The Fixpoint Theorem in the Σλ-Calculus

S W Postma and A McGee
Department of Computer Science, University of Natal, P O Box 375, Pietermaritzburg 3200

Abstract

The Σλ-calculus is presented as an extension to the standard λ-calculus, and a Σλ-term called the strong conditional is discussed. The strong conditional is shown to be non-monotomic, and an extension is made to the Σλ-calculus to allow the fixpoints of recursive Σλ-abstractions to be established.
Keywords: Octolisp, fixpoints, equisimultaneity, nondeterminism.
Computer Review Categories: F.3.1, F.4.1

Received January 1990, Accepted May 1990.

1. Introduction

The $\Sigma\lambda$-calculus is an extension of the standard (untyped) λ-calculus as presented in Landin [4], Stoy [12] et al., and it provides a consistent mathematical basis for the functional / applicative / relational language Octolisp, developed by Postma [8]. A full description of the $\Sigma\lambda$-calculus is given by Postma [7], but we shall concentrate here on one particular $\Sigma\lambda$-term.

The essential difference between the calculi lies in the inclusion of a conditional as a $\Sigma\lambda$-term. The construct is called the strong conditional by Postma and Phillips [9], and has a corresponding evaluation strategy which is "angelic" (after Hoare in Manna [5]), or "fair" (Plotkin, [10]), in that not all guards need be defined. Such a strategy has been named equisimultaneous evaluation by Postma [8]. The strong conditional is a natural generalization of both the COND of Lisp 1.5 and Dijkstra's **if-fi** construct [1].

Since function application is naturally allowed in the $\Sigma\lambda$-calculus, the nondeterminism inherent in the strong conditional poses a semantic problem. By allowing general recursive definitions, a method is required to specify their mathematical meaning - the traditional approach (Tennent, [13]) is to take the (unique) least fixpoint, or least defined function in a chain of functions which satisfy the defining equation, as the meaning. Kleene's first recursion theorem (Manna, [5]) establishes the existence of this fixpoint, provided that the functional specification is defined by composition of monotonic functions. The strong conditional, however, is not monotonic, as we shall see.

The purpose of this paper is to investigate the behaviour of the strong conditional (and the derivable weak and left conditionals), and to develop Postma's suggested solution [7] to the way in which Kleene's fixpoint theorem may become applicable in the $\Sigma\lambda$-calculus.

2. Fundamentals of the Σλ-Calculus

The abstractions of the $\Sigma\lambda$-calculus are more complex than those of the λ-calculus, and correspond to the closures of Landin [4]. We are, however, more concerned here with the syntax and semantics of the strong conditional, and we shall confine ourselves to defining a simplified version of the abstractions, omitting details of their substitution and reduction rules. A full description of $\Sigma\lambda$-abstractions with their corresponding substitution and reduction rules is given by Postma [7].

Atom

The irreducible elements of the language are called atoms. The set of atoms, A, is given by

$$A = I \cup C_L \cup C_N \cup C_C,$$

where I is an infinite set of identifiers/variables,
C_L is the set of basic constants {t, f, ⊥, ~},
C_N is the set of integers,

C_C is an empty, finite, or infinite set of constants,
and these four sets are pairwise disjoint.

Note that the symbol $\perp$ denotes the value "undefined" or "bottom" (after Stoy, [12]), or "don't care" or "don't know" or "can't know", or infinite computations (for example, an expression with no normal form).

Terms are denoted by upper-case letters, while lower-case letters (which may be subscripted) are used to indicate identifiers or variables.

$\Sigma\lambda$-terms

The $\Sigma\lambda$-terms are defined inductively as follows:

1. Any atom is a $\Sigma\lambda$-term.

2. If $X, Y_0,\ldots,Y_n$ are $\Sigma\lambda$-terms, then so are
   $\phi\ X\ \phi$ and
   $\phi\ X\ Y_0\ \ldots\ Y_n\ \phi$
   (Such terms are called applications, of a function-form to zero or more argument-forms).

3. If $G_0,\ldots,G_n,\ V_0,\ldots,V_n$ are $\Sigma\lambda$-terms, then so are
   $\mathbb{K}\ \{\ \}$ and
   $\mathbb{K}\ \{\ G_0 \Rightarrow V_0\ G_1 \Rightarrow V_1\ \ldots\ G_n \Rightarrow V_n\ \}$
   (Such terms are called (strong) conditionals).

4. If $x_0,\ x_1,\ldots,x_i$ are individual variables, and X is a $\Sigma\lambda$-term, then
   $\phi\{\ \}\ X\phi$ and
   $\phi\{\ x_0,\ x_1,\ldots,x_i\ \}\ X\phi$ are also $\Sigma\lambda$-terms.
   (Such terms are called abstractions, and $\{x_0,\ldots,x_i\}$ are called the α-variables, corresponding to the bound variables of the λ-calculus).

Substitution and Conversion Rules

We now describe the substitution and reduction/conversion rules, after the manner of Hindley and Seldin [2], and Stoy [12], using $\equiv$ for syntactic identity.

We note first that if
$Y \equiv \mathbb{K}\ \{\ G_0 \Rightarrow V_0\ \ldots\ G_n \Rightarrow V_n\ \}$
then X occurs in Y
iff X occurs in G_i or V_j for at least one i, $0 \leq i \leq n$, and/or one j, $0 \leq j \leq n$.

Substitution Rules

The result of substituting N for free occurrences of x in M is denoted by [N/x] M, and is defined inductively as follows:

1. If M has no free occurrences of x, then [N/x] M $\equiv$ M.

2. If M is an atom, then:
   if M $\equiv$ x, then [N/x] M $\equiv$ N
   else [N/x] M = M.

3. If $M \equiv \phi\ M_0\ M_1\ \ldots\ M_n\ \phi$, then
   $[N/x]\ M \equiv \phi\ [N/x]\ M_0\ [N/x]\ M_1\ \ldots\ [N/x]\ M_n\phi$.

4. If $M \equiv \mathbb{K}\ \{\ G_0 \Rightarrow V_0\ \ldots\ G_n \Rightarrow V_n\ \}$, then
   $[N/x]\ M \equiv \mathbb{K}\ \{\ [N/x]\ G_0 \Rightarrow [N/x]\ V_0\ \ldots\ [N/x]\ G_n \Rightarrow [N/x]\ V_n\ \}$.

5. For abstraction rules, see Postma [7].

Reduction Rules

The reduction rules define the calculus proper.

The basic rules relating to the logical symbols are:

$\lambda 1.\ \phi\ \sim\ t\ \phi \vdash_\lambda f$
$\lambda 2.\ \phi\ \sim\ f\ \phi \vdash_\lambda t$
$\lambda 3.\ \phi\ \sim\ \perp\ \phi \vdash_\lambda \perp$

The reduction rules for the strong conditional are given inductively by:

$\kappa 1.\ \mathbb{K}\ \{\ \} \vdash_\kappa \perp$

$\kappa 2.\ \mathbb{K}\{\ G_0 \Rightarrow V_0\ \ldots\ f \Rightarrow V_i \ldots\ G_n \Rightarrow V_n\ \}$
$\vdash_\kappa \mathbb{K}\ \{\ G_0 \Rightarrow V_0\ \ldots\ G_{i-1} \Rightarrow V_{i-1}$
$G_{i+1} \Rightarrow V_{i+1}\ \ldots\ G_n \Rightarrow V_n\ \}$

$\kappa 3.\ \mathbb{K}\ \{\ G_0 \Rightarrow V_0\ \ldots\ t \Rightarrow V_i\ \ldots\ G_n \Rightarrow V_n\ \}$
$\vdash_\kappa V_i$

$\kappa 4.\ \mathbb{K}\ \{\ \perp \Rightarrow V_0\ \ldots\ \perp \Rightarrow V_n\ \} \vdash_\kappa \perp$

We note that an equisimultaneous reduction of the guards is necessary, i.e. no guard may be excluded from the reduction process, but in an arbitrary but finite number of guard reductions, each guard must be reduced at least once.

3. The Weak and Left Conditionals

The general form of the strong conditional, as described above, is given by

$$K_S = \mathbb{K} \{ G_0 \Rightarrow V_0 \ \ldots \ G_n \Rightarrow V_n \}$$

where (G_i, V_i) are guard-value pairs, and the brackets $\{,\}$ are used to indicate equisimultaneous evaluation. The strong conditional is so-named because the weak and left conditionals are derivable from it, but not conversely.

For the left conditional corresponding to the COND of Lisp 1.5, we define

$$K_L = \mathbb{K} \{\!\!\{ G_0 \Rightarrow V_0 \ \ldots \ G_n \Rightarrow V_n \}\!\!\}$$

where the brackets $\{\!\!\{, \}\!\!\}$ indicate a list, implying the left sequential evaluation of Lisp. The guards are evaluated from the left, and each one is evaluated until either non-enabling (i.e. false), or enabling. If G_j is non-enabling, then G_{j+1} is evaluated, otherwise all guard evaluation ceases, and V_j is taken as the value of the conditional.

We may define K_L recursively in terms of K_S by

$$\mathbb{K} \{ G_0 \Rightarrow V_0 \ \ \sim G_0 \Rightarrow \mathbb{K} \{\!\!\{ G_1 \Rightarrow V_1 \ \ldots \ G_n \Rightarrow V_n \}\!\!\} \}$$

For the weak conditional corresponding to Dijkstra's **if-fi** construct, we define a form that ensures that all guards are defined (note that anything defined and not false is taken as true), and an arbitrary enabled value is chosen.

We write

$$K_w = \mathbb{K} \lceil G_0 \Rightarrow V_0 \ \ldots \ G_n \Rightarrow V_n \rfloor$$

and we may define K_w in terms of K_s by

$$\mathbb{K} \{ G_0 \ {}_w\!/ \ G_1 \ {}_w\!/ \ \ldots \ {}_w\!/ \ G_n \Rightarrow \mathbb{K} \{ G_0 \Rightarrow V_0 \ \ldots \ G_n \Rightarrow V \} \}$$

where ${}_w\!/$ is the weak three-valued disjunction (Kleene, [3]), with the truth-table:

${}_w\!/$	t	f	$\perp$
t	t	t	$\perp$
f	t	f	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$

4. Fixpoints and Nondeterminism

By including the strong conditional (and its derivative weak conditional) in the $\Sigma\lambda$-calculus, we clearly have a nondeterministic language, and as Schmidt [11] notes, we shall require new methods by which to specify its complete denotational semantics.

Furthermore, from the definition of the $\Sigma\lambda$-terms, we may create recursive functional specifications which may be applied to the strong (or weak) conditional. We shall need to look briefly at the fundamentals of fixpoint theory, as presented by Manna [3], to discuss the properties of recursive definitions in general, before dealing with the problem of nondeterminism.

The Partial Ordering $\sqsubseteq$

We first define the partial ordering $\sqsubseteq$, corresponding to the idea "is less defined than or equal to", on every extended (i.e. lifted) domain $D^+ = D \cup \{ \perp \}$ by

$$\perp \sqsubseteq x \quad \text{and} \quad x \sqsubseteq x \quad \forall \ x \in D^+$$

For $(D^+)^n$

$x \sqsubseteq y$ iff $x = \langle x_0,\ldots,x_n \rangle$, $y = \langle y_0,\ldots,y_n \rangle$ and $x_i \sqsubseteq y_i$ for each i, $1 \leq i \leq n$.

Monotonic Functions

A function f from $(D_1^+)^n$ into D_2^+ is said to be **monotonic** if $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$ $\forall$ **x**, **y** $\in (D_1^+)^n$, and we shall denote by D, where

$$D = [(D_1^+)^n \rightarrow D_2^+]$$

the class of such monotonic mappings.

If $f,g \in D$, we write $f \sqsubseteq g$ if $f(x) \sqsubseteq g(x)$ $\forall$ **x** $\in (D_1^+)^n$. If f_0, f_1, f_2,... is a sequence of functions in D, we denote the sequence by $\{f_i\}$, and call it a chain if $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \ \ldots$. We call f an upper bound of $\{f_i\}$ if $f_i \sqsubseteq f$ $\forall$ i $\geq$ 0, and if $f \sqsubseteq g$ for every upper bound g of $\{f_i\}$, then f is the unique least upper bound of $\{f_i\}$, denoted by lub $\{f_i\}$. For a proof that every chain $\{f_i\}$ has a lub see Manna [5].

Continuous Functionals

A **functional** τ over D maps D into itself,

i.e. τ takes any monotonic function $f \in D$ as its argument and yields a monotonic function $\tau[f] \in D$ as its value.

A functional τ over D is said to be **monotonic** if $f \sqsubseteq g$ implies $\tau[f] \sqsubseteq \tau[g]$ $\forall$ f, $g \in D$; a monotonic functional τ over D is **continuous** if for any chain $\{f_i\}$, we have $\tau[\text{lub}\{f_i\}] = \text{lub}\ \{\tau[f_i]\}$

From Manna [5], we have the following important property of continuous functionals:
Any functional τ defined by composition of monotonic functions and the function variable F (representing known monotonic functions), is continuous.

Fixpoints of Functionals

A function $f \in D$ is a **fixpoint** of a functional τ if $\tau[f] = f$, i.e. τ maps f into itself. If f is a fixpoint of τ and $f \sqsubseteq g$ for any fixpoint g of τ, then f is the (unique) **least fixpoint** of τ.

Consider now the sequence of functions defined by

$\tau^0\ [\Omega] = \Omega$, where $\Omega : (D_1{}^+)^n \to \perp$
and $\tau^{i+1}\ [\Omega] = \tau\ [\tau^i\ [\Omega]]$

Clearly, since the totally undefined function Ω is monotonic, τ is continuous (and therefore monotonic), so that we have

$\Omega \sqsubseteq \tau[\Omega] \sqsubseteq \tau^2[\Omega] \sqsubseteq \ldots\ldots$

and the sequence $\{\tau^i[\Omega]\}$ must therefore be a chain, and hence has a lub.

This leads us to Kleene's first recursion theorem (Manna [5]):
Every continuous functional τ has a least fixpoint, $f\tau$. Actually, $f\tau$ is lub $\{\tau^i[\Omega]\}$.

If we have a recursive definition of the form:

$$F(x) \Leftarrow \tau\ [F]\ (x),$$

then we shall take the least fixpoint $f\tau$ of τ as the meaning of the definition.

For example, consider the functional τ over $[\mathbb{N}^+ \to \mathbb{N}^+]$
where
$\tau[F]$ (n): if x = 0 then 1 else x.F(x - 1)
(Manna, [5]).

Now $\tau^0\ [\Omega]$ (x) : $\perp$

$\tau^1\ [\Omega]$ (x) : if x = 0 then 1 else $\perp$
$\tau^2\ [\Omega]$ (x) $\equiv \tau\ [\tau\ [\Omega]\ ]$ (x):
if x = 0 then 1 else x. (if x-1 = 0 then 1 else $\perp$)
$\equiv$ if x = 0 then 1 else if x = 1 then x else $\perp$
$\tau^3\ [\Omega]$ (x) $\equiv \tau\ [\tau^2\ [\Omega]\ ]$ (x):
if x = 0 then 1 else x. (if x -1 = 0 then 1 else if
x - 1 = 1 then x - 1 else $\perp$)
$\equiv$ if x = 0 then 1 else if x = 1 then x else if x = 2 then x . (x-1) else $\perp$.
:
:
Clearly, $\tau^i\ [\Omega]$ (x): if x < i then x! else $\perp$, i $\geq$ 0 and
$f\tau$ = lub $\{\tau^i[\Omega]\}$ = x! as the following table illustrates:

Chains for fixed x values →

x	0	1	2	3	4	5	6	....
$\tau^0\ [\Omega](x)$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	...
$\tau^1\ [\Omega]$	0!	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	...
$\tau^2\ [\Omega]$	0!	1!	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	...
$\tau^3\ [\Omega]$	0!	1!	2!	$\perp$	$\perp$	$\perp$	$\perp$	...
$\tau^4\ [\Omega]$	0!	1!	2!	3!	$\perp$	$\perp$	$\perp$	...
$\tau^5\ [\Omega]$	0!	1!	2!	3!	4!	$\perp$	$\perp$	...
.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	

Now since $f\tau = \tau\ [f\tau]$ = x!, we may claim that the recursive definition

$F(x) \Leftarrow$ if x = 0 then 1 else x . F (x - 1)

denotes the factorial function. □

A critical question now arises: does Kleene's theorem hold in the $\Sigma\lambda$-calculus? The ability to find the least fixpoint of a recursive definition is essential to a complete denotational semantic description of the $\Sigma\lambda$-calculus (and hence Octolisp). The strong conditional is, however, not monotonic, as a simple example illustrates:

since $[\![\{ \ldots t \Rightarrow 0 \ldots t \Rightarrow 1 \ldots \}]\!]$ may be 0 or 1

we see that

$f_k = \phi\{ g_0\ v_0 \ldots g_n\ v_n \}\ [\![\{ g_0 \Rightarrow v_0 \ldots \}]\!]\ \phi$

is not monotonic. (The weak conditional is

naturally also non-monotonic).
We shall now examine the problem and its possible solution in some detail.

5. The Operator S_n

The denotation of a nondeterministic construct is no longer a single value d from a domain D, but a set of values $\{d_0,d_1,\ldots\}$ (from the powerdomain $\mathbb{P}(D)$) containing all possible results from any evaluation (Schmidt,[11]).

For the strong conditional, K_S, we now define a mathematical function S_n which will produce exactly this set. (This is a redefinition of the S^n_k function given by Postma [7]).

We define

$S_n : (L^n)^+ \times (E^n)^+ \rightarrow \{\perp\} \cup \{X \mid X \in 2^{ER} \;\&\; \perp \in X\}$
$: \langle G_1\ G_2\ \ldots\ G_n\ ;\ V_1\ V_2\ \ldots\ V_n \rangle$
$\rightarrow$ **if** $G_1\ {}_S\!/\ G_2\ \ldots\ {}_S\!/\ G_n$ **then** $\{V_i \mid G_i = t,\ 1 \le i \le n\} \cup \{\perp\}$
else $\perp$

where $L^+ = \{t,f,\perp\}$, $E^+ = \{V \mid V$ is in normal form$\} \cup \{\perp\}$, $ER = E^+$,
and **if** e **then** a **else** b is the monotonic function defined by:

$\langle t,a,b\rangle \rightarrow$ a where b may be $\perp$,
$\langle f,a,b\rangle \rightarrow$ b where a may be $\perp$, and
$\langle \perp,a,b\rangle \rightarrow \perp$

Theorem: S_n is monotonic if we extend the definition of $\sqsubseteq$ to sets as follows: If A and B are sets, then $A \sqsubseteq B$
iff
$\forall\ a \in A\ \exists\ b \in B$ s.t. $a \sqsubseteq b$
and $\forall\ b \in B\ \exists\ a \in A$ s.t. $a \sqsubseteq b$
(i.e. the Egli-Milner powerdomain definition [11]).

Lemma: If S^+ is a collection of sets such that $\perp$ occurs in each element of S^+, then for any $A,B \in S^+$, we have $A \sqsubseteq B$ iff $A \subseteq B$.

Proof: Consider

$$\begin{array}{lll} & \mathbf{G} = & \langle G_1,\ldots,G_i,\ldots,G_j,\ldots,G_n\ ; \\ & & \quad V_1,\ldots,V_i,\ldots,V_j,\ldots,V_n\rangle \\ \sqsubseteq & \mathbf{H} = & \langle H_1,\ldots,H_i,\ldots,H_j,\ldots,H_n\ ; \\ & & \quad W_1,\ldots,W_i,\ldots,W_j,\ldots,W_n\rangle \end{array}$$

i.e. for any i, either $G_i = \perp$ and $H_i = \perp$
or $H_i \neq \perp$
or $G_i \neq \perp$ and $G_i = H_i$,
and similarly for V_i.

Cases

0. No enabling G_i, one enabling H_i but enabled value is $\perp$:
$\phi S_n\ \mathbf{G}\phi = \perp \sqsubseteq \phi S_n\ \mathbf{H}\phi = \{\perp\}$

1. If $\mathbf{G} = \mathbf{H}$ then $\phi S_n\ \mathbf{G}\phi = \phi S_n\ \mathbf{H}\phi$
2. If $\mathbf{G} \sqsubset \mathbf{H}$ then
   2.1 if $G_i = \perp$ and $H_i \neq \perp$ then
      2.1.1 either $H_i = t$
      and then $\phi S_n\ \mathbf{G}\phi = \{Vs_1\ Vs_2\ \ldots\ Vs_i\ \perp\}$,say,
      and $\phi S_n\ \mathbf{H}\phi = \{Vs_1\ Vs_2\ \ldots\ Vs_i\ \perp\} \cup \{W_i\}$
      so that $\phi S_n\ \mathbf{G}\phi \sqsubseteq \phi S_n\ \mathbf{H}\phi$
      2.1.2 or $H_i = f$
      and then $\phi S_n\ \mathbf{G}\phi = \phi S_n\ \mathbf{H}\phi$ i.e. $\sqsubseteq$ holds.

   2.2 if $V_j = \perp$ and $W_j \neq \perp$ then

   2.2.1 either $G_j = H_j$ and
      2.2.1.1 $G_j = t$ i.e. $\phi S_n\ \mathbf{G}\phi \sqsubseteq \phi S_n\ \mathbf{H}\phi$ as above
      2.2.1.2 $G_j = f$ then $\phi S_n\ \mathbf{G}\phi = \phi S_n\ \mathbf{H}\phi$
      2.2.1.3 $G_j = \perp$ then $\phi S_n\ \mathbf{G}\phi = \phi S_n\ \mathbf{H}\phi$

   2.2.2 $G_j = \perp$ and $H_j \neq \perp$
      2.2.2.1 $H_j = t$ i.e. $\phi S_n\ \mathbf{G}\phi \sqsubseteq \phi S_n\ \mathbf{H}\phi$
      2.2.2.2 $H_j = f$ i.e. $\phi S_n\ \mathbf{G}\phi = \phi S_n\ \mathbf{H}\phi$

3. If $\mathbf{G} \sqsubset \mathbf{H}$ because more than one G_i/H_i or V_j/W_j pairs are affected, we set up a sequence $\mathbf{G}_1 \sqsubset \mathbf{G}_2 \sqsubset \ldots \sqsubset \mathbf{H}$ to satisfy part two of the proof.

6. The $\Sigma\lambda$-Fixpoint Theorem

In the associated $\Sigma\lambda$-calculus, we omit the κ-reductions and replace each occurrence of $\mathbb{K}\{G_1 \Rightarrow V_1\ \ldots\ G_n \Rightarrow V_n\}$ by $\phi S_n\ G_1\ G_2\ \ldots\ G_n\ V_1\ V_2\ \ldots\ V_n\phi$.

Any definition now of a functional in terms of the associated $\Sigma\lambda$-calculus is given in terms of compositions of monotonic functions, and hence the fixpoint theorem is applicable.

The function
$F(x) \Leftarrow \tau[F](x)$
will map input into lifted sets of possible output values.

Since the associated $\Sigma\lambda$-calculus yields all possible results, we see that a single application of McCarthy's **amb** function [6] yields a result that may be obtained in the $\Sigma\lambda$-calculus. The possibility of selecting $\perp$ even on computation where some guards are always $\mathtt{t}$ may either be considered to be a reflection on the theory or else as an emphatic statement that inappropriate reduction strategies may well lead to $\perp$ as the result. We prefer the latter interpretation.

7. Conclusion

The $\Sigma\lambda$-calculus as defined in §2 of this paper is not strong enough to allow fixpoints of recursive definitions to be established. In extending the calculus, however, by replacing the strong conditional with the more powerful relational conditional, we not only produce a system in which Kleene's fixpoint theorem applies, but define a structure which strongly reflects its intended semantics.

8. Acknowledgements

The research covered by this paper was supported by the FRD and junior lecturership funds from the University of Natal, Pietermaritzburg.

9. References

[1] Dijkstra, E W : Guarded Commands, Nondeterminacy and Formal Derivations of Programs. In *Communications of the ACM*,vol 18,pp.453-419, 1975.

[2] Hindley, J R ; Seldin, J P : *Introduction to Combinators and λ-Calculus*. Cambridge University Press, 1986.

[3] Kleene, S C : *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.

[4] Landin, P J : A λ-Calculus Approach. In *Advances in Programming and Non-Numerical Computation*. Pergammon, 1966.

[5] Manna, Z : *Mathematical Theory of Computation*. McGraw-Hill, New York, 1974.

[6] McCarthy, J : A Basis for a Mathematical Theory of Computation. In *Computer Programming and Formal Systems*. North-Holland, Amsterdam, (1963).

[7] Postma, S W : *Basic Definitions of Octolisp in Σλ-Calculus*. University of Natal, Computer Science Report PMB-TR/89-01, 1989.

[8] Postma, S W : *Octolisp Syntax and Language*. University of Natal, Computer Science Report, in preparation, 1989.

[9] Postma, S W & Phillips, N C K : *The Parallel Conditional*. In *QUAESTIONES INFORMATICAE,6*,3,1988, pp.109-112.

[10] Plotkin, G D : A Powerdomain for Countable Nondeterminism. In *Automata, Languages and Programming*. LNCS 140, Springer-Verlag, Berlin, 1982.

[11] Schmidt, D A : *Denotational Semantics : A Methodology for Language Development*. Allyn & Bacon Inc., Newton, Mass., 1986.

[12] Stoy, J E : *Denotational Semantics : The Scott-Strachey Approach to Programming Language Theory*. MIT Press, Cambridge, Mass., 1977.

[13] Tennent, R D : The Denotational Semantics of Programming Languages. In *Communications of the ACM*, vol.19, pp.437-452, 1976.

[14] Yasuhara, A : *Recursion Function Theory and Logic*. Academic Press, New York, 1971.

Book Reviews

Software Engineering
by Stephen R Schach, Richard D Irwin, Inc. and Aksen Associates, Inc., 1990
Reviewer: Mr N L O Cowley, University of Port Elizabeth

In my bookcase are a number of texts with the title "Software Engineering". Schach's new textbook is the latest one to bear this name that has crossed my desk. My task as a reviewer is to explain how this book differs from those other books and to evaluate its quality.

Software engineering books can be separated into two major categories. Specialised books Like "Software Reliability" by John D Musa, Anthony Iannino and Kazuhira Okumoto (McGraw-Hill 1987) deal with specific topics in depth, and presuppose a fairly deep knowledge and understanding of software engineering.

General books like Schach's text give broad overviews of the field. They are suitable for novice practitioners, and for senior or post-graduate students who lack a solid grounding in software engineering (and there are many of these). It will be some time before the market for such textbooks shifts down to the freshman level.

The book is divided into four parts. The first part provides the mandatory overview of the field and a jeremiad about the problems and difficulties associated with it.

The second part deals quite broadly with the software life-cycle. Two major themes that run through this section - and thorugh the entire book - are the importance of developing for economical maintenance, and the organic nature of testing throughout the software life-cycle. The survey of current life-cycle models, time-cost estimation and testing are particularly well handled.

The third part deals with phases of the software life-cycle in depth. The presentation of semi-formal and formal specifications and modularity was good, although I missed the presence of the model-based specification language, Z. Object-oriented design and real-time systems design are treated, albeit briefly.

The fourth part deals with some major current topics in software engineering. These are CASE, portability and reusability, Ada, and the vitally important topic of experimental work in software engineering to win data so that a sound theoretical foundation for the field can be built. Finally that holy grail, automatic programming, is discussed.

In general the book is pedagogically sound. The writing is clear and I liked the quantitative and semi-formal flavour of the text, which reflects a trend in the field. Software engineering has suffered much from excessive hand-waving in the past, and deserves better.

The book is well organised. At the end of each chapter is a chapter review, an annotated further reading guide, a set of problems and a chapter reference list. I found the problems interesting, particularly as many of them encourage divergent thinking. Answers to the questions are contained in the instructor's manual. The references are copious and an adequate number of recent references is provided. Indexes are provided at the end of the text.

I found the instructor's manual very useful. Besides containing answers to the questions, as previously mentioned, there are guidelines on the work distribution and central themes of each chapter. The pages in this manual can be used as overhead projector transparency masters.

I recommend this book and its accompanying instructors's manual as very sound texts around which to build an upper-level undergraduate or graduate course. its logical and clear structure, built-in reference, question and project resources and lack of excessive detail make it an easy book for a lecturer to use, yet a challenging one for students if the references listed in the book are skilfully employed.

Computer-Aided System Engineering
by Howard Eisner, Prentice-Hall International Editions, 1988.
Reviewer: Andrew Morris, Southern Life Association, Cape Town.

It is readily accepted that IT practitioners are in the business of 'engineering systems' and many techniques are being incorporated into the systems tool box. Although 'systems engineering' is an accepted term there is still considerable resistance to the level of investment required in order to create systems utilising these principles.

Eisner's book begins with a broad introduction to the concepts, and proceeds to outline the current tools and techniques, before looking at the more complex aspects. By developing the subject in this manner, it is difficult to argue the case for *not* applying rigorous control of systems development. In conclusion, the book looks at future developments in the fields of artificial intelligence and CASE tools, which clearly illustrates the dangers of applying these techniques without first adopting engineering principles.

There is a move towards implementing such methods as Quality Assurance (QA), but in isolation this is unlikely to prove successful. This book illustrates the major support functions for adopting systems engineering, of which QA is only one part. This provides a clear understanding that any segmentation increases the implementation effort whilst reducing the success rate. No longer can systems development occur as a 'shoot from the hip' process - too much wastage has occurred and too much is at stake.

As a student text the book provides a broad introduction with suitable references for further reading. For the practitioners, it is a first rate foundation book which should encourage the adoption of these techniques, since the book contains many real examples and progressively develops the subject with a combination of theory and practice. There are chapters focusing on basic mathematical theories which are simply explained and provide insight into the underlying principles.

The question that springs to mind is that the principles have been around for some time, so why only now are we beginning to look at them in relation to systems? This is clearly explained in the introductory chapter.

"Thus we seem to be influenced at this time by both a 'systems pull' and a 'technology push'. The systems pull requires better systems engineering formalisms and tools... The technology push leads with the ubiquitous digital computer chip... The evolution of our systems engineering capability, therefore, is proceeding within the context of revolutionary changes in computer hardware technology and the evolution of other related technology areas such as communications."

The only criticism of the book is the number of references to US Department of Defence documents, and the resulting impression that these techniques are mainly applicable to large scale systems. As the PC continues to develop and systems engineering software emerges to support the complete range, the principles can be applied at any level.

Eisner's book is a very useful, readable text, which will be of use to students and practitioners in gaining some understanding of the major influence shaping the industry.

Books Received

The following books have been sent to SACJ. Anyone with suitable credentials who is willing to review a book should contact the editor. The book will be sent for review, and may be kept provided that a review is received within a reasonable period.

- A L Decegama, [1989], *The Technology of Parallel Processing,* Prentice-Hall Inc., Englewood Cliffs.
- N Ford, [1989], *PROLOG Programming,* John Wiley & Sons.
- R C Detmer, [1990], *Fundamentals of ASSEMBLY LANGUAGE PROGRAMMING - Using the IBM PC and Compatibles,* D C Heath and Co., Canada, USA.
- S N Kamin, [1990], *Programming Languages - An Interpreter-Based Approach,* Addison-Wesley Publishing Company, Inc., USA.
- R A Mueller & R L Page, [1988], *Symbolic Computing with Lisp and Prolog,* John Wiley & Sons, Canada, USA.
- A M Tenenbaum, Y Langsam & M J Augenstein, [1990], *Data Structures Using C,* Prentice-Hall Inc., Englewood Cliffs.
- A S Tanenbaum, [1990], *Structured Computer Organization,* Third Edition, Prentice-Hall Inc., Englewood Cliffs.
- S Taylor, [1989], *Parallel Logic Programming Techniques,* Prentice-Hall Inc., Englewood Cliffs.

INSPEC to Index SACJ

INSPEC is a widely used international indexing agency of the Institute of Electrical Engineers in the UK. The first issue of SACJ was sent to INSPEC for consideration as a journal to be indexed. Contributors will be pleased to note the extract below from a letter in response to SACJ's application. However, it should be noted that contributions will be assessed individually, and that indexing is not automatic.:

Thank you for your letter of 24 January and for your copy of the above journal which you sent to us.

It has been assessed by our information scientists for suitable articles to be included in our database, and I am pleased to let you know that most of the articles included in the first issue were suitable for selection. We are therefore very happy to accept your offer of sending us your journal on publication.