ON THE RELATION OF REFINEMENT
BETWEEN ALGORITHMS

-- Preliminary Report --

Robert L. Constable
TR 73-187

December 1973

Department of Computer Science
Cornell University
Ithaca, New York 14850
ON THE RELATION OF REFINEMENT BETWEEN ALGORITHMS

-- Preliminary Report --

Robert L. Constable
TR 73-187

December 1973

Department of Computer Science
Cornell University
Ithaca, New York  14850
On the Relation of Refinement between Algorithms
-- Preliminary Report --

Abstract

§1 Introduction
2 Tree schemata
trees
schemata

§3 Algebraic structure
finite sum
finite product
algebraic terms

§4 Refinement
s ≤ t, t refines s
≤ is a quasi-order

§5 Approximation
simple approximations, t_n
Scott approximations, ≤_S
≤_S is a partial order

§6 Convergence
attempts to program & non-converging chains
convergence of Scott chains

§7 Fixed points
classical fixed point theorem
Cauchy continuous functions
fixed point theorem
trees as Cauchy continuous functions
Retaining our concepts to Scott's, the semantics of flow diagrams. To concentrate in this paper on
the semantics of flow diagrams, recently employed by Scott to describe
"fixed point semantics" recently employed by Scott to describe
and certain concepts such as "approximation", "convergence" and
the "structure" of an algorithm. We obtain as a special case
us to talk about different "ways to program an algorithm" and
merit between trees (finite or infinite). These concepts allow
we derive a class of tree schemata and a notion of relation

Abstract:

Ithaca, N.Y.
Cornell University
Department of Computer Science
Robert L. Constable

--- Preliminary Report ---
On the Relation of Permutation between Algorithms

This work was supported in part by National Science Founda-

tion Grant G-799.
ON THE RELATION OF REFINEMENT BETWEEN ALGORITHMS

Robert L. Constable

§1. Introduction.

When people talk informally about programming and computing they use many interesting and apparently meaningful concepts which at present have no precise meaning. They say, "This program is easy to understand." "This Algol program and that FORTRAN program are just different ways to code the same algorithm." "This is a top down, structured, way to program." "For this problem we see an apparent trade-off between structural complexity and efficiency."

One of the goals of computing theory is the clarification and organization of these imprecise concepts that appear useful in the conduct of computing. Perhaps some of the concepts can not be made precise, and in discovering this we may learn to modify our intuitive understanding of computation. On the other hand, certain of these concepts can be made precise. In the brief history of computing theory we have given exact meaning to such concepts as, "computable", as in "this function is not computable," or "optimal" as in "this function has no optimal program," or "computational complexity" as in "the computational complexity of this parsing algorithm is asymptotically n^3."

In this paper we describe an attempt to make more exact the concept of an algorithm as distinct from a program and we attempt to give exact meaning to certain uses of the phrase
"a way to program this algorithm." The concepts we develop will also allow us to discuss certain aspects of the structural complexity of programs.

Ultimately we hope to develop these concepts to the point where we can discuss the issue of trade-off between structural and computational complexity. Furthermore we feel that these ideas will help in solving problems in computational complexity theory which appear to need new concepts, such as the problem of describing the class of natural complexity measures and feasibly computable functions.

We begin with a notion of program refinement at the level of program schemata. This will allow us to talk about the development of an algorithm by successive refinements. We examine this concept in depth and relate it to the recent work of Dana Scott on the semantics of algorithmic languages.

The connection of this work to Scott's is one of the main topics of this paper. We feel that this more general approach to the semantics of programs makes Scott's work on the lattice of flow diagrams, [16,17,18] clearer and more computationally meaningful.

The mathematics used in this paper is mainly constructive. Some of the concepts have a different meaning in classical mathematics. We will alert the reader to these differences when they arise.
This is either finite or infinite. A node with no edges leaving
\[ \text{span degree} \text{ of a node is the number of edges leaving } \]
\[ \text{in the unweighted path from } u \text{ to the root. The out-degree (or) } \]
definition 2: The depth of a node in is the number of edges
actually able to effectively enumerate nodes and edges.
Because our mathematics is constructive, we are

(1) \( T \) iff \( T \) is a tree except for property

a node, not a path. We will call an object an 

\[ \text{root x\text{thing, neither the root, nor the edges leaving } } \]
\[ \text{could find anything, neither the root, nor the edges leaving } \]
could find anything, neither the root, nor the edges leaving

Remarks: (1) A class of definition would not require that we

edges are 

defined to.

(2) There exist trees are identical if their sets of nodes and

\[ \text{Two such trees are identical if their sets of nodes and } \]
\[ \text{no edge enter. } \]

that exactly one edge enters each node except the root, \( 0, \)

which

Remark: This implies that there is never an edge <u,v>

path from the root to u.

(1) We can find for any node in except the root a unique

Realize it and we can enumerate them and

(1) Given any node in we can determine the number of edges

\[ u \text{ and a distinguished node } 0 \text{, called the root, such that } \]

\[ \text{set of pairs of nodes, } v \text{, called edges, which enter } w \text{ and } \]

\[ \text{set in of nodes (points), } (0,1,2,\ldots) \]

Definition 1: To construct a tree one must define an enumerable

that a tree is a connected acyclic directed graph.

of this concept are not constructively correct. We use the idea
Many class definitions of a tree (finite and infinite). To define these objects precisely, we need a precise definition.

Examples:

Informally, a tree schema is a tree, finite or infinite.
it is called a leaf. The set of leaves is called the frontier.

The reader can easily verify the following property of trees.

**Proposition 1:** We can enumerate all nodes of depth \( n \). If the degree of each node is finite, then the number of nodes of depth \( n \) is finite.

**Remark:** A constructive proof of this theorem should allow anyone to calculate the number of nodes of depth \( n \) given the finite degree of each node.

**Definition 3:** A tree scheme is a tree whose only nodes have degree one or two and whose degree one nodes are from the set of function variables, \( f_0, f_1, \ldots \) or tree variables \( x_0, x_1, x_2, \ldots \), and whose degree two nodes are from the set of predicate variables \( p_0, p_1, p_2, \ldots \). The left edge leaving a predicate variable is called the true exit, the right the false exit. The root node is also called the entrance or start node. The leaves are called exits.

We imagine each function as mapping a state space \( S \) to itself

\[
  f: S \to S
\]

and each predicate as mapping \( S \) to true or false

\[
  p: S \to \{ \text{true}, \text{false} \}
\]
To treat predicates with the same generality as functions we should introduce nodes of the form $\Box$. However, this would complicate our treatment of refinement (§ 4). Instead we could treat predicate evaluations similar to function evaluations and provide predicate boxes $\Box P$. Then conditionals would have the form $\begin{array}{c} \text{false} \\ \text{true} \end{array} \Box P$ where we can think of the diamond as testing the result of computing $P$.

In this paper we adopt a modification of this method. We assume that predicates can be represented by functions $f : D \to D$. This is true as long as $D$ has at least two distinct elements, say $1, T'$ which are distinguishable by some basic predicate. Then $p(x) \iff f(x) = T$. Such an approach to predicates results in tree schemes which are identical to those obtained using special predicate boxes, $\Box P$, yet avoids the necessity to distinguish predicate and function boxes. Such an approach is adequate for the results of this report.

**Definition 4**: To execute a tree scheme $t$ with input functions $f_1, \ldots, f_n, \ldots$ and predicates $p_1, p_2, \ldots, p_n, \ldots$ on state space $S$, ...
It is for example.

**Remark:** Given a tree $t$ there may be many terms which describe $t$. For the reader:

- Such that $t = t_i = t$.

**Proposition I:** For any tree $t$, if there is a finite term $\mathcal{T}$ then $\mathcal{T}$ is defined by

\[
\begin{align*}
\mathcal{T} & = (\mathcal{T}_1 t_1 + t_2) t_1 (t_3) \\
\mathcal{T} & = (t_1 t_2) t_3 (t_4) \\
\mathcal{T} & = (t_3 t_4) t_1 (t_2)
\end{align*}
\]

is denoted $t_i$ and is defined by term $t$. The tree defined by term $t$ are terms. The finite algebraic term $t$ is $t_i$ or $(t_1 t_2)$. The sum and product are defined as:

- A finite algebraic term $t$ if $t_i$ and $t_2$.
- $t_2$ is attached to $t_i$ and whose left part $t_1$ is attached to $t_i$.
- $t_2$ is the term whose root node is $p$.

Definition 2: 

- For any tree $t_i$, if $t_i$ has no exit, then

\[ (t_1 t_2) = t_i \]
By generalizing the operation, finite trees we are able to build up the same trees as we could.

Finite trees we will do the latter and it is exactly clear that on complex operations or we can force trees to have at most one exit. To handle more exits, we can either consider more.

These simple operations assume that no tree has more than one exit.

\[ T_2 \rightarrow T_1 \rightarrow p \]

and the sum of trees

\[ T_2 \rightarrow T_1 \]

Simple algebraic operations, the product of trees

set of all tree schemata. We can build finite trees using two.

Let \( T \) denote the set of finite tree schemata and \( T \) the algebraic structure.

\[ 3. \text{ Algebraic structure.} \]

If encountered as a tree variable, then execution stops (abnormal).

true and to the true exit if the test result is false. If any node

the subtree connected to the false exit if the test result is

and if the start node is a predicate test, then by deterring also

operation. This subtree is obtained by deterring the start node

execute the start node on \( S \) and then execute the subtree obtained
$$((p_1 + f_1; f_2); (p_2 + g_1, g_2)) = (p_1 + (f_1; p_2 + g_1, g_2), (f_2; p_2 + g_1, g_2))$$

§4. Refinement.

The tree variables $X_1$ denote unfinished parts of the algorithm. We can imagine a sequence of trees which represent the way an algorithm is developed. For example,
To express this relationship we want to talk about substituting trees for variables.

If $t$ is a tree containing the variables $X_{i_1}, \ldots, X_{i_n}, \ldots$ then we can display this fact by the notation $t[X_{i_1}, \ldots, X_{i_n}, \ldots]$, where we list only those variables which we want to display.

**Definition 1:** To substitute a tree $t$ for an occurrence of a variable $X$ in tree $s$, remove the node $\boxed{X}$, attach the incoming edge to the root of $t$ and attach the exit of $t$ (if $t$ has an exit) to the outgoing edge.

**Remark:** We write the result of substituting $t_2$ for $X$ in $t_1[X]$ as $t_1[t_2]$.

**Examples 1:**

1) substituting a finite tree
Proposition 1: The relation $\mathcal{E}$ is reflexive and transitive on $X$, and antisymmetric on $\mathcal{T}$.

Let $\mathcal{E}$ consider elements such as substitution. So that we can substitute different trees for different
- ... can be used for any variable and we change our notion of sub-
- ... among untyped components, so a single symbol such as

In a structurally free tree scheme, one need not do-

understood parties.

Remarks: (1) In free algorithms we have no constraints between

the variable $X$ appears more than once.

definition: A tree scheme $T$ is structurally free iff

that these components, whatever they are, must be the same.

we have an algorithm within two occurrences of $T$ and $T$.
we must substitute the same tree. Therefore, when

many different variables as one refinement, but for each occur-

The definition above us to substitute for instrumentation

places as an act of refinement.

The root is not some $x$, $x \neq x$ prevents us from simply remaining variable.

Remark (1) The condition that $T$, is not a variable and that

any $X \neq 0$. we use $\varphi$ as a special variable which can be referring to
able, when we want a unique value.

root some variable $X \neq 0$, which is not a variable and does not have as

\[ t \neq 1, 2, \ldots \] a tree $T$ which is not a variable and does not have as

Thus in $T$ and substituting for each occurrence of variable $X$,

or $T$ is obtained by selecting a subset of variables $X$. occu-
We now say how one algorithm can replace another, essentially by "filling in" underlined parts of the other. The substitution a tree with no extt...
Proof: (1) clearly $t \subseteq t$.

(2) Suppose $t_1 \subseteq t_2$ and $t_2 \subseteq t_3$. We claim $t_1 \subseteq t_3$. Suppose further that $t_2$ results by substituting $s_i$ for $X_i$, and $t_3$ results by substituting $q_j$ for $Y_j$. In each $s_i$, substitute $q_j$ for each occurrence of $Y_j$ for all $j$ (there may be no such $Y_j$). Call the resulting tree $r_i$ (which may equal $s_i$). Now for each occurrence of $Y_j$ in $t_1$ substitute $q_j$. For each occurrence of $X_i$ in $t_1$ substitute $r_i$. Call the resulting tree $t$. Clearly $t_1 \subseteq t$. We now claim $t = t_3$. We show $t$ can be obtained from $t_2$ exactly as $t_3$ is. One way to see this is to consider how $t_3$ is decomposed by the substitutions which formed it.

To see this consider the following shorthand notation:

Let $t_2$ be written as $t_1[s[...y...],Y,Z]$ which means $t_2$ arises from $t_1$ by making a number of substitutions of the form $s_i[ ]$ for $X_i$ and some of these $s_i[ ]$ may contain $Y_i$ but $t_2$ also has occurrences of $Y_i$ directly in $t_1$ along with other variables $Z$.

Then in this shorthand, $t_3$ is $t_1[s[q],q,Z]$ and $t$ is $t_1[r[ ],q,Z]$ where $r[ ] = s[q]$.

Remark: A more "tree theoretic" view of transitivity results by imagining the nodes of $t_1$ colored black ($\bullet$), those of $t_2$ not in $t_1$ colored red ($\otimes$) and those new nodes of $t_3$ not in $t_2$ colored yellow ($\bigcirc$). Now look at subtrees with red roots (the $r_i$) and those remaining in $t_1$ with yellow roots ($q_k$). It is clear how $t_3$ can be built from $t_1$ using $r_i$ and $q_k$. See figure 3.
This proves the claim of transitivity.

(3) We claim \( \leq \) is anti-symmetric on \( T_f \). That is, if \( t_1 \leq t_2 \) and \( t_2 \leq t_1 \), then \( t_1 = t_2 \). To see this, suppose that \( t_1 \) refines to \( t_2 \) via \( t \). Then if \( t_1 \neq t_2 \), \( t \) must be a tree with more than one node (since in the definition of \( \leq \), \( t_i = X_j \) is not allowed). Thus \( t \) adds a node to \( t_1 \). So \( t_1 \) has fewer nodes than \( t_2 \). But now it must be possible to go from \( t_2 \) to \( t_1 \), by substituting a tree \( s \neq X_j \). Thus \( t_2 \) has fewer nodes than \( t_1 \) which is impossible. Q.E.D.

It is not a fact that \( \leq \) is anti-symmetric on \( T \) as the following example shows.
5. Approximations.

Given an infinite tree scheme \( T \), we can imagine various ways to approximate it by finite trees. Some of these represent the usual ways to specify \( T \) as an algorithm by successive refinements. Other ways may not be the usual approximations but may be mathematically useful.

Definition 1: Let \( T^n \) be the set of all tree schemes of depth at most \( n \). Clearly \( T^n \subset T^{n+1} \).

Definition 2: Given \( t \in T \) let \( T^n T \) be the depth \( n \) tree obtained from \( t \) by replacing all nodes of depth exactly \( n \) by variables (each \( f_i \) or \( p_i \) getting a distinct variable \( X_i \) not used in \( t \)) and discarding all nodes of depth greater than \( n \). Call \( T^n T \) the simple depth \( n \) approximation of \( t \).
At this point a word is in order explaining why we must set on a quasi-ordered set. For a quasi-ordered or pre-ordered set, we call it a GO set, for thus the structure of is not a partial ordering, only if it is a partial ordering.

The idea clearly applies to trees with exits. We may for by substituting for . Then for by substituting for .

Consider the tree and let be the tree. Example 2:
Example 3:

\[ t_0 := X_0, \quad t_1 := \]

\[ t_2 := \]
Proposition 1: Let \( t_n \) be the simple depth \( n \) approximation of \( t \). Then \( t_0 \leq t_1 \leq t_2 \leq \ldots \leq t \).

Proof: Elementary.

Remark: One way to program any algorithm is to specify its simple approximations. This is one of the worst possible ways. It is bottom-up, non-structured and miserable.

When one does not already have an algorithm but is instead trying to specify it, then he may proceed by giving a sequence of ever better approximations. These are captured by the following sequences.

Definition 3: A Scott sequence \( d_0, d_1, d_2, \ldots \) is any sequence of finite tree schemes such that

(i) \( d_n \in T_n \)

(ii) \( d_n \leq d_{n+1} \)

(iii) there is no element of \( T_n \) such that \( d_n \preceq s \preceq d_{n+1} \),

that is, \( d_n \) is a maximal approximation to \( d_{n+1} \) by trees of depth \( n \).

Example 1: \( d_0 := X_1 \)

\[
\begin{align*}
  d_1 &:= X_1 + X_2 \\
  d_2 &:= X_1 + f + X_2 \\
  d_3 &:= X_1 + f + X_2 + g \\
  d_4 &:= X_1 + f + f + X_2 + g \\
  d_5 &:= X_1 + f + f + X_2 + g + g 
\end{align*}
\]

Example: Consider a Scott sequence for Example 2, §4.
no way to remove it to get back to it.

is extended to \( \sum \) and there is

\( \exists \alpha \) \( \alpha \rightarrow \sum \) then \( \exists \alpha \)
This is clear since an extension occurs at leaves. If nodes
claim: \( T_i \subseteq T_\cap T_j \) and \( S_i \subseteq T_\cap T_j \) then
\( S_i \cap T_j \cap T_i \subseteq \).
\( \gamma \subseteq \gamma \).

In an extension of \( s \) to \( r \) an extension of \( t \) into \( r \) is
substituted by a finite amount of \( \gamma \) by a finite amount.
We must show that \( S_i \subseteq T_\cap T_j \) and \( s \subseteq t \).

Proof: We must show that \( T_i \) is an ordering.

**Theorem:** Free \( T_i \) is a partially ordered (p.o.) set.

Then that of extension. We have

The mathematical structure of \( T_i \) on free trees is

depth in approximations.

\( t \subseteq s \), \( s \subseteq t \) if and only if all \( \gamma \)

**Definition:** Given a \( s \) and \( t \) say that \( s \) is

sequence and a form of approximation to initialize tree schemes.

Using simple sequences, we can extend the notion of a Scott

\[
\begin{align*}
2_X + b + t_X : & = \varepsilon_P \\
2_X + b + T_X : & = \varepsilon_P \\
2_X + T_X : & = T_P \\
T_X : & = 0_P
\end{align*}
\]

We can imagine various attempts to construct algorithms for problems which have no algorithmic solution. Perhaps the problem is only vaguely conceived and is refined and clarified as the program is written. Such attempts are represented by non-converging chains of approximations. For example

**Examples 1**: (1) \( x_1 \)
- \( x_1 + x_2 \)
- \( x_1 + f + x_2 \)
- \( x_1 + f + x_2 + g \)
- \( x_1 + f + f + x_2 + g \)
- \( x_1 + f + f + x_2 + g + g \)
- \[ \vdots \]

(2) \( x_1 + x_2 \)
- \( x_1 + x_3 + x_2 \)
- \( x_1 + x_4 + x_3 + x_2 \)
- \[ \vdots \]

In example (1) the approximations cannot converge to a tree because the programmer has not provided any exit from his iteration of \( f \). In example (2) he continues to refine the variable \( x_1 \). First he decides it must be \( x_1 + x_2 \). Then he refines \( x_1 \) again to \( x_1 + x_3 \), then \( x_1 + x_4 \), etc. This does not converge because he cannot decide how to refine that variable.

Our reaction to this situation is to look for conditions which will guarantee convergence; that is, conditions for acceptable programming. Scott in [16] takes another view. To obtain
a smoother mathematical theory, he adds limit points for all attempts to program. We consider this approach briefly for comparison.

**Definition 1**: A Scott scheme \( S \) is any sequence \( \{s_i\} \) which is a Scott chain of free finite trees. Scott approximation on such sequences is defined by

\[
\{s_i\} \sqsubseteq_S \{t_i\} \iff s_i \sqsubseteq t_i \text{ for all } i.
\]

Let \( \mathcal{S} \) be the class of Scott schemata.

We can now prove that \( \langle \mathcal{S}, \sqsubseteq_S \rangle \) is a PO set and is complete in the sense that all ascending chains have limits (but these limits may not be trees). Call such a set a CPO set; it has a unique bottom element denoted \( \perp \).

**Theorem 1**: \( \langle \mathcal{S}, \sqsubseteq_S \rangle \) is a CPO, i.e. every ascending chain has a limit, and \( \perp \), the only variable needed, is the minimum element.

**Proof**: Given a chain \( s^0 \sqsubseteq_S s^1 \sqsubseteq_S \ldots \) where each \( s^j \) is \( \langle s_i^j \rangle \) we have an array of elements

\[
\begin{align*}
s^0 & \quad \text{which is} \quad s_0^0 \sqsubseteq s_1^0 \sqsubseteq s_2^0 \sqsubseteq s_3^0 \sqsubseteq \ldots \\
\perp & \quad \perp \quad \perp \quad \perp \quad \perp \quad \perp \\
\perp & \quad \perp \quad \perp \quad \perp \\
s^1 & \quad s_0^1 \sqsubseteq s_1^1 \sqsubseteq s_2^1 \sqsubseteq s_3^1 \sqsubseteq \ldots \\
\perp & \quad \perp \quad \perp \quad \perp \\
\perp & \quad \perp \quad \perp \quad \perp \\
s^2 & \quad s_0^2 \sqsubseteq s_1^2 \sqsubseteq s_2^2 \sqsubseteq s_3^2 \sqsubseteq \ldots \\
\perp & \quad \perp \quad \perp \quad \perp \\
\vdots & \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots
\end{align*}
\]

To define the limit point \( t_i \), we must specify each component, \( t_i \). Since each tree in column \( i \) is of depth \( i \), to
root. Otherwise, leave $X_i$ as the root and let $T_i$ be the start root. If it does, say to root $T_i$ then take $f_i$ as a

their this occurrence of $X_i$ gets referred to a tree with con-

If the root of $T_0$ is a variable $X_j$, then determine what-

roof.

write as $T_i$. We build $T_i$ level by level starting at the

Proof: We show how to construct the upper bound $T_i$ which we

we can find an upper bound $T_i$ for it.

Theorem 2: If $T_0 \subseteq T_1 \subseteq T_2 \subseteq \cdots \subseteq T_i \subseteq T_{i+1}$ is a Cauchy sequence, then

able $T_i$ to another programmer, we know when he will finish his $j$-job.

the requirements. If we assign a subtask in the form of a var-

Remark. Thus a programming effort is Cauchy if we can control

or that $X_i$ is never referred beyond a certain term $T_i$ or

either that $X_i$ is referred to a tree with a non-variable root

called Cauchy iff for each occurrence of a variable $X_i$ we know

Definition 2: A sequence $T_0 \subseteq T_1 \subseteq T_2 \subseteq \cdots \subseteq T_i \subseteq T_{i+1}$ of trees is

on a term by term criterion.

with depend on the overall behavior of the sequence rather than

an ascending chain has a limit (constructively). Convergence

Returning now to the general situation we want to know when

$\omega$-chains at interior points of the structure.

which is not a tree since it contains

$\exists \theta \in T_i$. Thus $\theta$ has a limit in $\theta$. Example 2: The Scott chain $X_i$ has the

this can happen if and only if $T_i = T_{i+1}$.
To prove (3), we notice that \( x^2 \leq t \), then \( s = 0 \). For all \( t \) and \( s \), for all \( t \), for all \( t \), the term was used and \( x^2 \geq s \). For \( f > x \), we still know \( s \leq t \) because \( x^2 \) was beneath determined, and \( s^2 \) was more determined, then that more determined. For \( f > x \), we know this immediately by transitivity of \( x^2 \). Therefore, for all \( t \), we must show \( s^2 \leq t \) for all \( t \).

To prove (2), that \( s = x^2 \) is a Scott chain, we must show the assumption that \( x^2 \in \omega \{ (1+1)^x \} \)

which we know \(s = t \), \( s = t \), \( s = t \), \( s \leq t \) by transitivity of \( x^2 \). Moreover, \( s = t \) and \( t \leq x \), then for \( t \), \( s \leq t \). Clearly, \( t \leq x \) for all \( t \), so \( t \) is a good \( \tau \)-approximation to \( t \).

We must now prove that the limit, \( t \), is the limit:

\[ (3) \quad s^2 \leq t, \text{ then } t = t \]
\[ (2) \quad s \leq t, \text{ for all } t \]
\[ (1) \quad \sigma \leq t \]

We must now prove that:

Note: \( \sigma \) is variable, \( x \) remains in \( t \), then \( x \) appears in

\[ (4) \quad x \text{ variable, } x \text{ remains in } t \]

To determine all substitutions,

\[ \tau \text{ is } \leq (\tau) \text{ for all } t \]

By substitution, \( x \) for \( x \) then replace \( x \) by \( x^2 \), then replace \( x \) by \( x \), and replace \( x \) by \( t \), \( t \) is obtained from \( t \).

As follows, starting with \( m(0) = 0 \),

A non-constructive description of \( t \) is given inductively appearing in these in the column can be inserted in the more precise

determine \( t \) we look in column \( t \). We look to see it and variable
last term in which $X_i$ is refined.

To determine the successor (or successors) of the nodes known at depth $n$, let $t_{kn}$ be least term (term of smallest index) in the sequence such that all variables of depth less than or equal to $n$ have either been refined to constant roots or have ceased being refined (we know such a $t_{kn}$ because $\{t_i\}$ is Cauchy). Notice that the trees $t_{kn}$ and $t$ are identical up to depth $n$. This can be proved by induction on depth up to $n$. Look at the nodes of depth $n+1$ in $t_{kn}$ and for each occurrence of a variable $X_i$ determine whether it becomes refined to a tree with constant root or ceases to be refined. In the former case, replace $X_i$ by the constant root; in the latter case, leave $X_i$ alone. These are then the nodes of level $n+1$ in the limit tree $t$. This process defines $t$. Notice that if $s_j$ are the simple depth $j$ approximations to $t$, then $t_{kj} = s_j$.

We now claim that $t_i \subseteq t$ for all $i$. Given any $i$ there is some $j$ such that $t_i \subseteq t_{kj}$. We define an appropriate sequence of trees $s_m$ such that $t$ is obtained by extending $t_{kj}$ by substituting $s_m$ for $X_m$.

Given variable $X_m$ in $t_{kj}$, determine whether it is ever refined. If it is, say to a tree $s^1$ with root $f$, then take the root of $s^1$ as root of $s_m$. Now proceed to build a tree $s_m$ just as $t$ was built, i.e. by considering the depth 1 variables of $s^1$, substituting for them to obtain a tree $s^2$, considering the depth 2 variables of $s^2$ and substituting, etc.

The rule given describes a tree $s_m$. If $s_m$ is substituted for $X_m$, then the result produces the subtree of $t$ whose root is the location of variable $X_m$. This is because $s_m$ and $t$
are produced in the same way. Q.E.D.

It is easy to see that we cannot in general strengthen this theorem to say that \( \lim t_i \) is the least upper bound. Consider the following example.

**Example 3**: \( t := X_0 + f + g + f + g + \ldots \)

\( s := X_0 + g + f + g + f + \ldots \)

and let \( t_i := X_0 + f + \ldots + g + X_i \). Then \( t_i \) is a Cauchy sequence converging to \( t \). But also there is an \( s \) such that \( t_i \subseteq s \subseteq t \).

We can obtain a unique least upper bound if the limit tree is constant i.e. contains no tree variables.

**Corollary 1**: If \( t = \lim t_i \), \( \{t_i\} \) is Cauchy and if \( t \) is a constant, then \( t \) is the least upper bound (lub) of \( \{t_i\} \), i.e. if \( t_i \subseteq s \) for all \( i \), then \( \lim t_i = s \).

In the example above, not only \( t_i \subseteq s \subseteq t \) but also \( t \subseteq s \). Thus in some sense \( s \) and \( t \) are "equivalent." This suggests that we see whether this always happens, i.e. does \( t_i \subseteq s \subseteq \lim t_i \) imply \( \lim t_i \subseteq s ? \) If so, then we would have a non-unique least upper bound. It does hold as Lemma 1 in §7 shows.
Example 1: Consider the tree

- Is Cauchy
  - \( \{ (\bar{t}(x)) \} \)
  - \( \forall t \in T \) \( \exists \bar{t}(x) \)

- \( \{ (\bar{t}(x)) \} \)` is Cauchy, then so is \( \{ t(x) \} \)

Definition 2: A function \( f : T \rightarrow T \) is Cauchy continuous if

continuous function.

The central concept is a Cauchy fixed points can be developed for \( \forall \) which is insufficient for because \( \forall \) is not a CPO. However, a constructive treatment of the classical fixed point approach will apply in \( \forall \).

Q.E.D.

Thus \( x \in \exists x \in 0 \forall x \)

For all \( u \), \( x = (x) \) \( \exists (\forall) \( u \) \)

so \( x = (x) \) \( \exists (\forall) \( x \) \)

Suppose for all \( x \) \( \exists x \in 0 x \) \( \forall x \) we have \( \exists x \). Then since \( x = (x) \) \( \exists 0 x \) we claim that \( 0 x \) is the least fixed point.

Hence \( (\forall) (u) \) \( \forall \exists x \) \( (\forall) (u) \) \( \forall \exists x \) \( (\forall) (u) \)

Now \( (\forall) (u) \) \( \forall \exists x \) \( (\forall) (u) \) \( \forall \exists x \) \( (\forall) (u) \)

To see this note \( 0 x = (0 x) \) \( \forall \exists x \)
The superindex 0 indicates a classcal concept.

Thus, for any x in n, 0x ∈ (T)_{(n)}φ.

Since the sequence is ascending and d ∈ CPO, there exists

• (T)_{(T+n)}φ ⊆ (T)_{(n)}φ ' • (T)φ ⊆ (T)φ, for all x ∈ T.

Hence, the sequence is ascending because (T)φ ⊆ T.

Proof: Consider the sequence

• x ⊆ 0x, then x = (x)φ ⊆ (T)φ

0x = (0x)φ (T)

If there is a least fixed point 0x and

0x ∈ \{ x ∈ T : T > I \} then x is a CPO and d + d \simeq x is continuous.

Theorem: If \{ x \} \subseteq T and a CPO and d + d \simeq x is continuous, then

\( (\mathfrak{A})φ \subseteq (x)φ \) and \( (\mathfrak{A})φ = ((\mathfrak{A})φ')φ(\mathfrak{A})φ(\mathfrak{A})φ(\mathfrak{A})φ \)

= (\mathfrak{A})φ. So, \( \mathfrak{A} = (\mathfrak{A}x)φ(\mathfrak{A})φ(\mathfrak{A})φ(\mathfrak{A})φ \)

Proof: Given 0x ∈ T, then x is a CPO, \( \mathfrak{A} \subseteq x \) is monotone.

Proposition: If \( (T)φ \subseteq T \) is continuous, then \( (T)φ \) is continuous.

\( (\mathfrak{A})φ \subseteq (x)φ \) is the simplest \( \mathfrak{A} \subseteq x \) \subseteq \text{monotone} \subseteq T \) \subseteq T

Definition: Suppose \( (T)φ \subseteq T \) is continuous and \( \mathfrak{A} \subseteq x \) \subseteq \text{monotone} \subseteq T \) \subseteq T

If there is a least fixed point of \( (T)φ \), then

\( (\mathfrak{A})φ \subseteq (x)φ \) and \( (\mathfrak{A})φ = ((\mathfrak{A})φ')φ(\mathfrak{A})φ(\mathfrak{A})φ(\mathfrak{A})φ \)

= (\mathfrak{A})φ. So, \( \mathfrak{A} = (\mathfrak{A}x)φ(\mathfrak{A})φ(\mathfrak{A})φ(\mathfrak{A})φ \)

The classcal theory of continuous functions.

Classic fixed point results as summarized by Scott [16', 17'18].

[1] To motivate this approach we review briefly the results on account of fixed points for Cauchy continuous functions. This is an account of fixed points and recursive schemes. Key to such

In § 8 we will apply this work to give a simple development

§7. Fixed Points.
This can be regarded as a function $f: T \rightarrow T$ because whenever $X$ is replaced by a tree $t \in T$, the result is a new tree. We can easily see that this function is Cauchy continuous.

We immediately have

**Proposition 2**: Every Cauchy continuous function is monotone.

**Proof**: (just as in the classical case).

We also have a fixed point theorem.

**Theorem 2. (Fixed Point)**: If $f: T \rightarrow T$ is Cauchy continuous, then we can find $t_0$ such that $f(t_0) = t_0$, and $t_0$ is a least fixed point, and is unique if it is a constant.

**Proof**: Consider $f(X_0)$. By the unique property of variable $X_0$, namely that $X_0 \leq t$ for all trees $t$, we know that $X_0 \leq f(X_0)$. Now also $f(X_0) \leq f(f(X_0)) =: f^{(2)}(X_0)$ by monotonicity, and in general $f^{(n)}(X_0) \leq f^{(n+1)}(X_0)$. Also $\{f^{(n)}(X_0)\}$ is Cauchy since $f$ is Cauchy continuous.

Thus we can find \( \lim f^{(n)}(X_0) =: t_0 \). Now compute

$$f(t_0) = f(\lim f^{(n)}(X_0)) = \lim f^{(n)}(X_0)$$

since $f$ is continuous. But $\lim f^{(n+1)}(X_0) = \lim f^{(n)}(X_0)$, so $f(t_0) = t_0$ as we claimed.

We claim if $t_0$ is a constant, then it is the least fixed point. We have $X_0 \leq t$ for all $t \in T$. Thus $f(X_0) \leq f(t)$. So $f^{(n)}(X_0) \leq f^{(n)}(t)$ for all $n$, all $X$. If $f(t) = t$, then we have $\lim f^{(n)}(X_0) = t_0$ and $f(t) = t$, so $f^{(n)}(X_0) \leq t$ for all $n$.

To conclude $t_0 \leq t$ we use the following lemma which is stronger than we need. To conclude uniqueness we use Corollary 1 to
Theorem 2 in §6.

Lemma 1: If \( t_i \) is Cauchy and \( t_i \subseteq s \) for all \( i \), then
\[
t_i = \lim t_i \subseteq s
\]
(thus if \( t \) is a constant, then \( t = s \)).
(Constructively, if \( t_i = \lim t_i \), then \( \forall t \notin s \) and if \( t \) is a constant, \( \forall t \notin s \).

Proof: We consider several cases and their subcases.

Case I: Suppose \( t_i \subseteq s \) for all \( i \geq n \). The suppose further that the substitutions leading to \( s \) are never made at an interior variable or a permanent variable, i.e. the substitutions are made for \( X_i \) which occurs at a leaf of \( t_i \) and is eventually refined to a constant. Then it must be the case that for \( t_i = \lim t_i \), \( t = s \).

Because consider any finite path in \( s \), and consider a substitution in \( t_n \) which leads to this path; it has the form \( \underbrace{\ldots \rightarrow X} \)
\[
\rightarrow \underbrace{\ldots \rightarrow \ldots}
\]
but at some \( t_{n'} \), \( n' > n \), the variable \( X \) is refined to a constant, and the constant must be the root of the tree substituted for \( X \) because there are no interior variables into which we could substitute to obtain this portion of \( s \). (Any variable which might suffice must occur somewhere along the path and thus must be interior.)

Case II: Again suppose \( t_i \subseteq s \) for all \( i \geq n \). But now suppose that substitution is made for permanent variables \( X_j \). These \( X_j \) are either interior variables or permanent leaves. In any case we distinguish two subcases.

(a) the permanent variable \( X_j \) occurs only finitely many times in the limit \( t \).

In this case, let \( t_{p_j} \) be the finite tree containing all
constant function. 

variable \( x \) as input. It has no variable then it is a
so we let \( y \) denote the function arising by taking the
times Cauchy conditions. We state this as a theorem. To do
selecting one tree variable as input. Such functions are some-
Every tree \( t \) defines a function \( t \). ↑ \( t \) by

g.z.d.

Can make the same substitutions as for the approximations of
Thus we see by this argument that in the limit it, we
occurrence would be damaged.

occurrences, hence in the limit, otherwise the first for \( x \).
In order to use a theorem, therefore whatever substitution is made 
substitution has been made, then the substitution of the variable 
substituted for \( x \). Again we can argue that once the \( x \)
example, \( X \). In general we may have to look within the a
Now consider the next earliest variable \( x \) in the
always be used).

\( P \) are later substituted, so the composite \( g \) could 
occurrence of \( x \) one might substitute \( x \), but then the
could be decomposed, say \( g = f(x)^p \cdots T(x) \) and at some
since is the constant (actually the substitution \( g \)
Clearly the same substitution must be made for all \( x \).
Thus in the example, let the dotted part:
consider the substitution \( \alpha \) made to obtain part of:

\[
\begin{array}{c}
  x^0 \xrightarrow{0_\alpha} t^\alpha \xrightarrow{2_\alpha} z^\alpha \\
  x \xrightarrow{0_\alpha} t^\alpha \\
  x^0 \xrightarrow{0_\alpha}
\end{array}
\]

Now pick \( x^0 \).  

\( \forall x \) used in obtaining a occurs earlier. In the example be-
whose distance to the root is least and for which no variable

\[
\text{Now select the variable } x^0 \text{ which occurs earliest, i.e.}
\]

the same on every path.

\( \forall x \) in such a way that the segment from \( x \) to \( x \)
In short this means that no variable \( x \) can occur earlier.

\( \forall x \) can be made (otherwise we would examine \( x \), not the X
for which a substitution to obtain a without using
which occurs earlier on each path from the root to \( x \) and
which occurs earliest without loss of generality, that there is no variable
made occur instantaneously in the limit. We assume
the permanent variables \( x \) for which a substitution is
\( \forall x \) can be obtained from \( x \) by the same substitution for \( x \).

Any path in \( s \) which is obtained by substitution into
\[
([b]) t = s' \xrightarrow{[f]} x \]

substitution \( a \) for \( x \) where \( a \) is such that from
occurrences of \( x \). Then in the limit tree it make the

-294-
Theorem 3: For all trees \( t \in T \), \( \lambda X_1[t] \) is Cauchy continuous iff \( t \) has at most one variable \( X_1 \) and a constant root.

Proof: (1) Let \( s_0 \subseteq s_1 \subseteq s_2 \subseteq \ldots \) be a Cauchy sequence. We must show that \( t(s_0) \), \( t(s_1) \), \( t(s_2) \), \( \ldots \) is a Cauchy sequence if \( X_1 \) is the only tree variable. If \( X_1 \) is the only variable, then clearly \( t(s_i) \subseteq t(s_{i+1}) \) because we can use the refinement giving \( s_i \subseteq s_{i+1} \) to show that \( t(s_{i+1}) \) refines \( t(s_i) \). However, if \( t \) contains another variable, say \( X_2 \), then we can input to \( t \) a tree \( s_1 \) involving variable \( X_2 \) which is refined to give \( s_2 \). So while \( s_1 \subseteq s_2 \), not \( t(x_1) \subseteq t(s_2) \) because not all occurrences of \( X_2 \) are replaced in \( t(s_1) \). For example let \( t \) be

and let \( s_1 \) be \( f \rightarrow X_2 \), let \( s_2 \) be \( f \rightarrow g \). Then \( s_1 \subseteq s_2 \) but \( t(s_1) \not\subseteq t(s_2) \) as the figure shows.

(2) Next we must show that \( t(\lim s_i) = \lim t(s_i) \). This can be done by induction on the depth of the trees (since both \( t(\lim s_i) \) and \( \lim t(s_i) \) are trees) starting at the root and considering the cases that the depth n node is either an \( X_1 \).
We leave the tedious details to the reader.

(3) Finally we must show that \( \{ t^{(n)}(s) \} \) is Cauchy for any \( s \in T \) as long as \( t \) has a constant root. Clearly having a constant root is necessary as the following example shows.

Take \( t := \rightarrow X \rightarrow f \rightarrow X \rightarrow \). The sequence \( t(t), t(t(t)), \ldots \) is not Cauchy because the first occurrence of \( X \) never "settles down."

On the other hand, given that \( t \) has a constant root, say \( f \) or \( P \), then for any occurrence of \( X_0 \) in \( t^{(n)}(X_0) \) we know that at \( t^{(n+1)}(X_0) \) that occurrence is refined to the root value.

Q.E.D.

We can talk about a function \( f: T \rightarrow T \) being Cauchy continuous on a subset \( D \) of \( T \). This is the concept we need if we want to regard each \( t \in T \) as a Cauchy continuous function.

**Definition 3:** \( f: T \rightarrow T \) is Cauchy continuous on \( D \subseteq T \) iff

(i) if \( \{ s_i \} \) is Cauchy and \( s_i \in D \) for all \( i \), then \( \{ f(s_i) \} \) is Cauchy and \( f(\lim s_i) = \lim f(s_i) \)

and

(ii) \( \{ f^{(n)}(s) \} \) is Cauchy for all \( s \in D \).

**Example 2:** The function \( \lambda X_1[t] \), given by

\[
\begin{array}{c}
\text{P} \\
\downarrow \\
X_1 \\
\downarrow \\
X_2
\end{array}
\]

is not Cauchy on \( T \) but is Cauchy on the set of all trees not containing the variable \( X_2 \).

**Theorem 4:** Any tree \( t \) with constant root is Cauchy continuous on the set \( D_t \) of all trees which do not contain tree variables.

**Proof:** for reader
The $P^f$ are called given functions or base functions.

The $P$ are called defined functions, $P^f$ is the primitive function.

That the identity function is being used, the letters $P^f$ right we also write $I$ or $x \mapsto I$, in $P^f$, to indicate

When only the term $x$ is present on the

From which compute the same functional (see $\geq$) for a proof.

be transversed into a set of (right linear) recursion equations.

The semantics of flowcharts arise as a special case of the

is indeed the tree obtained by unwindning the flowchart.

least fixed point. The reader can check that this fixed point

is constant, so it is a least upper bound. Thus this has a

one variable, so it is Cauchy continuous. Moreover, if $\lim_{n \to \infty} x(n)$

This function is $I$ if $x = I$. It has a constant root and only

but we can also consider a function naturally arising from the
This can be "unwound" to form the tree.

**Example:** Consider the flowchart.

Very primitive example.

In detail and compare to other methods. We begin with a point in section 6.1. We now examine this fixed point semantics. It is based on the fixed.

There are numerous ways to show that the flowchart denote.

We consider flowcharts and systems of recursion equations as

For a thorough discussion of this fascinating class of schemes the reader should see [7]. We assume familiarity with the standard operational semantics of these schemata.

Example 2: \[ F_1 x := P(x) \rightarrow hx, F_2 x \]

\[ F_2 x := Q(x) \rightarrow fP_1 fx, gP_1 gx \]

We can write these as a single equation if we nest the conditionals.

\[ Fx := Px \rightarrow hx, (Qx \rightarrow ffx, gfgx). \]

The meaning of this scheme can be found by systematically enumerating paths of execution in the following spirit: "first we must test \( P \), if true then exit with \( x \), otherwise test \( Q \), if true, then compute \( f \) and test \( P \), if true then apply \( f \) to result and exit, otherwise ...:" The result of this rambling is the tree depicted below.
A more elegant way to assign a tree to this recursion scheme is to consider it as a function. Consider the tree

This defines a function $F: T \to T$. The function is Cauchy and has a constant fixed point which is its least fixed point. That fixed point is in fact the correct meaning for the recursion scheme in the sense that the tree and the scheme are strongly equivalent -- they compute the same functional. Below is a diagram of the term $f^{(2)}(X_0)$ in the Cauchy sequence $\{f^{(n)}(X_0)\}$.

Example 3: diagram. (see next page)

The example above suggests the following definition and theorem.

**Definition 2:** Given a single equation monadic recursion scheme $f$ (with nesting allowed), the derived tree function is the function $f: T \to T$ obtained by replacing the variable $F$ by a tree variable $X_0$.

**Theorem 1 (semantics):** Any single equation recursion scheme is strongly equivalent to the least fixed point of the derived tree function.
Definition: If $I^n = I^n$ then define:

\[ x = \left( u, \cdots, u, T_x \right) \left( u \right)^I \Rightarrow x = \left( u, \cdots, u, T_x \right) \left( I + u \right)^I \]

The natural way: Let $x = \left( u, \cdots, u \right)^I$ be the interpretation in $\mathcal{L}_n$.

By definition, we have a vector function $f: \mathbb{Z}^n \to \mathbb{Z}^n$ which assigns to each input vector $z = \left( z_1, \cdots, z_n \right)$ the output vector $f(z) = \left( f(z_1), \cdots, f(z_n) \right)$.

Let $\mathcal{T}_x$ be the tree in $\mathcal{L}_n$ to $\mathcal{L}_n$. Let $\mathcal{T}_x$ be the function that assigns to each input vector $z = \left( z_1, \cdots, z_n \right)$ the output vector $x = \left( x_1, \cdots, x_n \right)$.

Mathematically, we consider functions from $\mathbb{Z}^n$ to $\mathbb{Z}^n$. In order to "summate" these functions, we must form two trees that are so interconnected that they cannot be replaced by a single tree. We think of $\mathcal{T}_x$ as variables, $x$ as variables.

The corresponding tree functions are:

\[
\begin{align*}
(x^0)^2, x^1, x^2, T_x \quad &\rightarrow \quad x^1, x^2, T_x \\
(x^0)^2, x^1, x^2, T_x \quad &\rightarrow \quad x^0, x^1, T_x
\end{align*}
\]

Example: Consider the following system of two equations:

\[
\begin{align*}
(x^0, x^1, T_x \quad &\rightarrow \quad x^2, x^3, T_x, T_x \\
(x^0, x^1, T_x \quad &\rightarrow \quad x^0, x^1, T_x
\end{align*}
\]
theory for flowcharts as a special case.

of recursion equations. Thus will then give us a smooth semantic

We now want to generalize this approach to any finite system

Proof: see Theorem 5.
In the above example, \( t^{(2)}(F_1, F_3) \) is \( <t_1(t_1(x_1, x_2), t_2(x_1, x_2)), t_2(t_1(x_1, x_2), t_2(x_1, x_2))> \). We can now examine the sequence \( \{t^{(n)}(x_0, x_0)\} \) in \( T^2 \) and ask if it has a limit which can be regarded as a fixed point of the vector function \( t: T^2 \rightarrow T^2 \). Such a fixed point consists of two infinite trees. The tree in the first component, the principal tree, should be the meaning of the recursion equations.

The fixed point approach to the semantics of recursion equations will work if we can extend the results of \( \S 6 \) on convergence and \( \S 7 \) on fixed points to \( n \)-tuples of trees, \( T^n \). It is easy to check the following facts.

**Definition 4:** For \( t, s \in T^n \) define \( t \subseteq s \) iff \( t_i \subseteq s_i \) for all \( i \) where \( t = <t_1, \ldots, t_n> \), \( s = <s_1, \ldots, s_n> \).

**Theorem 2:** \( <T^n, \subseteq> \) is a QO set.

**Definition 5:** A sequence \( t_0 \subseteq t_1 \subseteq \ldots \) in \( T^n \) is Cauchy iff the sequence in each component is Cauchy.

**Definition 6:** A function \( t: T^n \rightarrow T^n \) is Cauchy continuous on \( D \) if it is Cauchy continuous on \( D \) for each component.

**Theorem 3** (convergence): Every Cauchy sequence \( \{t_i\} \) in \( T^n \) converges to \( \lim t_i \) which is the least upper bound if each component of the limit is constant.

**Proof:** just as in Theorem 2, \( \S 6 \).

**Theorem 4** (fixed point): If \( t: T^n \rightarrow T^n \) is Cauchy continuous, then we can find a \( t_0 \in T^n \) such that \( t(t_0) = t_0 \), and if \( t_0 \)
is constant in all components then it is the least fixed point.

**Definition 7:** Given a set of recursion equations $E$

$$F_i x := P_i x + \alpha_i x, \beta_i x \quad i=1,\ldots,n$$

define the derived tree function as the vector function $E: T^n \rightarrow T^n$ obtained by taking a derived function for equation $F_i$ in component $i$ (obtained by considering each $F_i$ as a tree variable).

**Example 5:** In example 4 we defined the derived function

$$t_1(F_1,F_2) := P_1 + f_0, (P_2 + F_1f_1,F_2f_2)$$

$$t_2(F_1,F_2) := P_3 + f_3, (P_4 + F_1f_4,F_2f_5)$$

$$t(F_1,F_2) := \langle t_1(F_1,F_2), t_2(F_1,F_2) \rangle .$$

We can now prove

**Theorem 5 (semantics):** Any recursion scheme $E$ is strongly equivalent to the least fixed point of its derived tree function.

**Proof:** First we note that the derived tree function is Cauchy continuous because it has a constant root in each component. So the limit of $\{ t_E^{(n)}(x_1,\ldots,x_m) \}$ exists.

Consider any computation of $E$, either terminating or non-terminating. This computation is a sequence of function calls and predicate tests such as

$$f_{i_1}, P_{i_2}, f_{i_3}, P_{i_4}, f_{i_5}, f_{i_6}, \ldots$$

We can show that such a sequence is a path in $\lim t_E^{(n)} = t$. 

trees is universal (for schemata constructed as described) the class of functions recursively presented

presentation features of the algorithms from the data

the scheme level appears to be appropriate for

quite different!

describe an algorithm, so they will be part of any adequate
the trees are the main structures needed to

philosophical virtues among them are

schemes. It appears that this class has many recursive and
approach is merely the class of schemes we choose, the tree
others also lead in this direction. The new feature of this
theory such as gentle [6], Lackham, Park and Parson [11] and
an appropriately abstract notion of algorithm. Work in schema
and SCOTT and STENROTH in [18] discuss the problem of finding

constructional results of this paper to each of the topics.
we will provide a brief historical perspective on the new

property of programs.

a determination of a term, or as a structural

for recursive procedures

a constructive rendering of a fixed point semantics

possible mathematical interpretation as a function.
distinguishment from its representation as a program and from

a determination of an algorithm as an abstract object

In this report we have considered three main ideas.

§9. Summary and conclusion.

- 42 -
We leave the routine details to the reader.

To prove this proceed by induction on the length of the sequence.
The class of tree schemes is simpler than Scott's schemes in "The Lattice of Flow Diagrams" [16], yet is richer than the class of flowchart schemes or recursion schemes. Indeed the class of recursively presented tree schemes is universal in the sense of Strong [20], Paterson [15], Constable & Gries [3]. It is, in fact, not difficult to see that the recursively presented tree schemes are equivalent to Strong's deterministic effective functionals and that a natural generalization to non-deterministic trees (allowing non-deterministic branch nodes) gives a class of schemes equivalent to Strong's non-deterministic effective functionals. We will discuss these matters elsewhere.

The constructive fixed point semantics considered in this paper is based directly on Scott's work in "The Lattice of Flow Diagrams" [16], which is in turn based on Scott's 1970 Princeton paper [17]. The origins of this fixed point theory can be found in Kleene's recursion theorem for functionals [9] and other work of logicians. In particular the article by Krieger & Ritchie [10] exploits the fixed point interpretation of recursion equations. However, Scott's 1970 paper was the first work to explicitly use fixed point semantics to solve problems in computer science dealing with the meaning of recursive procedures. His 1970 paper showed the way to a unified treatment of various methods for proving the correctness of recursive procedures.

The version of a fixed point semantics given here differs from Scott's in two minor ways. First, we consider the meaning of a recursive procedure to be an algorithm, not a function in a function space. Second, we present the entire mechanism
constructively which helps distinguish the computational aspects of these semantics.

Finally, the notion of refinement used here is new. We are attempting to capture some aspects of the concept of program structure and program development. We shall explore this notion in several directions in forthcoming work. We examine further its role in describing substitutions, its role in unifying various arguments in schematology and its role in describing "structured ways to program."
I thank Pauline Cameron for her unselfishly excellent arrangement of these ideas on the typed page.

Professor John Williams
Martin Solomon
Janes Stimson
Norton Stegel
John Porteeta
Susan O'Connell
Michael O'Donnell
Steve Muchnick
Kurt Methimann
Zvi Cat
Dr. Herbert Eggert
Edward Clarke

I would like to thank the students and visitors in computer science 78, Advanced Computing Theory, for consult for helping.

Acknowledgments


