November 1989

Lambda Calculus,Conservative Extension and Structural Induction

Val Tannen
University of Pennsylvania

Ramesh Subrahmanyam
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-89-64.

Lambda Calculus,Conservative Extension and Structural Induction

Abstract

The issue of whether embedding algebraic theories in higher-order theories such as the simply typed and polymorphic lambda calculi is of interest in programming language design. The establishment of such a conservative extension result permits modularity in the verification of the correctness of datatype and function implementations. In earlier work [Breazu-Tannen & Meyer 1987a], [Breazu-Tannen & Meyer 1987b] and [Breazu-Tannen 1988], conservative extension results have been obtained for algebraic theories. However, in modelling inductive datatypes, the principle of structural induction needs to be admitted in the inference system, and the question of whether conservative extension holds in the presence of the principle of structural induction needs to be addressed. In this paper we look at the question of whether inductive algebraic theories are conservatively extended when embedded in the simply typed lambda calculus.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-89-64.

Lambda Calculus, Conservative Extension And Structural Induction

MS-CIS-89-64
LOGIC & COMPUTATION 16

Val-Breazu-Tannen
Ramesh Subrahmanyam

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

November 1989

Lambda Calculus,Conservative Extension and Structural Induction

Val Breazu-Tannen Ramesh Subrahmanyam
Department of Computer and Information Science
Moore School of Electrical Engineering
University of Pennsylvania,Philadelphia,Pa19104

November 30, 1989

Abstract

The issue of whether embedding algebraic theories in higher-order theories such as the simply typed and polymorphic lambda calculi is of interest in programming language design. The establishment of such a conservative extension result permits modularity in the verification of the correctness of datatype and function implementations. In earlier work [Breazu-Tannen & Meyer 1987a], [Breazu-Tannen & Meyer 1987b] and [Breazu-Tannen 1988], conservative extension results have been obtained for algebraic theories. However, in modelling inductive datatypes, the principle of structural induction needs to be admitted in the inference system, and the question of whether conservative extension holds in the presence of the principle of structural induction needs to be addressed. In this paper we look at the question of whether inductive algebraic theories are conservatively extended when embedded in the simply typed lambda calculus.

1 Introduction

The objective here is to establish the conservativity and extension of algebraic theories with structural induction by the corresponding higher order theory obtained by adding the pure simply typed lambda calculus and structural induction. The spirit of this work is very much along the lines of the work reported in [Breazu-Tannen & Meyer 1987a] [Breazu-Tannen & Meyer 1987b] and [Breazu-Tannen 1988]. Another objective is to understand the scope and limits of higher order reasoning. Colson's result [L.Colson 1989] regarding the expressibility of a more efficient algorithm for computing the *min* of two integers in a higher order calculus, than in an algebraic calculus raises the question of the powerfulness of computations in strongly typed theories. Further, there is also

the issue of the interference of higher-order computation with first-order computation. Thus an important question that was answered in [Breazu-Tannen 1988] was the following: if we embed the constants of an algebraic signature into a simply-typed calculus, does the set of theorems expressible over the algebraic signature remain the same? This question was answered in the affirmative.
The primary application of embedding algebraic theories in a lambda calculus arises in reasoning with abstract datatypes. Usually, a finite set of algebraic rules are used to define the datatype. However reasoning about programs requires a higher-order theory , say a lambda calculus. Thus when we combine these two systems we would like that the resultant system prove no more theorems expressible in these individual languages than before. This requirement is the foundation for a modularization of reasoning about correctness of data and program implementations.This requirement is called "conservative extension".
However most datatypes are inductive datatypes, and the use of the rule of structural induction in reasoning with them is presupposed. Further, as has been shown, the set of theorems provable by structural induction are true in the initial model of the datatype equations(which is the usually accepted semantics). Thus there is a need to evaluate the question of whether the conservative extension by the simply typed lambda calculus holds in the presence of structural induction.
In this paper,we will prove that indeed it is the case that the required conservative extension result holds. We will assume formal knowledge of the notions of equational proofs, axioms of the simply typed lambda calculus, rules for typing lambda expressions etc. However, we will discuss a few essential preliminaries. For more details on the simply typed lambda calculus [Barendregt 1984] may be consulted. [J.Goguen & J.Meseguer 1985] algebraic theories and structural induction

2 Algebraic Theories and Structural Induction

Consider a fixed many sorted algebraic signature Σ , and set of equations E. Further ,also consider a sort-indexed collection of countable sets of variables X. The set of algebraic terms over this signature, T_Σ(X) is the smallest set containing X, such that if f is an n-ary constant symbol of sort ($s_1..s_n$,s) in the signature, t_1,..,t_n are contained in the set and t_i is of sort s_i, then so is f$t_1..t_n$.
We use the metavariable Γ to denote a finite functional set of variable-sort ordered pairs. Further given the following two sort inference rules:

$$\frac{(x,s) \in \Gamma}{\Gamma \cdot x : s}$$

$$\frac{\Gamma \cdot t_i : s_i \qquad 1 \leq i \leq n}{\Gamma \cdot f t_1..t_n : s}$$

where the symbol f has arity ($s_1..s_n$,s).
If there is a proof tree with $\Gamma\cdot$ t:s for some sort s, then we say that that (Γ,t) type-checks.
We are going to be concerned only with one-sorted algebraic theories in the sequel. The result extends to the many sorted case as well. For simplicity of concept, we confine ourselves to one-sorted theories.
Let (Σ,E) be a given algebraic theory. We, then, define a (Σ,E)-proof tree as follows:
(a) $(\Gamma, \Delta) \vdash$ t = t, the one node tree is a (Σ,E)-proof tree.
(b) $(\Gamma, \Delta) \vdash t_1 = t_2$, where $(t_1 = t_2) \in$ E $\cup\Delta$, is a (Σ,E)-proof tree.
(c) If

$$\begin{array}{c} T_1 \\ (\Gamma, \Delta) \vdash t_1 = t_2 \end{array}$$

is a (Σ,E)-proof tree, then so is

$$\frac{\begin{array}{c} T_1 \\ (\Gamma, \Delta) \vdash t_1 = t_2 \end{array}}{(\Gamma, \Delta) \vdash t_2 = t_1}$$

.

(d) If $\begin{array}{c} T_1 \\ (\Gamma, \Delta) \vdash t_1 = t_2 \end{array}$, and $\begin{array}{c} T_2 \\ (\Gamma, \Delta) \vdash t_2 = t_3 \end{array}$ are (Σ,E)-proof trees, then so is

$$\frac{\begin{array}{c} T_1 \\ (\Gamma, \Delta) \vdash t_1 = t_2 \end{array} \qquad \begin{array}{c} T_2 \\ (\Gamma, \Delta) \vdash t_2 = t_3 \end{array}}{(\Gamma, \Delta) \vdash t_1 = t_3}$$

.

(e) If $\begin{array}{c} T_1 \\ (\Gamma, \Delta) \vdash t_1 = t_2 \end{array}$ is a (Σ,E)-proof tree, then so is

$$\frac{\begin{array}{c} T_1 \\ (\Gamma, \Delta) \vdash t_1 = t_2 \end{array}}{(\Gamma, \Delta) \vdash t_1\theta = t_2\theta}$$

(f) If $\begin{array}{c} T_1 \\ (\Gamma, \Delta) \vdash t_1 = t_2 \end{array}$ is a proof tree, then so is

$$\frac{\begin{array}{c}T_1\\(\Gamma,\Delta)\vdash t_1=t_2\end{array}}{(\Gamma,\Delta)\vdash C[t_1]=C[t_2]}$$

where C is a context.
A proof by induction uses one more rule ,the rule of Structural Induction, which is stated below. Since structural induction is done on a constructor signature, a constructor signature needs to be specified in addition to Σ and E. We will notate the constructor signature by Ω. Thus the Structural Induction rule is:

$$\frac{(\Gamma,\Delta)\vdash e[a_i\backslash x] \qquad (\Gamma,\Delta\cup\{e[d_j\backslash x]:1\leq j\leq arity(f_k)\})\vdash e[fd_1..d_n\backslash x]}{(\Gamma,\Delta)\vdash e}$$

Here the a_i are the nullary constants in the constructor signature, and the f_k are the other constants in Ω. A $\lambda^{\rightarrow}$ Ind(Σ,E)-prooftree is constructed using the same rules as a (Σ,E)-Ind proof tree , except that the following axiom gives rise to an additional one-node proof tree.
(β) (λ x. M)N = N[N/x] provided N is free for x in M.

Further, at every node in the proof tree the terms must be typeable by the type assigning function Γ.

3 Adding Algebraic Rules to $\lambda^{\leftarrow}$

When we add the algebraic terms to the simply-typed terms, we consider all unary constants and algebraic variables to be of a constant base type,say o. in the resultant system; a function of arity n is thought of as a constant of type $o^n \rightarrow o$. It is easy to see that the typing rules of $\lambda^{\rightarrow}$ will infer the type o for any algebraic term.

Lemma 1 Let $(\Gamma,\Delta)\vdash t_1=t_2$ be at the root of a $\lambda^{\rightarrow}$ proof tree T. Let w be any free variable in t_1 or t_2, which is not of base type, and hence has a type of the form $\sigma_1\rightarrow(\sigma_2\rightarrow\cdot\cdot(\sigma_n\rightarrow o)\cdot\cdot)$, for simple types σ_i. Let l be a fresh variable (we also assume that the sets of free and bound variables are disjoint). Then , let T' be the result of replacing every occurence of w by the $\lambda^{\rightarrow}$ term $\lambda\ x_1{:}\sigma_1..\lambda x_n{:}\ \sigma_n.l$ in the formulae in T, and adding l:o to every type assignment in T. Then, T is a $\lambda^{\rightarrow}$-Ind(Σ,E) proof tree.

Proof We look at every possible instance of the mutated inference rules and axioms, and show that they are valid instances. The proof is by induction, and the base case is the empty tree, for which the claim clearly holds. Assume that it holds for all trees of depth $\leq$ n. Consider a tree of depth (n+1). The various cases corrrespond to various deductions at the root.

Case(i) $(\Gamma, \Delta) \vdash$ t = t, where $\Gamma \vdash t : \tau$. Then,
$T' \equiv \Gamma \cup \{l : \mathrm{o}\} \vdash t[\lambda x_1..\lambda x_n.l \backslash w] = t[\lambda x_1..\lambda x_n.l \backslash w]$
is clearly a valid one node proof tree.

Case(ii) $T \equiv (\Gamma, \Delta) \vdash P[(\lambda x.M)N \backslash y] = P[M[N \backslash x] \backslash y]$
$T' \equiv (\Gamma \cup \{l : \mathrm{o}\}, \Delta) \vdash P[(\lambda x \cdot M)N \backslash y][\lambda x_1 \cdot\cdot \lambda x_n \cdot l \backslash w] = P[M[N \backslash x] \backslash y][\lambda x_1 \cdot\cdot \lambda x_n \cdot l \backslash w]$

Note that $\Gamma \cup \{l : \mathrm{o}\}$ typechecks both terms $P[(\lambda x \cdot M)N \backslash y][\lambda x_1 \cdot \cdot \lambda x_n \cdot l \backslash w]$ and $P[M[N \backslash x] \backslash y][\lambda x_1 \cdot \cdot \lambda x_n \cdot l \backslash w]$

Note also that $\mathrm{P}[(\lambda x \cdot M)N \backslash y][\lambda x_1 \cdot \cdot \lambda x_n \cdot l \backslash w] \equiv P'[\lambda x_1 \cdot \cdot \lambda\ x_n \cdot l \backslash w][(\lambda\ x.M')N' \backslash y]$, where $M' \equiv M[\lambda x_1 \cdot \cdot x_n \cdot l \backslash w]$, and $N' \equiv N[\lambda x_1 \cdot \cdot x_n \cdot l \backslash w]$

Thus $(\Gamma, \Delta') \vdash P[(\lambda x \cdot M)N \backslash y][\lambda x_1 \cdot \cdot \lambda x_n \cdot l \backslash w] = P[M[N \backslash x] \backslash y][\lambda x_1 \cdot \cdot \lambda x_n \cdot l \backslash w]$
(i.e.) $(\Gamma, \Delta') \vdash P'[(\lambda x M')N' \backslash y] = P'[M'[N' \backslash x] \backslash y]$, which is an instance of the β-rule, and hence T' is a valid proof tree.

Case(iii) $T \equiv (\Gamma, \Delta) \vdash t_1 = t_2$, where $(t_1{=}t_2) \in \Delta$ Clearly if $(t_1 = t_2) \in \Delta$, then $(t_1' = t_2') \in \Delta'$.
Hence, $T' \equiv (\Gamma, \Delta') \vdash t_1' = t_2'$ is a valid proof tree.

Case(iv) If

$$\frac{\begin{array}{c} T_1 \\ (\Gamma, \Delta) \vdash t_1 = t_2 \end{array}}{(\Gamma, \Delta) \vdash t_2 = t_1}$$

then T', the transformed tree is

$$\frac{\begin{array}{c} T_1' \\ (\Gamma^+, \Delta') \vdash t_1' = t_2' \end{array}}{(\Gamma^+, \Delta') \vdash t_2' = t_1'}$$

where, $t_1' \equiv t_1[\lambda x_1 \cdot\cdot \lambda x_n \cdot l \backslash w$, and $t_2' \equiv t_2[\lambda x_1 \cdot\cdot \lambda x_n\ dotl \backslash w$, and T_1' is T_1 transformed. Further we use the symbol Γ^+ to stand for $\Gamma \cup \{l : \mathrm{o}\}$. Clearly by induction

$$\begin{array}{c} T_1' \\ (\Gamma^+, \Delta') \vdash t_1' = t_2' \end{array}$$

is a valid proof tree, and

$$\frac{(\Gamma^+, \Delta') \vdash t_1' = t_2'}{(\Gamma^+, \Delta') \vdash t_2' = t_1'}$$

is an instance of the symmetry inference rule. Thus T' is a valid proof tree.

Case(v) We handle the case where the inference at the root is transitive likewise.

Case(vi) This is the case where the rule at the root of the tree is the substitutivity rule.

$$\frac{\begin{array}{c} T_1 \\ (\Gamma, \Delta) \vdash t_1 = t_2 \end{array}}{(\Gamma, \Delta) \vdash t_1\theta = t_2\theta}$$

transforms to

$$\frac{\begin{array}{c} T_1' \\ (\Gamma^+, \Delta') \vdash t_1' = t_2' \end{array}}{(\Gamma^+, \Delta') \vdash (t_1\theta)' = (t_2\theta)'}$$

By the induction hypothesis, T_1' is a valid proof tree. Note that $(t_i\theta') = (t_i'\theta')$, for i=1,2, where $\theta'(\mathrm{x}) = \theta(\mathrm{x})[\lambda x_1 \cdot \cdot \lambda x_n \cdot l \backslash w]$.
Therefore, the transformed tree is a valid proof tree.
Case(vii)

$$\frac{\begin{array}{c} T_1 \\ (\Gamma, \Delta) \vdash e \end{array}}{(\Gamma^+, \Delta^+) \vdash e}$$

transforms to

$$\frac{\begin{array}{c} T_1' \\ (\Gamma', \Delta) \vdash e' \end{array}}{(\Gamma'^+, \Delta'^+) \vdash e'}$$

It is easily seen that the transformed tree is a valid proof tree.
Case(viii) The rule at the root is the rule of replacement.

$$\frac{\overset{T_1}{(\Gamma,\Delta)\vdash e_1} \qquad \overset{T_n}{(\Gamma,\Delta)\vdash e_n}}{(\Gamma,\Delta)\vdash ft_1..t_n = fs_1..s_n}$$

where $e_i \equiv (t_i = s_i)$

$$\frac{\overset{T'_1}{(\Gamma^+,\Delta')\vdash e'_1} \qquad \overset{T'_n}{(\Gamma^+,\Delta')\vdash e'_n}}{(\Gamma^+,\Delta')\vdash ft_1..t_n[\lambda x_1..x_n\backslash w] = fs_1..s_n[\lambda x_1..x_n\backslash w]}$$

But, $\mathrm{f}t_1..t_n[\lambda x_1 \cdot \cdot \lambda x_n \cdot l\backslash\ \mathrm{w}] = \mathrm{f}t'_1..t'_n$, and $\mathrm{f}s_1..s_n[\lambda x_1 \cdot \cdot \lambda x_n \cdot l\backslash\ \mathrm{w}] = \mathrm{f}s'_1..s'_n$.
Thus the transformed tree is a valid proof tree.
Case(ix) The rule of struictural induction occurs at the root.

$$\frac{L_1..L_q \qquad M_1..M_p}{(\Gamma,\Delta)\vdash e}$$

$$\frac{L'_1..L'_q \qquad M'_1..M'_p}{(\Gamma,\Delta')\vdash e'}$$

where, L_j is the proof tree for $\Delta \vdash \mathrm{e}[a_j\backslash\mathrm{x}]$, a_j being the j'th nullary constant, and M_i is the proof tree for Δ,e$[u_1\backslash\mathrm{x}]$,..,e$[u_n\backslash\mathrm{x}] \vdash \mathrm{e}[f_i u_1..u_n\backslash\mathrm{x}]$, f_i being the i'th non-nullary constant and n being its arity.

$$(\Gamma',\Delta')\vdash e[a_j\backslash x][\lambda x_1..x_n.l\backslash w] \quad \overset{T'_1}{}$$

$$(\Gamma',\Delta')\vdash e[\lambda x_1..x_n.l\backslash w][a_j\backslash x] \quad \overset{T'_1}{}$$

$$\equiv (\Gamma',\Delta')\vdash e'[a_j\backslash x] \quad \overset{T'_1}{} \quad \equiv (\Gamma,\Delta)\cup\{e[x_j\backslash x] \mid 1\le j\le k_i\}\vdash e[f_i^{k_i}(x_1..x_n)\backslash x] \quad \overset{T_i}{}$$

transforms to

$$(\Gamma',\Delta')\cup\{e'[x_j\backslash x] \mid 1\le j\le k_i\}\vdash e'[f_i^{k_i}(x_1..x_n)\backslash x] \quad \overset{T'_i}{}$$

Clearly T' is a valid proof tree.

Lemma 2 Let C be a context with one hole **u**. Let T be a proof tree for $(\Gamma,\Delta)\vdash$ s=t. Then there is a proof T' for $(\Gamma^+, C(\Delta))\vdash C[u \leftarrow s] = C[u \leftarrow t]$, provided $(\Gamma^+, C[u \leftarrow s]$ typechecks, and Γ and Γ^+ assign the same type to s (and t)(Notation:

the set $C(\Delta)$ is defined as $C(l) = C(r) \mid (l = r) \in \Delta)$.

Proof. Consider the following transformation.

$$\frac{\overset{T_1}{(\Gamma,\Delta) \vdash t_1 = t_2} \qquad \overset{T_2}{(\Gamma,\Delta) \vdash t_2 = t_3}}{(\Gamma,\Delta) \vdash t_1 = t_3}$$

transforms to

$$\frac{\overset{T_1'}{(\Gamma, C(\Delta)) \vdash C[u \leftarrow t_1] = C[u \leftarrow t_2]} \qquad \overset{T_2'}{(\Gamma, C(\Delta)) \vdash C[u \leftarrow t_2] = C[u \leftarrow t_3]}}{(\Gamma, C(\Delta)) \vdash C[u \leftarrow t_1] = C[u \leftarrow t_3]}$$

Further,

$$\frac{\overset{T_1}{(\Gamma,\Delta) \vdash t_1 = t_2}}{(\Gamma,\Delta) \vdash t_2 = t_1} \qquad \Rightarrow \overset{T_1'}{(\Gamma, C(\Delta)) \vdash C[u \leftarrow t_2] = C[u \leftarrow t_1]}$$

where,

$$\overset{T_1}{(\Gamma,\Delta) \vdash t_1 = t_2} \quad \Rightarrow T_1'$$

When the root inference is using tyhe rule of replacement,

$$\frac{\overset{T}{(\Gamma,\Delta) \vdash s = t}}{(\Gamma,\Delta) \vdash r[u \leftarrow s] = r[u \leftarrow t]} \qquad \Rightarrow \qquad \frac{\overset{T}{(\Gamma, C(\Delta)) \vdash s = t}}{(\Gamma, C(\Delta)) \vdash C[r[u \leftarrow s]] = C[r[u \leftarrow t]]}$$

When the root inference is substitution we need to ensure that the free variables in the root equation are distinct from the free variables in the context in which everything is to be placed. We tus use a renaming substitution σ.

$$\frac{\overset{T}{(\Gamma,\Delta) \vdash s = t}}{(\Gamma,\Delta) \vdash s\theta = t\theta} \qquad \Rightarrow T_1'\sigma^{-1}$$

$$T_1 \equiv \quad \frac{\begin{matrix} T' \\ (\Gamma, \Delta) \vdash s_1 = t_1 \end{matrix}}{(\Gamma, \Delta) \vdash s_1\theta = t_1\theta}$$

where, $s_1 = s\sigma$, $t_1 = t\sigma$,and $(\mathrm{fv}(s_1) \cup \mathrm{fv}(t_1))$ are new and σ is a renaming substitution, and $T' = \mathrm{T}\sigma$ (i.e.) all free-variables are renamed using σ.
It is easy to see that $\mathrm{C}[s_1\theta] = \mathrm{C}[s_1]\theta$, and $\mathrm{C}[t_1\theta] = \mathrm{C}[t_1]\theta$. Looking at the case where the root inference uses structural induction,

$$\frac{\begin{matrix} T & S \\ (\Gamma, \Delta) \vdash e[c_i\backslash x] & (\Gamma, \Delta \cup \{e[x_i\backslash x] \mid i \leq n\}) \vdash e[f_j x_1..x_n\backslash x] \end{matrix}}{(\Gamma, \Delta) \vdash e}$$

transforms to

$$\frac{\begin{matrix} T' & \\ (\Gamma^+, C(\Delta\sigma)) \vdash C[e[c_i\backslash x]] & S' \end{matrix}}{\begin{matrix} (\Gamma^+, C(\Delta\sigma) \cup \{C[e[x_i\backslash\sigma(x)]] \mid 1 \leq i \leq n\}) \vdash C[e\sigma[f_j x_1..x_n\backslash\sigma(x)]] \\ (\Gamma^+, C(\Delta\sigma)) \vdash C[e\sigma] \end{matrix}}$$

Clearly, since $\mathrm{C}[e[c_i\backslash x\sigma]] = C[e](c_i\backslash x\sigma)$, the form of the induction rule is preserved. That the transformation t $\Rightarrow$ C[t] preserves the structure of axioms is easy to see. Further it is easy to see that the transformations presented do not increase the depth of the tree. This fact will be used in applications of the present lemma.

Lemma 3. Let T be a proof of $\Delta \vdash$e:o. Then we can effectively transform it into a proof T' of $\Delta \vdash e$, where every equation is of base type.
Proof. If T is a one node tree , then the theorem holds trivially. If T is of the form

$$\frac{\begin{matrix} T_1 & T_2 \\ t_1 = t_2 & t_2 = t_3 \end{matrix}}{t_1 = t_3}$$

then, by induction $\exists T_1' \exists T_2$.

$$\begin{matrix} T_1 & T_1' \\ t_1 = t_2 \ \Rightarrow & t_1 = t_2 \end{matrix}$$

$$\begin{array}{c} T_2 \\ t_2 = t_3 \end{array} \Rightarrow \begin{array}{c} T_2' \\ t_2 = t_3 \end{array}$$

$$\mathrm{T} \Rightarrow \dfrac{\begin{array}{cc} T_1' & T_2' \\ t_1 = t_2 & t_2 = t_3 \end{array}}{t_1 = t_3}$$

When the root inference is either symmetry or induction, the argument is similar. If T is of the form

$$\dfrac{\begin{array}{c} T_1 \\ s = t \end{array}}{r[u \leftarrow s] = r[u \leftarrow t]}$$

by the previous lemma the tree

$$\begin{array}{c} T_1 \\ s = t \end{array}$$

can be transformed into a proof tree T_2 for r[u$\leftarrow$ s] = r[u $\leftarrow$ t] of depth atmost equal to that of T. By the induction hypothesis, there exists a prooftree T_2' corresponding to T_2 all of whose nodes only contain equations of base type. This is the required transform of T.

If T is of the form

$$\dfrac{\begin{array}{c} T_1 \\ s = t \end{array}}{s\theta = t\theta}$$

then

$$\dfrac{\begin{array}{c} T_1' \\ s = t \end{array}}{s\theta = t\theta}$$

is the requird proof, where T_1' is the transform of T_1.

Lemma 4 Consider the following transformation on each node of a proof,every equation in which is of base type and has no variables of higher type free variables. $\Delta \vdash e \Rightarrow nf(\Delta) \vdash nf(e)$, where nf($\Delta$) = $\{nf(l) = nf(r) \mid (l = r) \in \Delta\}$, and e$\equiv$ (l = r) $\Rightarrow$ nf(e) $\equiv$ nf(l) = nf(r). This transformation transforms a valid proof into another valid proof.

Proof.

$$\frac{nf(t_1) = nf(t_2) \qquad nf(t_2) = nf(t_3)}{nf(t_1) = nf(t_3)}$$

$$\frac{nf(t_1) = nf(t_2)}{nf(t_2) = nf(t_1)}$$

$$\frac{nf(\Delta) \vdash nf(e[c_i \backslash x_i]) \qquad nf(\Delta), nf(e[x_i \backslash x]) \vdash nf(e[f x_1..x_n \backslash x])}{nf(\Delta) \vdash nf(e)}$$

are valid inference rule instances.

Note that s and t have no higher type free-variables. We require that the free variables in the range og θ(restricted to the free variables of s and t) should be disjoint from bound variables in s and t. Thus nf($s\theta$) = nf(s)[nf(θ)]. The rule,now, has the form

$$\frac{nf(s) = nf(t)}{nf(s)[nf(\theta)] = nf(t)[nf(\theta)]}$$

which is a substitutivity instance.

Thus given any proof tree of an equation in $\lambda^{\leftarrow}$E with structural induction, we first transform it into a proof with no free variables of higher type(lemma 1). We then rewrite it into a proof each of whose nodes is labelled by an equation of base type(lemma 3). We then normalize each node to give a proof tree each of whose nodes is labelled by an algebraic equation (lemma 4).

4 Conclusion

We have demonstrated a set of proof tree transformations that can be used to translate any proof in $\lambda^{\leftarrow}$E with structural induction into an algebraic proof with structural induction. This extends the earlier results of [Bre 87] for the case without structural induction.

References

[Barendregt 1984] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Volume 103 of *Studies in Logic and*

the Foundations of Mathematics, North-Holland, Amsterdam, second edition, 1984.

[Breazu-Tannen & Meyer 1987a] V. Breazu-Tannen and A. R. Meyer. Computable values can be classical. In *Proceedings of the 14th Symposium on Principles of Programming Languages*, pages 238–245, ACM, January 1987.

[Breazu-Tannen & Meyer 1987b] V. Breazu-Tannen and A. R. Meyer. Polymorphism is conservative over simple types. In *Proceedings of the Symposium on Logic in Computer Science*, pages 7–17, IEEE, June 1987.

[Breazu-Tannen 1988] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings of the Symposium on Logic in Computer Science*, pages 82–90, IEEE, July 1988.

[J.Goguen & J.Meseguer 1985] J.Goguen and J.Meseguer. Initiality, induction and computability. In M.Nivat and J.C.Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–543, Cambridge University Press, Cambridge, 1985.

[L.Colson 1989] L.Colson. On primitive recursive algorithms. In M.Dezani-Ciancaglini G.Ausiello and S.R.Della Rocca, editors, *International Colloquium on Automata, Languages, and Programming*, pages 194–206, European Association for Theoretical Computer Science, Springer-Verlag, Berlin, July 1989.