiv & \{\text{definition of } \sqcup\} \\
& \langle \forall j : j \in J : x\ j'\ y \rangle \\
\equiv & \{\text{from F4.4}\} \\
& \langle \forall j : j \in J : \mathit{pre}(x)\ j\ \mathit{pre}(y)\ \wedge\ \mathit{last}(x) = \mathit{last}(y) \rangle \\
\equiv & \{\text{definition of } \sqcup\} \\
& \mathit{pre}(x)\ (\sqcup J)\ \mathit{pre}(y)\ \wedge\ \mathit{last}(x) = \mathit{last}(y) \\
\equiv & \{\text{from F4.4}\} \\
& x\ (\sqcup J)'\ y
\end{array}
\qquad \Box
$$

Proof of F5.3, part(3); $\sqcup(`J) = `(\sqcup J)$: We prove the result using the method of indirect equality; for any t, $`(\sqcup J) \le t\ \equiv\ \sqcup(`J) \le t$.

$$
\begin{array}{ll}
& `(\sqcup J) \le t \\
\equiv & \{\text{from Galois connection, F5.1, part(4)}\} \\
& (\sqcup J) \le t' \\
\equiv & \{\text{definition of lub}\} \\
& (\forall j : j \in J : j \le t') \\
\equiv & \{\text{from Galois connection, F5.1, part(4)}\} \\
& (\forall j : j \in J : `j \le t) \\
\equiv & \{\text{definition of lub and } `J\} \\
& \sqcup(`J) \le t
\end{array}
$$

Proof of F5.3, part(4); $`(\sqcap J) \le \sqcap(`J)$: Omitted.

5.2 Lattice Structure over Inductive partitions

The set of all inductive partitions form a lattice. This is because the lub and glb of a set of inductive partitions is inductive, which we show below. Note that $\top$ and $\bot$ are inductive.

Let S be a set of inductive partitions. We show that $\sqcup S \le (\sqcup S)'$ and $\sqcap S \le (\sqcap S)'$. Since all elements of S are inductive, $\langle \forall s : s \in S : s \le s' \rangle$. Therefore, $\sqcup(S')$ is an upper bound for S and $\sqcap S$ is a lower bound of S'; hence, from the definitions of $\sqcup$ and $\sqcap$,

$\sqcup S \le \sqcup(S')$ and $\sqcap S \le \sqcap(S')$

From continuity, F5.3, $\sqcup(S') = (\sqcup S)'$ and $\sqcap(S') = (\sqcap S)'$. Therefore,

$\sqcup S \leq (\sqcup S)'$ and $\sqcap S \leq (\sqcap S)'$

The set of partitions of the form r', for some r, also form a complete lattice.

6 The Synthesis Methodology

The program synthesis problem treated in this paper is as follows. Given f find f^* such that (1) $f \preceq f^*$, (2) f^* is inductive, and (3) f^* is a least function satisfying (1) and (2). Observe that f^* may not be unique; any function g satisfies these requirements, provided $f^* \sim g$. However, the partition corresponding to f^* is unique (recall that $f^* \sim g \;\Rightarrow\; \pi(f^*) = \pi(g)$). Therefore, we compute the unique partition $\pi(f^*)$, as follows. Let $r = \pi(f)$. We define a function σ over partitions which is continuous. The limiting value of the sequence $\sigma^i(r)$, $i \geq 0$, is $\pi(f^*)$.

The next approximation s of r should satisfy $r \leq s \;\wedge\; r \leq s'$: first, $r \leq s$ means $f \preceq g$ (where $r = \pi(f)$ and $s = \pi(g)$), so $f(x)$ can be computed from $g(x)$; also, from $r \leq s'$, $f(xe)$ can be computed from $(g(x), e)$, for any e. Clearly, we should choose the smallest such s. So, it is reasonable to define $\sigma(r) = \sqcap R$, where $R = \{s |\; r \leq s \;\wedge\; r \leq s'\}$. We show that $\sqcap\{s |\; r \leq s \;\wedge\; r \leq s'\}$ is equivalent to $r \sqcup `r$.

$$\begin{array}{cl}
 & \sqcap\{s|\; r \leq s \;\wedge\; r \leq s'\} \\
= & \{\text{apply Galois connection, F5.1, part(4), on } r \leq s'\} \\
 & \sqcap\{s|\; r \leq s \;\wedge\; `r \leq s\} \\
= & \{\text{definition of lub}\} \\
 & \sqcap\{s|\; (r \sqcup `r) \leq s\} \\
= & \{\text{definition of glb}\} \\
 & r \sqcup `r
\end{array}$$

Definition of σ: For any r, let $\sigma(r) = r \sqcup `r$.

F6.1 (monotonicity of σ) σ is monotonic with respect to $\leq$, i.e.,

$$r \leq t \;\Rightarrow\; \sigma(r) \leq \sigma(t)$$

Proof:

$$\begin{array}{cl}
 & r \sqcup `r \\
\leq & \{r \leq t.\ \text{And, lub is monotonic}\} \\
 & t \sqcup `r \\
\leq & \{r \leq t \;\Rightarrow\; \{\text{F5.1, part(3)}\}\ `r \leq `t.\ \text{And, lub is monotonic}\} \\
 & t \sqcup `t
\end{array}$$

□

F6.2 (continuity of σ) σ is continuous, i.e., for a set of partitions S

$$\sigma(\sqcup S) = \sqcup(\sigma(S)), \text{ where } \sigma(S) = \{\sigma(r)|\; r \in S\}$$

Proof:

$$\begin{array}{cl}
 & \sqcup(\sigma(S)) \\
= & \{\text{definition of } \sigma(S)\} \\
 & \sqcup\{s:\ s\in S:\ \sigma(s)\} \\
= & \{\text{definition of } \sigma\} \\
 & \sqcup\{s:\ s\in S:\ s\sqcup `s\} \\
= & \{\text{property of } \sqcup\} \\
 & \sqcup\{s:\ s\in S:\ s\}\ \sqcup\ \sqcup\{s:\ s\in S:\ `s\} \\
= & \{\text{simplification}\} \\
 & (\sqcup S)\ \sqcup\ \sqcup(`S) \\
= & \{\text{from F5.3, part(3) } \sqcup(`S) = `(\sqcup S)\} \\
 & (\sqcup S)\ \sqcup\ `(\sqcup S) \\
= & \{\text{definition of } \sigma\} \\
 & \sigma(\sqcup S)
\end{array}$$
□

F6.3 $r \leq r'$ iff $r = \sigma(r)$

Proof:

$$\begin{array}{cl}
 & r \leq r' \\
\equiv & \{\text{Galois connection, F5.1, part(4)}\} \\
 & `r \leq r \\
\equiv & \{\text{property of lub}\} \\
 & r = r \sqcup `r \\
\equiv & \{\text{definition of } \sigma\} \\
 & r = \sigma(r)
\end{array}$$
□

Theorem 1: Let r^* be the lub of $\{\sigma^i(r) |\ i \geq 0\}$. Then, r^* is the least fixed point of σ at or above r, i.e.,

$r \leq r^*$,
$\sigma(r^*) = r^*$, and
$r^*\ =\ \sqcap\{s |\ r \leq s\ \wedge\ \sigma(s) = s\}$.

Proof: The proof is similar to the proof of the least fixed point theorem for a continuous function, see Stoy[8]. □

Each fixed point of σ is an inductive partition, from F6.3; so, we have:

Corollary: r^* is the least inductive partition at or above r. □

7 Characterization of Least Inductive Partition

In the previous section, we have characterized the least inductive partition, r^*, of r as the limit of a sequence. In this section, we give a direct characterization which can be used to verify if a given partition (or function) is indeed the least inductive generalization. It is possible that the characterization given in this section may become the basis of a synthesis procedure. For any f, let $r = \pi(f)$, and r^* be defined as in Theorem 1 and f^* be given by $r^* = \pi(f^*)$.

Theorem 2: $f^*(x) = f^*(y) \;\equiv\; \langle \forall z :: f(xz) = f(yz) \rangle$ $\square$

We establish an auxiliary result before we prove Theorem 2.

F7.1 For arbitrary x and y:

$x \; \sigma^i(r) \; y \;\equiv\; \langle \forall z : |z| \le i : xz \; r \; yz \rangle$, where $|z|$ is the length of string z.

Proof: The proof is by induction on i.

• $i = 0$:

$$\begin{array}{ll} & \langle \forall z : |z| \le i : xz \; r \; yz \rangle \\ \equiv & \{i = 0\} \\ & x \; r \; y \\ \equiv & \{i = 0\} \\ & x \; \sigma^i(r) \; y \end{array}$$

• $i + 1$, $i \ge 0$:

$$\begin{array}{ll} & x \; \sigma^{i+1}(r) \; y \\ \equiv & \{\sigma^{i+1}(r) = \sigma^i(\sigma(r))\} \\ & x \; \sigma^i(\sigma(r)) \; y \\ \equiv & \{\text{induction hypothesis}\} \\ & \langle \forall z : |z| \le i : xz \; \sigma(r) \; yz \rangle \\ \equiv & \{\sigma(r) = r \sqcup {}^{\backprime}r\} \\ & \langle \forall z : |z| \le i : xz \; (r \sqcup {}^{\backprime}r) \; yz \rangle \\ \equiv & \{\text{definition of lub}\} \\ & \langle \forall z : |z| \le i : xz \; r \; yz \;\wedge\; xz \; {}^{\backprime}r \; yz \rangle \\ \equiv & \{\text{definition of intension}\} \\ & \langle \forall z : |z| \le i : xz \; r \; yz \;\wedge\; (\forall e :: xze \; r \; yze) \rangle \\ \equiv & \{\text{predicate calculus}\} \\ & \langle \forall z : |z| \le i + 1 : xz \; r \; yz \rangle \end{array}$$ $\square$

Proof of Theorem 2, $f^*(x) = f^*(y) \;\equiv\; \langle \forall z :: f(xz) = f(yz) \rangle$:

$$\begin{array}{ll} & f^*(x) = f^*(y) \\ \equiv & \{r^* = \pi(f^*)\} \\ & x \; r^* \; y \\ \equiv & \{r^* = \sqcup\{\sigma^i(r) |\; i \ge 0\}; \text{ definition of lub from F3.3}\} \\ & \langle \forall i : i \ge 0 : x \; \sigma^i(r) \; y \rangle \\ \equiv & \{\text{from F7.1}\} \\ & \langle \forall i : i \ge 0 : \;\; \langle \forall z : |z| \le i : xz \; r \; yz \rangle \;\; \rangle \\ \equiv & \{\text{predicate calculus}\} \\ & \langle \forall z :: xz \; r \; yz \rangle \\ \equiv & \{r = \pi(f)\} \\ & \langle \forall z :: f(xz) = f(yz) \rangle \end{array}$$ $\square$

8 Discussion

Functions over Recursive Data Structures

It is possible to generalize this theory to apply to functions over recursive data structures. An inductive function over such a structure has the property that its value for any specific structure can be computed from the function values over the components of the structure. For example, function f is inductive over binary trees if for any tree t, which has t_l, t_r and r as the left and right subtrees, and the root, respectively, $f(t)$ is a function of $f(t_l)$, $f(t_r)$ and r. The definitions of extension and inductive function (partition) are analogous to the treatment given in section 4. The results of that section need to be proven for the specific recursive data structure. The results of section 6 are almost independent of the specific data structure. The characterization given in section 7 has to be specialized for each recursive structure.

Limitations of the theory

The theory provides justification for the programming methodology described in section 1.1. It does not provide a procedure for computing the fixed point which can be implemented on a machine. In fact, it is unlikely that a present-day theorem prover can establish that a function is inductive, or show a counterexample to guide programmers in their search for function generalizations.

We defined $f \preceq g$ to be $f = p \circ g$, for some function p. In any practical situation, we want all three functions —f, g and p— to be computable. In fact, we would like p to be polynomially (even linearly) computable so that the desired value, $f(x)$ for input x, is extracted from $g(x)$ efficiently. A serious drawback of our theory is that p is not guaranteed to be computable. Recall from Page 5, F2.1(part 5), that

$$f \preceq g \;\equiv\; \langle \forall x, y :: g(x) = g(y) \Rightarrow\; f(x) = f(y) \rangle.$$

This is an important property, which has been used extensively in this paper (it is called **Ord** for partitions, in Page 6). This equivalence is destroyed if p is restricted to be computable, because there are functions f and g such that $\langle \forall x, y :: g(x) = g(y) \Rightarrow\; f(x) = f(y) \rangle$, yet there is no computable function p so that $f = p \circ g$. In particular, let g be the identity function and f be any uncomputable function.

Finally, we have considered only computations which proceed from left to right on a sequence. Therefore, a function like *quicksort* (see Knuth [5]) can not be synthesized by our methods. However, *heapsort*(see Knuth [5]), where the heap is built by scanning the sequence from left to right, is inductive, and can, possibly, be synthesized by our scheme.

A Appendix: Proof of F3.3

The statement of F3.3 is: The relations u and v, defined below, are $\sqcup J$ and $\sqcap J$, respectively.

$$\begin{array}{l} \langle \forall x,y :: x\ u\ y \ \equiv\ (\forall r : r \in J : x\ r\ y)\rangle \\ \langle \forall x,y :: x\ v\ y \ \equiv\ (\exists P : P \in J^* : x\ P\ y)\ \rangle \end{array}$$

First, we show that u and v are partitions, i.e., equivalence relations over A^*. For u, it follows from the fact that conjunction of equivalence relations is an equivalence relation. For the proof of v, define a *graph* of J. The nodes in the graph are the elements of A^*. The edges are labeled with the elements of J (i.e., the names of the partitions in J). There is an undirected edge labeled r between x and y, for any r in J, provided $x\ r\ y$. Then, $x\ v\ y$ holds provided x and y belong to the same connected component of the graph of J. Hence, v is an equivalence relation.

Next, we show that $u = \sqcup J$. For any r, $r \in J$, $x\ u\ y \ \Rightarrow\ x\ r\ y$, from the definition of u. So, $r \le u$; that is, u is an upper bound. To see that u is the lub, consider any upper bound t; we show $u \le t$.

$$\begin{array}{ll} \langle \forall x,y :: x\ t\ y \ \Rightarrow\ (\forall r : r \in J : x\ r\ y)\rangle & \text{, (Ord) and that } t \text{ is an upper bound} \\ \langle \forall x,y :: x\ t\ y \ \Rightarrow\ x\ u\ y\rangle & \text{, from above and definition of } u \\ u \le t & \text{, (Ord)} \end{array}$$

Now, we show that v is the glb of J. Note that J^* is never empty; if J is the empty set, the only path in J^* is the empty path; so, v is the identity partition.

First, we show that v is a lower bound of J, i.e., for any r, $r \in J$, $v \le r$. From (Ord), we need to show that for arbitrary x and y, $x\ r\ y \ \Rightarrow\ x\ v\ y$.

$$\begin{array}{ll} & x\ r\ y \\ \Rightarrow & \{\text{let } P \text{ be the sequence that consists only of } r\} \\ & x\ P\ y \\ \Rightarrow & \{\text{definition of } x\ v\ y\} \\ & x\ v\ y \end{array}$$

We show that v is the glb, as follows: for any lower bound t of J, $x\ v\ y \ \Rightarrow\ x\ t\ y$, for any x and y; therefore, $t \le v$, from (Ord). From the definition of v, $x\ v\ y$ implies that there is a path P between x and y. The proof is by induction on n, the length of P.

- $n = 0$:

$$\begin{array}{ll} & x\ v\ y \\ \Rightarrow & \{P \text{ connects } x \text{ and } y;\ t \text{ is a partition, so } t \text{ is reflexive}\} \\ & x\ P\ y\ \wedge\ x\ t\ x \\ \Rightarrow & \{P \text{ is empty since its length is } 0\} \\ & x = y\ \wedge\ x\ t\ x \\ \Rightarrow & \{\text{predicate calculus}\} \\ & x\ t\ y \end{array}$$

- $n+1$:

$$
\begin{array}{ll}
 & x \; v \; y \\
\Rightarrow & \{\text{a path of the form } rQ \text{ connects } x \text{ and } y\} \\
 & x \; rQ \; y \\
\Rightarrow & \{\text{definition of path}\} \\
 & \langle \exists z :: x \; r \; z \;\wedge\; z \; Q \; y \rangle \\
\Rightarrow & \{t \leq r. \text{ Hence } x \; r \; z \;\Rightarrow\; x \; t \; z\} \\
 & \langle \exists z :: x \; t \; z \;\wedge\; z \; Q \; y \rangle \\
\Rightarrow & \{\text{induction hypothesis: length of } Q \text{ is } n\} \\
 & \langle \exists z :: x \; t \; z \;\wedge\; z \; t \; y \rangle \\
\Rightarrow & \{t \text{ is a partition, so } t \text{ is transitive}\} \\
 & \langle \exists z :: x \; t \; y \rangle \\
\Rightarrow & \{\text{predicate calculus}\} \\
 & x \; t \; y \qquad \Box
\end{array}
$$

Acknowledgment I am grateful to Tony Hoare and Phil Wadler for commenting on an earlier draft. In particular, Wadler pointed out a serious error which led to a major revision of the paper. Special thanks to Jose Meseguer for providing extraordinarily detailed comments and pointing out many interesting algebraic connections.

References

[1] John Bentley. *Programming Pearls*, chapter 8, pages 77–86. Addison-Wesley, Reading, MA, 2000.

[2] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[3] Michael T. Goodrich and Roberto Tamassia. *Data Structures and Algorithms in Java.* John Wiley and Sons, Inc., 1998.

[4] Ankur Gupta. Personal communication, 2003.

[5] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming.* Addison-Wesley, Reading, Massachusetts, 1998.

[6] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[7] Jayadev Misra. A technique of algorithm construction on sequences. *IEEE, TSE*, January 1978.

[8] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, 1981.