HIFI Computing and Fixpoints
Cyrus F. Nourani

Extended Abstract

Abstract New tree computing techniques and an algebraic computing theory defining recursion tree amplification had been put forth by [Nourani 94]. The infinitary language category \( L_{1,B} \) had been defined. A generic language topology and a functor from \( L_{0,1,B} \) to Set are defined for a functorial model theory. The functorial model theory can define the formal semantics for computation with \( L_{0,1,B} \). The computations are fixpoints on both model definition and the tree computing notions. A preliminary recursive computing theory is defined on the categories defining recursion tree amplification by fixpoints on algebraic theories. The Amplification Principle, TAP, is put forth based on relating a minimal function set \( F \) to a recursion theoretic gain concept, defining computing efficiency by gain minimizations. A recursion theorem is stated in terms of HIFI principles

Keywords HIFI Logik, Tree Amplification, Fixpoints, Abstract Recursion, Infinitary Language Categories, Functorial Model Theory, Categorical Computing

Project METAAI@CompuServe.com
Affiliations METAAI and USA Academic UCSB

1. Functorial Model Theory and HIFI Computing

For the computing theories defined for HIFI logic the functorial model theory and the limit definitions are essential to get models defined. Functorial computing models are definable based on what we have presented in our 1994-96 papers. It is apparent we can define a computing theory based on a functorial model theory for INFLCS and HIFI computing. INFLCS can be viewed as an abstract language for defining computation. There is a categorical recursion theory defined by HIFI computing by which computing models can be defined for INFLCS.

2. G-diagrams for Initial Models

To present our model theory for categorical computing we present model theoretic diagrams referred to by G-diagrams. These diagrams are to be distinguished from the arrow diagrams for categories. There is a two diagram terminology in the present paper. Model-theoretic diagrams, we refer to them by M-Diagrams; and arrow diagrams, as in commutative diagrams. We refer to these by A-Diagrams. The G-diagrams are model-theoretic diagrams, defined by this author around 1980's, thus G-diagrams are algebraic M-Diagrams. The generalized diagram (G-diagram) is a diagram in which the elements of the structure are all represented by a minimal set of function symbols and constants, such that it is sufficient to define the truth of formulas only for the terms generated by the minimal set of functions and constant symbols. Such assignment implicitly defines the diagram. G-diagrams are M-diagrams. This allows us to define a canonical model of a theory in terms of a minimal family of function symbols. The minimal set of functions that define a G-diagram are those with which a standard model could be defined by a pair. Formal definition of diagrams are stated here, generalized to G-diagrams, and applied in the sections to follow.

Definition 2.1 A G-diagram for a structure \( M \) is a diagram \( D<A,G> \), such that there is a proper definition of a model-theoretic diagram by a specific set of function symbols. [1]

For set theory \( \Sigma_1 \) Skolem functions are a specific function set. Thus initial models could be characterized by their G-diagrams. Further practical and the theoretical characterization of models by their G-diagrams are presented by this author in the 80's. It builds the basis for some forthcoming formulations to follow, and the tree computation theories that we have put forth.

3. Computing at \( L_{0,1,B} \)

Let \( L_{0,1,B} \) be the least fragment of a well-behaved infinitary language \( K_{0,1} \), \( \omega \) obtained from \( L_{0,1} \), \( \omega \). The fragment had been defined by Keisler. \( L_{0,1,B} \) has been brought up to a functorial model theory by Nourani [1994,96] where a small-complete category \( L_{0,1,B} \) is defined from \( L_{0,1,B} \). The category is the preorder category defined by the formula ordering defining the language fragment. The objects are the
small fragment sets, and the arrows the preorder arrows. There is a two diagram terminology in the present paper. Model-We had defined a small-complete category $L_{1,B}$ from $L_{01,B}$. The category is the preorder category defined by the formula ordering defining the language fragment.

Lemma 3.2 $L_{01,B}$ is a small-complete category. [Nourani 1994]

Define a functor $F: L_{01,B} \rightarrow Set$ by a list of sets $M_n$ and functions $f_n$. The sets correspond to an initial structure [ADJ 75] on $L_{01,B}$. For example, to $f(t_1,t_2,... tn)$ in $L_{01,B}$ there corresponds the equality relation $f(t_1,t_2,...tn) = ft_1...tn$. Let us refer to the above functor by the generic model functor since we can show it can defines generic sets from language strings to form limits and models. It suffices to define the functor up to an initial model without being specific as to what the model is to have a generic functor. It is a fixpoint computation. Specific categorical fixpoint computations are defined.

Theorem 3.3 The generic model functor has a limit. [Nourani 1994-95].

4. Initial Tree Computing and Amplification

4.1 The TAP Computational View

To define TAP we start with defining gain functions. Let $F$ be a minimal set of functions, for example type constructors, from which a CTA and a standard initial model with signature $\Sigma$ is definable, where standard in the sense of the definitions above, and definable is in the model-theoretic sense.

Definition 4.2 A function $f$ is $\Sigma$-tree-computable iff $f$ is definable with $\Sigma$ and $f$ is a computable function. 

Let $\{f_0, f_1, f_2, ..., \}$ be the set of functions $\Sigma$ -tree-computable. Further let $d_{<Fi>}(f_i)$ be the recursion depth for $Fi$-trees appearing in $f_i$, with $Fi$ in $F$. Then a gain function for $f_i$ is defined from $d_{<Fi>}(f_i)$.

Let $G_{<F>}(f_i) = \sum d_{<Fi>}(f_i) x WSum(Fi)$, for all $Fi$'s in $F$, where $WSum$ is a weighted sum function defined from the $G_{<F>}$ for the subtrees formed from $Fi$ at each recursion level. The weighted sum function might be defined from the weight of the subtrees rooted by $Fi$ at each recursion level. The weight $W$ of a subtree $t_j$, denoted $W(t_j) = \sum G_{<Fm>}, Fm in t_j, Fm in F$. $G_{<Fi>} = 1$ for a nullary function $Fi$. For $Fi$ of arity $(t_1,...tn)$, $G_{<Fi>} = \sum \{W(t_1),...,W(t_n)\}$. The $WSum$ for $Fi$ is defined by induction as follows.

For a depth 0 term $Fi(t)$, $WSum(0)(Fi) = W(tj)$

For a depth $n$ term, $WSum(n)(Fi) = n x WSum(n-1)(Fi)$

$G_{<F>}$ for the computable functions $\{f_0, f_1, f_2, ..., \}$ on a signature $\Sigma$ might be defined by the Maximum \{ $G_{<F>}(f_i)$ \}. This is not the only way to define gain, but it is a way to start.

Brief Overview - For a signature $\Sigma$ and minimal functions $F$ for defining initial models on signature $\Sigma$ we have defined gain as follows. For a signature $\Sigma$ and a choice $F$, a gain set is defined for the functions on the signature by \{ $G_{<F>} (f_1), G_{<F>} (f_2), ..., \}$, where each $G_{<F>} (f_i)$ defines gain on $F$ for a function $f_i$ definable from a signature $\Sigma$. We have further defined gain for all computable functions on a signature $\Sigma$, to be $G_{<F>}(\Sigma) = \cup \{G_{<F>} (f_i): f_i a computable function definable on $\Sigma$ \}, where $\cup$ is the least upper bound operation. Thus we have defined gain functions for functions computed by a minimal functions set $F$. For a particular signature, and a set $F$ of minimal functions, we have defined gain on $F$ for all computable functions definable on the signature. This is only a starting formulation, based on what we had presented in our 1994-1996 papers, from which new tree computing results could be obtained, shedding light on how the choice for $F$ and a signature, leads to complex or simpler computation. These are topic for future research and outside the intended goals of the present paper.

The TAP PrincipleThe trees welcomed into the computations by the choice $F$, are amplified by recursion with gain $G_{<F>}$. Computational efficiency could be defined by gain minimization.

Measures for computational complexity based on TAP could be defined, thus allowing us to base the selection on the computing model and the computing that is intended to minimize gain, thus have efficient computation.

4.3 The Recursion Theorem
To get a start for what we can define with the basis for an abstract recursion theory based on the concepts defined by the present paper. The recursion theorem is the following.

Definition 4.3 Let X be any fixed set. A function F: P(X) \rightarrow P(X) is called an operator over X. F is said to be inclusive iff for all Y \subseteq X, Y \subseteq F(Y), monotone iff all Y \subseteq Z \subseteq X, F(Y) \subseteq F(Z), and inductive iff F is either inclusive or monotone. [ ]

We define by transfinite recursions the sequence F[a] by

F[a] = F(∪ { F k: k < a}) and set F& = U { F: a in Ordinals}.

We write F <a> for U { F k: k < a} so that F[a] = F(F <a>). Think of the set F& as being “built up” is stages. Starting with the empty set, F( ), F(F()),......are obtained. F[a] is called the a-th stage.

Inductive definitions are closures in the following sense. Let Y \subseteq X, F a family of finitary functions on X. Let S \subseteq X, F a set of finitary functions on X. Let Ar(f) be the arity f in F. For each pair (S,L) define an inductive operator F <S,L>(Y), for Y a subset of S by

F <S,L>(Y) = S \cup \{ f(y'): f in L & Y' in Ya \} F <S,L> is called the closure of S under F. Since F <S,L> is monotone, it is the smallest set inductively S-closed under L. Monomorphic pairs, well-known from abstract recursion, are <function,set> pairs where the functions are defined injective on disjoint subsets of the set.

Theorem 4.3 For any monomorphic pair (S,F) and any set X*, suppose G: X* \rightarrow X* and for each f in F, f*: X*Ar(f) \rightarrow X*, then there exists a unique function T : F <S,L> \rightarrow X* such that

(i) for all s in S, T (s) = G(s),
(ii) for all f in F and all x in F <S,L>, T(f(x)) = f* (T(x 0),...,T(x Ar (f)-1)).

Defining an abstract recursion for algebraic theories [ADJ 75] is another enterprise altogether. HIFI computing is combining abstract recursion with recursion on a functorial algebraic theory to define a computing theory. It is not defining semantics for an existing computing theory. Hence it has to define its foundations a new. Let us start with a gain fixpoint recursion theorem.

Theorem 4.4 For any nonmonomorphic pair (S,L), and a computation sequence defined on the signature, there is a unique gain function F <S,L> defined.

Proof For the nonmonorphic pair S,L, take F(S,L) to be the tree inductive closure for the functions S defined by the signature Σ. Since gain is defined on Σ-computable functions by inductive definitions, by definition 4.2, for each function f in S, it is sufficient to view the tree rooted with f, with maximum height. Let G:X \rightarrow X be a function on the f-trees, which for a computation sequence on a signature Σ, for each f in S, returns the f-tree with maximum height. By a well-known basic recursion theorem for algebraic theories, the function G, extends to a unique Σ -homorphism from the set of f-trees in the computation sequence to the Σ -algebra on X with the Σ -definable functions. Let us refer to it by morphism recursion. By theorem 4.3, there is a unique function

T: F (S,L) \rightarrow X, where X is the set of f-trees defined by the computation sequence, assigning terms from the inductive set to X such that

(i) T (s) = G(s) for all s in S appearing in the computation sequence;
(ii) for all f in F and all x in F <S,L>, T(f(x)) = f* (T(x ),...,T (x Ar (f)-1)).

Composing theorem 4.3 with morphism recursion for algebraic theories, we can obtain gain direct from G. Hence a unique gain is defined. [ ]

References