To be presented at the 8th Symposium on Computer and Information Sciences 1993

CSM-195

Transformations as Proofs

Martin C. Henson,
Department of Computer Science, University of Essex, Colchester, Essex, ENGLAND.
hensm@uk.ac.sx

§1 Abstract

This paper is a companion to [Hen93] which explores in depth the relationship between transformational programming and intuitionistic proofs in a theory of operations and types. Here we concentrate on discussing the extension of the theoretical development to algebraic data types and illustrating the techniques with an example.

§2 Introduction and background

We are concerned primarily with the task of uncovering the precise mathematical proofs which underlie certain semi-formal arguments. In the area of program development the program transformations [BuD77] are an excellent example of semi-formality, since, as is well known, the calculus of transformations is not, in general, sound[1], and it is capable of effecting significant shifts in logical complexity by, what superficially appears to be, equational manipulation. In [Hen93] we provided an interpretation of certain transformations over N as derivations within a theory of operations and types (the theory $\mathcal{EON}$ of [Bee85]). The main results can be summarised as follows:

Let π range over the transformations $\mathcal{PT}$. The interpretation [_] maps $\mathcal{PT}$ into $\boldsymbol{Der}(\mathcal{EON}) + \{\boldsymbol{fail}\}$ where $\boldsymbol{Der}(\mathcal{EON})$ is the set of proof derivations in the theory $\mathcal{EON}$.

Theorem 2.1 *For every $\pi \varepsilon \mathcal{PT}$, if $[\pi]$ is not* ***fail*** *then the computational content of $[\pi]$ is equal (up to the intensional equality of the underlying theory of operations) to the final system of equations of π.*

In [Hen93] this theorem is stated more precisely via the system $\mathcal{EON}^{\mathcal{TA}}$ which captures the notion of the *computational content of a derivation* exactly. As a corollary of this theorem we obtain our correctness conditions.

Theorem 2.2 *Let $\pi \varepsilon \mathcal{PT}$. If $[\pi] \varepsilon$ $\boldsymbol{Der}(\mathcal{EON})$ then π is correct.*

In §3 we discuss the extension of the analysis summarised above to a more ambitious class of inductive data types: the positive algebraic types (of which N is a simple member) and then in §4 we illustrate the techniques we describe via examples.

1 A transformation is *correct* when the function computed by the initial operation is equal to the function computed by the final operation. Unrestricted use of the *folding* transformation may not preserve correctness.

§3 Transformations over arbitrary algebraic types as proofs

§3.1 Generalising $\mathcal{EON}$ to $\mathcal{TK}$

The algebraic types are those constructed by the disjoint union of cartesian products of type variables, constants and recursion. We give the general case and some examples, which we use later in §4. We utilise a notation similar to Miranda [Tur85][2]:

tree	::=	Leaf *num* | Node *tree tree*
list	::=	Nil | Cons *num list*
num	::=	Zero | Succ *num*
T	::=	$\mathsf{DC}_0\,\boldsymbol{T}_0 \mid \ldots \mid \mathsf{DC}_n\,\boldsymbol{T}_n$

where each $\boldsymbol{T}_i = T_{i0} \ldots T_{im_i}$ and where T and each T_{ij} are type variables.

The first stage is the generalisation of the theory $\mathcal{EON}$ of [Bee85]. This is simply accomplished by replacing the rules for N by rules which capture the least fixpoints of positive type operations[3].

$$\frac{\Gamma \vdash z \in B(\Xi(\lambda X.B))}{\Gamma \vdash z \in \Xi(\lambda X.B)}\ (\Xi\text{–}intro) \qquad\qquad \frac{\Gamma \vdash B(T) \subseteq T}{\Gamma \vdash \Xi(\lambda X.B) \subseteq T}\ (\Xi\text{-}elim)$$

Intuitively, $\Xi(\lambda X.B)$ is the smallest type closed under the operation B. We also need to add rules for comprehension types:

$$\frac{\Gamma \vdash z \in \{x \mid \varphi(x \leftarrow z)\}}{\Gamma \vdash \varphi(x \leftarrow z)} \qquad\qquad \frac{\Gamma \vdash \varphi(x \leftarrow z)}{\Gamma \vdash z \in \{x \mid \varphi(x \leftarrow z)\}}$$

Small adjustments are also required to the rules governing definedness of the underlying partial logic but these are not central in this context and we omit the details. The new theory is, in fact, the theory $\mathcal{TK}$ of, for example, [Hen92].

The algebraic types now are special cases formed by careful choice of the operation B. Taking $B(X) =_{\text{def}} \{\mathsf{DC}_0\} \times \prod(\boldsymbol{T}_0[T \leftarrow X]) + \ldots + \{\mathsf{DC}_n\} \times \prod(\boldsymbol{T}_n[T \leftarrow X])$[4] we obtain the expected rules for T as special cases of Ξ–*intro* and Ξ-*elim* including, in particular:

$$\frac{\boldsymbol{x}_0 \in \boldsymbol{T}_0,\ \Psi_0(\boldsymbol{y}_0) \vdash \psi(\mathsf{DC}_0\,\boldsymbol{x}_0)\ \ldots\ \boldsymbol{x}_n \in \boldsymbol{T}_n,\ \Psi_n(\boldsymbol{y}_n) \vdash \psi(\mathsf{DC}_n\,\boldsymbol{x}_n)}{x \in T \vdash \psi(x)}\ (T\text{-}elim)$$

where each $\boldsymbol{x}_i = x_{i0} \ldots x_{im_i}$ and where $\Psi_i(\boldsymbol{y}_i) = \psi(y_{i0}), \ldots, \psi(y_{ik_i})$ with the y_{ij} distinct variables among the $\boldsymbol{x}_i$ such that, if $y_{ij} = x_{pq}$ then $T_{pq} = T$ and if $T_{pq} = T$ then $y_{ij} = x_{pq}$ for some y_{ij}.

As in [Hen93] we require a *term assignment* version of $\mathcal{TK}$ which we denote $\mathcal{TK}^{\mathcal{TA}}$. Of significance is the rule corresponding to Ξ-*elim* which is:

$$\frac{\Upsilon \vdash f : B(T) \subseteq T}{\Upsilon \vdash \boldsymbol{irec}\, f : \Xi(\lambda X.B) \subseteq T}$$

where ***irec*** satisfies the equation: $\boldsymbol{irec}\, f\, x = f\, x\, (B\, (\boldsymbol{irec}\, f)\, x)$. Otherwise the theory $\mathcal{TK}^{\mathcal{TA}}$ follows the pattern set by $\mathcal{EON}^{\mathcal{TA}}$.

2 Miranda is a Trademark of Research Software Limited.

3 These significantly extend the algebraic types but are syntactically much easier to manage.

4 $\prod$ denotes iterated cartesian product.

§3.2 Extending the translations

We now turn to the transformations and their interpretation within $\mathcal{TK}$. For the most part the translation remains intact. There are two places where there are significant changes and we give them here. They concern the transformation steps known as *instantiation* and *serious folding*[5].

$$\frac{(f\,\boldsymbol{p}, p(x), \boldsymbol{q} = e)[x \leftarrow (\mathsf{DC}_0\,\boldsymbol{x}_0)] \quad \cdots \quad (f\,\boldsymbol{p}, p(x), \boldsymbol{q} = e)[x \leftarrow (\mathsf{DC}_n\,\boldsymbol{x}_n)]}{f\,\boldsymbol{p}, p(x \in T), \boldsymbol{q} = e} \qquad (ins)$$

$$\frac{e_0 \subseteq e \qquad f\,\boldsymbol{x} = d \,\varepsilon\, \mathcal{L} \qquad e_0 \lesssim_\theta d \qquad f\,\boldsymbol{p} = e[e_0 \Leftarrow f\,\boldsymbol{x}\theta]}{f\,\boldsymbol{p} = e} \qquad (fld)$$

The first rule is fairly simple. An instantiation generates a family of new equations, one for each summand in the datatype T. We are letting p range over patterns and $\boldsymbol{p}$ (etc.) over sequences of patterns. The second rule shows how the equation $f\,\boldsymbol{p} = e$ is converted to the equation $f\,\boldsymbol{p} = e[e_0 \Leftarrow f\,\boldsymbol{x}\theta]$ where $[\, e_0 \Leftarrow e_1 \,]$ indicates the replacement of a *specified* occurrence of e_0 by e_1. The auxiliary data shows that e_0 occurs as a subexpression of e, $f\,\boldsymbol{x} = d$ is the defining equation for f and e_0 is a substitution instance of d (*via* the substitution θ). Let β range over equations and B over *ensembles* of equations, then we may write $\beta \twoheadrightarrow_{\mathrm{I}} \mathrm{B}$ for an instantiation and $\beta_0 \twoheadrightarrow_{\mathrm{F}} \beta_1$ for a fold when the rest of the data is understood. We can highlight the terminal ensemble of π by writing $\pi(\mathrm{B})$ and one equation among that ensemble by writing $\pi(\beta)$.

As before we must define a map $[\,\_\,] \,\varepsilon\, \mathcal{PT} \rightarrow \boldsymbol{Der}(\mathcal{TK}) + \{\boldsymbol{fail}\}$ by recursion over the structure of transformation trees. We will take $f\,\boldsymbol{x} = e_r$ to be the eureka equation where $f \in \boldsymbol{T} \rightarrow T$. In what follows $\varphi[\,\_\,]$ will always mean the formula $(\exists y \in T)(y = \_)$[6]. Then we can immediately set the base case of the translation (when the transformation consists simply of the eureka definition) to be: $[f\,\boldsymbol{x} = e_r] =_{\mathrm{def}} \boldsymbol{x} \in \boldsymbol{T} \vdash \varphi[e_r]$. For the main cases of interest let us suppose that we have a transformation $\pi = \pi_0(\beta \twoheadrightarrow_{\mathrm{X}} \mathrm{B})$ where X is a prime transformation step.

Case X = I: Let $\boldsymbol{z}$ be the variables occuring in the patterns $\boldsymbol{p}$, p and $\boldsymbol{q}$ other than x. Let $\boldsymbol{u}$ be a sequence of variables occuring in the sequence $\boldsymbol{z}x$.

In the derivation fragment below $\Theta_0(\boldsymbol{u}_0)$ is the sequence $\Theta(\boldsymbol{u})$ with a formula of the form $\zeta(x)$ *should it occur* removed and if Φ is a sequence of of formulae then $\Phi \Rightarrow \varphi$ is given by: $\Rightarrow \psi$ is ψ and $\Phi, \varphi \Rightarrow \psi$ is $\Phi \Rightarrow (\varphi \Rightarrow \psi)$. Also, we say that a formula is *standard* if it has the form: $(\forall \boldsymbol{z} \in \boldsymbol{T})(\Theta(\boldsymbol{u}) \Rightarrow \varphi[e])$ and where each formula in $\Theta(\boldsymbol{u})$ *if any* are standard.

$$\left[\;\frac{\cdots\cdots \quad \dfrac{\cdots \quad (f\,\boldsymbol{p}, p(x), \boldsymbol{q} = e)[x \leftarrow (\mathsf{DC}_i\,\boldsymbol{x}_i)] \quad \cdots}{f\,\boldsymbol{p}, p(x \in T), \boldsymbol{q} = e} \quad \cdots\cdots}{\pi_0}\;\right]$$

5 The following are rules for constructing trees which represent the transformations, they are *not* proof rules.

6 $\varphi(\boldsymbol{x})$ will, as usual, distinguish $\boldsymbol{x}$ among the free variables of φ.

$$\dfrac{\dfrac{\dfrac{\dots \qquad \dfrac{x_i \in T_i, z \in T, \Theta_0(u_0), \Psi_i(y_i) \vdash \varphi[e[x \leftarrow \mathrm{DC}_i\, x_i]]}{x_i \in T_i, \Psi_i(y_i) \vdash (\forall z \in T)(\Theta_0(u_0) \Rightarrow \varphi[e[x \leftarrow \mathrm{DC}_i\, x_i]])} \qquad \dots}{x \in T \vdash (\forall z \in T)(\Theta_0(u_0) \Rightarrow \varphi[e])}}{\dots\dots \quad z \in T, x \in T, \Theta(u) \vdash \varphi[e] \quad \dots\dots}}{[\,\pi_0\,]}$$

We have, here, assumed that the open sequent corresponding to the equation being instantiated has a particular form. This assumption is, of course, warranted but the proof must be omitted from this short account. However, it is easy to see that if $\Theta(u)$ is standard then $\Psi_i(y_i)$ will be too.

Case X = F:

$$\left[\;\dfrac{\dfrac{e_0 \subseteq e \qquad f\,x = e_r \,\varepsilon\, \mathcal{L} \qquad e_0 \leq_\theta e_r \qquad f\,p = e[e_0 \Leftarrow (f\,x)\theta]}{\dots\dots f\,p = e \dots\dots}}{\pi_0}\;\right]$$

is:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma^- \vdash \Theta(u)\xi_0} \qquad \dfrac{\dfrac{}{\Gamma \vdash (\forall z \in T)(\Theta(u) \Rightarrow \varphi[e_1])}}{\Gamma \vdash \Theta(u)\xi_0 \Rightarrow \varphi[e_0]}}{\Gamma \vdash \varphi[e_0]} \qquad \dfrac{\Gamma \vdash \varphi[\,e_0 \Leftarrow w] \qquad \dfrac{}{\Gamma, w = e_0 \vdash w = e_0}}{\Gamma, w = e_0 \vdash \varphi[e]}}{\dfrac{\dots\dots \quad \Gamma \vdash \varphi[e] \quad \dots\dots}{[\,\pi_0\,]}}$$

where the assumption $(\forall z \in T)(\Theta(u) \Rightarrow \varphi[e_1])$ is chosen such that $e_0 \leq_{\xi_0} e_1 \leq_{\xi_1} e_r$[7] and Γ^- is the context Γ with the assumption $(\forall z \in T)(\Theta(u) \Rightarrow \varphi[e_1])$ removed. The derivation above is then completed, for each of the formulae comprising $\Theta(u)\xi_0$, as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{(\Gamma^-, v \in S, \Phi(w))^- \vdash \Phi_0(w_0)\theta_0} \qquad \dfrac{\dfrac{}{\Gamma^-, v \in S, \Phi(w) \vdash (\forall v_0 \in S_0)(\Phi_0(w_0) \Rightarrow \varphi[e_3])}}{\Gamma^-, v \in S, \Phi(w) \vdash \Phi_0(w_0)\theta_0 \Rightarrow \varphi[e_3\theta_0]}}{\Gamma^-, v \in S, \Phi(w) \vdash \varphi[e_2]}}{\Gamma^-, v \in S \vdash \Phi(w) \Rightarrow \varphi[e_2]}}{\Gamma^- \vdash (\forall v \in S)(\Phi(w) \Rightarrow \varphi[e_2])}$$

7 Note that this forces $\xi_0\xi_1 = \theta$ and that no assumption need exist with this property. In this case the translation denotes ***fail***.

choosing the assumption $(\forall v_0 \in S_0)(\Phi_0(w_0) \Rightarrow \varphi[e_3])$ so that $e_2 \leq_{\theta_0} e_3$. As expected the context $(\Gamma^-, v \in S, \Phi(w))^-$ is the context $\Gamma^-, v \in S, \Phi(w)$ with the assumption $(\forall v_0 \in S_0)(\Phi_0(w_0) \Rightarrow \varphi[e_3])$ removed. The process is still incomplete but the task at hand (completing the derivation above each component formula of $\Phi_0(w_0)\theta_0$) is solved by the same strategy. This then calls into question the termination of the entire process. In order to terminate we have to ensure that eventually along each path of the process an assumption of the form $(\forall z \in T)(\ \varphi[e_1])$ is selected. We do this by showing, by transfinite induction, that the contexts from which the assumptions are drawn become less complex. We begin by assigning a *rank* to standard formulae: if $\Theta(u)$ is a sequence of formulae $\zeta_0(u_0) \ldots \zeta_v(u_v)$ then $rank((\forall z \in T)(\Theta(u) \Rightarrow \varphi[e])) = 1+\boldsymbol{max}\{rank(\zeta_i(u_i)) \mid 0 \leq i \leq v\}$. Evidently *rank* is a map from such formulae into the ordinals below ω. We now extend this to contexts: $rank(z \in T, \Gamma) = rank(\Gamma)$, $rank(\kappa, \Gamma) = \omega^{rank(\kappa)} + rank(\Gamma)$ when κ is a standard formula. We now note that $lim\{rank(\Gamma) \mid \Gamma \textit{ is a standard context}\} = \omega^\omega$ so induction up to ω^ω will suffice if we can be sure that the process reduces the rank at each stage. But this is straightforward: the rank of the context Γ, from which our first assumption $(\forall z \in T)(\Theta(u) \Rightarrow \varphi[e_1])$ is drawn, is given by $rank(\Gamma) = \omega^{rank(\kappa_0)} + rank(\Gamma^-)$ where κ_0 is $(\forall z \in T)(\Theta(u) \Rightarrow \varphi[e_1])$. On the other hand the rank of the context $\Gamma^-, v \in S, \Phi(w)$, from which the subsequent assumption $(\forall v_0 \in S_0)(\Phi_0(w_0) \Rightarrow \varphi[e_3])$ is drawn, is given by $rank(\Gamma^-, v \in S, \Phi(w)) = rank(\Gamma^-) + rank(\Phi(w))$ so we must show that $rank(\Phi(w)) <_{\omega^\omega} \omega^{rank(\kappa_0)}$. We know that $rank(\kappa_1) <_{\omega^\omega} rank(\kappa_0)$ where κ_1 is $(\forall v \in S)(\Phi(w) \Rightarrow \varphi[e_2])$ because $\kappa_1 \,\varepsilon\, \Theta(u)\xi_0$, and $\Theta(u)$ occurs in κ_0, hence $\omega^{rank(\kappa_1)} <_{\omega^\omega} \omega^{rank(\kappa_0)}$and similarly $rank(\kappa_2) <_{\omega^\omega} rank(\kappa_1)$ for each $\kappa_2 \,\varepsilon\, \Phi(w)$ hence $rank(\Phi(w)) <_{\omega^\omega} \omega^{rank(\kappa_1)}$ and thus $rank(\Phi(w)) <_{\omega^\omega} \omega^{rank(\kappa_0)}$ as required.

§3.3 Properties of the translation

With the extended translation in place we may prove analogous results to those announced in §2.

Theorem 3.3.1 *If* $(\beta)\pi(\mathrm{B}) \,\varepsilon\, \mathcal{PT}$, *and* $[\,\pi\,] \,\varepsilon\, \boldsymbol{Der}(\mathcal{TK})$ *then if* $t : [\,\pi\,]$ *in* $\mathcal{TK}^{\mathcal{TA}}$ *then* $\vdash t = \mathrm{B}$.

Theorem 3.3.2 *Let* $\pi \,\varepsilon\, \mathcal{PT}$. *If* $[\,\pi\,] \,\varepsilon\, \boldsymbol{Der}(\mathcal{TK})$ *then* π *is correct.*

The proofs of these are not dissimilar to those given in [Hen93].

§4 An illustrative example

We have chosen to illustrate the extended translation by taking an example transformation over a data type of trees. Moreover, we have chosen the example because it involves a nested instantiation which gives rise to standard formulae in the context which are not of the lowest rank. Thus in translating the subsequent serious fold it is necessary to undertake a short[8] instance of the inductive process described in §3.2. The transformation yields a linear operation from a (worst case) quadratic one. The function we deal with with takes an arbitrary element of *Tree* and yields another which has the same fringe of leaves but is left-linear. That is, it satisfies the specification:

$(\forall s_0 \in tree)(\exists s_1 \in tree)(\textit{left-linear}(s_1) \wedge \textit{eq-fringe}(s_0, s_1))$

with the predicates given by:

8 Very short - this example requires only *one* extra stage in the process which we have demonstrated may require, in full generality, a finitely branching tree of stages in which each path in the tree is bounded by induction to ω^ω!

$$\frac{}{\textit{left-linear}(\mathsf{Leaf}\ n)} \qquad \frac{\textit{left-linear}(s)}{\textit{left-linear}(\mathsf{Node}\ (\mathsf{Leaf}\ n)\ s)} \qquad \frac{\textit{eq-fringe}(s_0, s_1)}{\textit{fringe}(s_0) = \textit{fringe}(s_1)}$$

where:

$$\begin{aligned}
\textit{fringe}\ s &= \textit{frg}\ s\ \mathsf{Nil}\\
\textit{frg}\ (\mathsf{Leaf}\ n)\ l &= \mathsf{Cons}\ n\ l\\
\textit{frg}\ (\mathsf{Node}\ s_0\ s_1)\ l &= \textit{frg}\ s_0\ (\textit{frg}\ s_1\ l)
\end{aligned}$$

We begin with the initial operation:

$$\begin{aligned}
\textit{rotate}\ (\mathsf{Leaf}\ n) &= \mathsf{Leaf}\ n\\
\textit{rotate}\ (\mathsf{Node}\ s_0\ s_1) &= \textit{join}\ (\textit{rotate}\ s_0)\ (\textit{rotate}\ s_1)
\end{aligned}$$

where:

$$\begin{aligned}
\textit{join}\ (\mathsf{Leaf}\ n)\ s &= \mathsf{Node}\ (\mathsf{Leaf}\ n)\ s\\
\textit{join}\ (\mathsf{Node}\ s_0\ s_1)\ s_2 &= \mathsf{Node}\ s_0\ (\textit{join}\ s_1\ s_2)
\end{aligned}$$

which may be derived in the term assignment system $\mathcal{TK}^{\mathcal{TA}}$ by induction over *tree* given the function *join* and the properties that *join* preserves linearity and the fringes of its arguments.

We adopt the very simple eureka definition: *rote* s = *rotate s* and the transformation then proceeds:

$$\begin{aligned}
\textit{rotate}\ (\mathsf{Leaf}\ n) &= \textit{rote}\ (\mathsf{Leaf}\ n) && \textbf{Instantiate}\\
&= \mathsf{Leaf}\ n. && \textbf{Unfold}\\
\textit{rote}\ (\mathsf{Node}\ (\mathsf{Leaf}\ n)\ s) &= \textit{rotate}\ (\mathsf{Node}\ (\mathsf{Leaf}\ n)\ s) && \textbf{Instantiate}\\
&= \textit{join}\ (\textit{rotate}\ (\mathsf{Leaf}\ n))\ (\textit{rotate}\ s_1) && \textbf{Unfold}\\
&= \textit{join}\ (\mathsf{Leaf}\ n)\ (\textit{rotate}\ s_1) && \textbf{Unfold}\\
&= \textit{join}\ (\mathsf{Leaf}\ n)\ (\textit{rote}\ s_1). && \textbf{Fold}\\
\textit{rote}\ (\mathsf{Node}\ (\mathsf{Node}\ s_0\ s_1)\ s_2) &= \textit{rotate}\ (\mathsf{Node}\ (\mathsf{Node}\ s_0\ s_1)\ s_2) && \textbf{Instantiate}\\
&= \textit{join}\ (\textit{rotate}\ (\mathsf{Node}\ s_0\ s_1))\ (\textit{rotate}\ s_1) && \textbf{Unfold}\\
&= \textit{join}\ (\textit{join}\ (\textit{rotate}\ s_0)\ (\textit{rotate}\ s_1))\ (\textit{rotate}\ s_1) && \textbf{Unfold}\\
&= \textit{join}\ (\textit{rotate}\ s_0)\ (\textit{join}\ (\textit{rotate}\ s_1)\ (\textit{rotate}\ s_1)) && \textbf{Law}\\
&= \textit{join}\ (\textit{rotate}\ s_0)\ (\textit{rotate}\ (\mathsf{Node}\ s_1\ s_2)) && \textbf{Fold}\\
&= \textit{rotate}\ (\mathsf{Node}\ s_0\ (\mathsf{Node}\ s_1\ s_2)) && \textbf{Fold}\\
&= \textit{rote}\ (\mathsf{Node}\ s_0\ (\mathsf{Node}\ s_1\ s_2)). && \textbf{Fold}
\end{aligned}$$

yielding the final operation given by the system of equations:

$$\begin{aligned}
\textit{rotate}\ s &= \textit{rote}\ s\\
\textit{rote}\ (\mathsf{Leaf}\ n) &= \mathsf{Leaf}\ n\\
\textit{rote}\ (\mathsf{Node}\ (\mathsf{Leaf}\ n)\ s) &= \mathsf{Node}\ (\mathsf{Leaf}\ n)\ (\textit{rote}\ s)\\
\textit{rote}\ (\mathsf{Node}\ (\mathsf{Node}\ s_0\ s_1)\ s_2) &= \textit{rote}\ (\mathsf{Node}\ s_0\ (\mathsf{Node}\ s_1\ s_2))
\end{aligned}$$

If we call this transformation π then $[\ \pi\ ]$ is not ***fail*** and, indeed, $[\ \pi\ ]$ is reproduced as Figure 1.

§5 Conclusion and future work

In this paper we have extended the translation of transformations given in [Hen93] to those which utilise general algebraic types. This considerably complicates the translation, in particular the treatment of instantiations and folding. To show that the translation is well defined we had to undertake a transfinite induction to the ordinal ω^{ω} which was entirely absent in the original case.

The most urgent area for future work concerns the classes of operations which are available within the *fine structure* of these transformations. To obtain this fine structure we would organise the transformations according to the number of instantiations which they utilise. In the numerical case [Hen93] we were able to make a link between the Péter Hierarchy of *k*-recursions [Pét57] [Ros84] and the class of transformations requiring *k* instantiations and then, via Tiat's theorem [Tai61], we were able to place a bound on the class of ordinal recursions accessible by the full calculus of numerical transformations at $\omega^{\omega^{\omega}}$. In the current context things look more complicated but the example we gave in §4 demonstrates a 2-recursion (over *tree*) from a transformation which utilises two instantiations. However, unlike the numeric case, this initial operation is *unary* suggesting that one might find a bound at $\omega^{\omega^{\omega}}$ even for the transformations out of unary tree operations. The situation for general algebraic types is even less clear at present.

§6 Acknowledgements

I have benefited from many discussions with members of the *Constructive set theory in programming group* at the University of Essex, in particular Ray Turner and Mike Sanderson. I would also like to thank the audiences for their comments at seminars at the University of Kent and the *Applied Logic Colloquium*, University of London, where I first presented this material.

§7 References

[Bee85] Beeson, M., *Foundations of Constructive Mathematics*, Springer Verlag, 1985.

[BuD77] Burstall, R. & Darlington, J., *A transformation system for developing recursive programs*, J. ACM, 24, pp 44-67, 1977.

[Hen92] Henson, M. C., *Transformational derivation in the programming logic TK*, Proc. Int. Symp. on Computer and Information Sciences VII, (eds: Gelenbe, E., Halici, U., Yalabik, N.), Presses de l'Ecole des Hautes Etudes en Informatique, Université René Descartes, Paris, pp 65-72, 1992.

[Hen93] Henson, M. C., *An intensional semantics for elementary program transformation,* submitted, J. Logic and Computation, 1993.

[Pét57] Péter, R., *Rekursive Funktionen*, Verlag der Ungarischen Akademie der Wissenschaften, Budapest., 1957 (English translation: Academic Press, New York, 1967).

[Ros84] Rose, H. E., *Subrecursion: functions and hierarchies*, Oxford Logic Guides 9, Clarendon Press, Oxford, 1984.

[Tai61] Tait, W. W., *Nested Recursion*, Math. Ann. 143, 236-50, 1961.

[Tur79] Turner, D. A., *A new implementation technique for applicative languages*, Software Practice and Experience, 9, 31-49, 1979.

[Tur85] Turner, D. A., *Miranda: A non-strict functional language with polymorphic types*, in: Proc. IFIP Int. Conf. on functional programming languages and computer architecture, Nancy, LNCS 201, Springer Verlag, pp 445-472, 1985.

FIGURE 1

```
      $\text{Leaf } n_1 = \text{Leaf } n_1$
      $n_1 \in num$
    $\text{Leaf } n_1 = Rotate\ (\text{Leaf } n_1)$
      $n_1 \in num$
    $\text{Leaf } n_1 \in tree$
  $\exists y \in tree.y = Rotate(\text{Leaf } n_1)$
                      $\exists x \in tree.x = Rotate\ v$
                          $\text{Node (Leaf } n_2) \ w_1 = \text{Node (Leaf } n_2) \ w_1$
                              $n_2 \in num$
                            $\text{Leaf } n_2 \in tree$
                            $w_1 \in tree$
                          $\text{Node (Leaf } n_2) \ w_1 \in tree$
                        $\exists y \in tree.y = \text{Node (Leaf } n_2) \ w_1$
                        $w_1 = Rotate\ v$
                      $\exists y \in tree.y = \text{Node (Leaf } n_2) \ (Rotate\ v)$
                        $\exists y \in tree.y = \text{Node (Leaf } n_2)(Rotate\ v)$
                    $n_2 \in num$
                      $v \in tree$
                    $Rotate\ v \in tree$
                  $\exists y \in tree.y = Join\ (\text{Leaf } n_2) \ (Rotate\ v)$
                  $n_2 \in num$
                $\exists y \in tree.y = Join\ (Rotate\ (\text{Leaf } n_2)) \ (Rotate\ v)$
                  $n_2 \in num$
                $\text{Leaf } n_2 \in tree$
                $v \in tree$
              $\exists y \in tree.y = Rotate\ (\text{Node(Leaf } n_2) \ v)$
            $[\exists y \in tree.y = Rotate\ v] \Rightarrow [\exists y \in tree.y = Rotate\ (\text{Node (Leaf } n_2) \ v)]$
          $\forall s \in tree.[[\exists y \in tree.y = Rotate\ s] \Rightarrow [\exists y \in tree.y = Rotate\ (\text{Node (Leaf } n_2) \ s)]]$
                                $\forall x \in tree.[[\exists y \in tree.y = Rotate\ x] \Rightarrow [\exists y \in tree.y = Rotate\ (\text{Node } x_1 \ x)]]$
                                $x_2 \in tree$
                                $x_3 \in tree$
                              $\text{Node } x_2 \ x_3 \in tree$
                            $[\exists y \in tree.y = Rotate\ (\text{Node } x_2 \ x_3)] \Rightarrow [\exists y \in tree.y = Rotate\ (\text{Node } x_1 \ (\text{Node } x_2 \ x_3))]$
                                $\forall x \in tree.[[\exists y \in tree.y = Rotate\ x] \Rightarrow [\exists y \in tree.y = Rotate\ (\text{Node } x_2 \ x)]]$
                                $x_3 \in tree$
                              $[\exists y \in tree.y = Rotate\ x_3] \Rightarrow [\exists y \in tree.y = Rotate\ (\text{Node } x_2 \ x_3)]$
                              $\exists y \in tree.y = Rotate\ x_3$
                            $\exists y \in tree.y = Rotate\ (\text{Node } x_2 \ x_3)$
                          $\exists y \in tree.y = Rotate\ (\text{Node } x_1 \ (\text{Node } x_2 \ x_3))$
                              $w_2 = w_2$
                              $w_2 \in tree$
                            $\exists y \in tree.y = w_2$
                              $w_2 = Rotate\ (\text{Node } x_1 \ (\text{Node } x_2 \ x_3))$
                          $\exists y \in tree.y = Rotate\ (\text{Node } x_1 \ (\text{Node } x_2 \ x_3))$
                        $\exists y \in tree.y = Rotate\ (\text{Node } x_1 \ (\text{Node } x_2 \ x_3))$
                      $\exists y \in tree.y = Join\ (Rotate\ x_1) \ (Rotate(\text{Node } x_2 \ x_3))$
                    $\exists y \in tree.y = Join\ (Rotate\ x_1) \ (Join\ (Rotate\ x_2) \ (Rotate\ x_3))$
                      $x_1 \in tree$
                    $Rotate\ x_1 \in tree$
                      $x_2 \in tree$
                    $Rotate\ x_2 \in tree$
                      $x_3 \in tree$
                    $Rotate\ x_3 \in tree$
                  $\exists y \in tree.y = Join\ (Join\ (Rotate\ x_1) \ (Rotate\ x_2))(Rotate\ x_3)$
                  $x_1 \in tree$
                  $x_2 \in tree$
                $\exists y \in tree.y = Join\ (Rotate\ (\text{Node } x_1 \ x_2)) \ (Rotate\ x_3)$
                  $x_1 \in tree$
                  $x_2 \in tree$
                $\text{Node } x_1 \ x_2 \in tree$
                $x_3 \in tree$
              $\exists y \in tree.y = Rotate\ \text{node } x_1 \ x_2) \ x_3)$
            $[\exists y \in tree.y = Rotate\ x_3] \Rightarrow [\exists y \in tree.y = Rotate\ (\text{Node (Node } x_1 \ x_2) \ x_3)]$
          $\forall s \in tree.[[\exists y \in tree.y = Rotate\ s] \Rightarrow [\exists y \in tree.y = Rotate\ (\text{Node (Node } x_1 \ x_2) \ s)]]$
        $\forall x \in tree.[\forall s \in tree.[[\exists y \in tree.y = Rotate\ s] \Rightarrow [\exists y \in tree.y = Rotate\ (\text{Node } x \ s)]]]$
        $z_1 \in tree$
      $\forall x \in tree.[[\exists y \in tree.y = Rotate\ x] \Rightarrow [\exists y \in tree.y = Rotate\ (\text{Node } z_1 \ x)]]$
      $z_2 \in tree$
    $[\exists y \in tree.y = Rotatez_2] \Rightarrow [\exists y \in tree.y = Rotate\ (\text{Node } z_1 \ z_2)]$
    $\exists y \in tree.y = Rotate\ z_2$
  $\exists y \in tree.y = Rotate\ (\text{Node } z_1 \ z_2)$
$\forall x \in tree.[\exists y \in tree.y = Rotate\ x]$
```