Direttore Responsabile: Ruggero Ferro
Iscrizione al Registro Stampa del Tribunale di Padova n. 1235 del 26.9.1990

Pubblicato con il contributo di:

Cassa di Risparmio di Padova e Rovigo

Stampa Veronese (Padova)
Laser Fotocomposizioni (Padova)

FLEXIBLE AND STRICT TYPING THEORIES*

Enrico Moriconi[1] - Francesca Vitale[2]

[1] *Dipartimento di Filosofia – Università di Pisa*
[2] *Dipartimento di Informatica – Università di Pisa*

Abstract. The aim of this paper is to define a translation from (a weakened variant of) the Theory of Operations and Classes IOC_{Λ} introduced by Feferman in [3], into the Intuitionistic Type Theory of [7], enriched with the (formation and introduction rules concerning the) first two Universes U_0 and U_1.

§ 1. *Introduction.*

In [2] Beeson laid special stress on interpreting Feferman's Theory of Operations and Classes into Martin-Löf's Intuitionistic Type Theory (ITT, for short). Actually, Beeson didn't consider the full Feferman's Theory, but the subtheory — called FT — which is obtained by preventing quantification on type variables. In that paper Beeson doesn't offer technical details, but limits himself to the observation that a natural approach to the subject could consist in translating FT into the theory termed ML_1, i.e., ITT with the first Universe, U_0. In such a way U_0 would supply the range both for individual and for type variables, and the combinators of FT would be interpreted «as suitable terms built up by the operation of "abstraction"» permitted in ITT.

Here we will refer to a subtheory of Feferman's theory which is different (stronger) from that considered by Beeson, even if we will follow Beeson in using the name FT because it works suitably both for Feferman's Theory and for Flexible Typing Theory. And it is different also the basic idea that features the definition of the function which translates individual terms, classes and formulas of FT into entities of ITT, in such a way that an inhabited type of ITT corresponds to a provable formula of FT.

As it is well known, both Feferman and Martin-Löf started to develop their systems in the middle of the Seventies, and one of their (shared) aim was to provide a formalization of Bishop's flavour of constructive mathematics. When people gradually became aware of the relevance of those systems for the theoretical computer science, they immediately realized

* Ricevuto il 2 marzo 1992; accettato in forma finale il 13 luglio 1992.

their being two different answers to the basic question of typing. In a typed theory every object must belong to a well-determined type, and the theory is dubbed strictly typed (or "monomorphic") if such a type is unique. Also Feferman's theories allow the formation of types through the mechanism of abstracting with respect to a bunch of class variables. But in these theories (also termed "polymorphic") it does not hold that every object must be classified in a type, and we can provide uniform constructions across types; i.e., objects belonging to more than one type. This is what people mean when speaking of flexible types.

This common source gives reason of the great deal of work spent on inquiring about the relationship between the two kinds of theories. The most practised field of research has been till now that one having FT-like theories as target theory. We remind, for example, the embedding of ITT into the operator-part **APP** of Feferman's theories, and into the weak theory of inductive definitions ID_1^-. As regards the opposite direction, the most significat result — to our knownledge — concerns the interpretation of HA^ω + AC + EXT into ITT (consult [1] and [8] for references). This situation explains, we think, the relevance of the point emphasized by Beeson in [2], and sets the scene for the translation result we are going to prove.

In order to save space and to keep this paper within reasonable limits of size, we assume that the reader is familiar with both FT and ITT. On the other hand we feel entitled to proceed this way because a clear and sufficient background for this paper can easily be found in the relevant chapters of [1] and [8]. So we limit ourselves to the following rough remarks. (Anyway, we supply in an *Appendix* a brief description of the two systems).

In [3] Feferman introduces two theories, termed EOC_Λ and IOC_Λ (with $\mathrm{IOC}_\Lambda \subseteq \mathrm{IOC}_\Lambda$). They are respectively the elementary (or predicative) and impredicative versions of the "theory of operations and classes" he previously worked out. By FT we mean the theory resulting from IOC_Λ of Feferman [3] when type variables are allowed to range only over constant types. In other words, **we rule out abstractions over types which depend on type variables (i.e., over polymorphic types).**

The Martin-Löf's theory we shall refer to is the theory which results by adding to ITT the first two universes U_0 and U_1, plus the relative formation- and introduction-rules. This theory is usually called ML_2, but we will use the simpler name ML. In the formulation of the rules relative to the universes we shall adopt the Tarski-formulation of Martin-Löf [7] (see the *Appendix* for the relevant definitions).

Before embarking on the definition of the translation of FT into ML, we have to face two problems of a general nature, problems which require some adjustements. First, classes are formed in FT by separation from the set V of all things. ML, on the contrary, is a strictly typed theory: this brings it about that every entity belongs to a single, unique type. This also prevents the possibility that any entity belong both to its proper type *and* to another (universal) type. To deal with this problem we have to find a way to "simulate" in ML a sort of universal type. We use intentionally the verb "simulate" because what we are going to define is simply the domain of the objects on which, at every level, we will work, and not the translation of V in ML. This fact helps to put under the correct perspective the question of the preservation of meaning under the translation (whose main feature is the preservation

of provability). The Ontological Axiom of Feferman [3], which expresses that the classes are regarded as some among all the objects in the universe, makes only sense with the proviso that the translation of an FT-class must belong to a higher level domain than that which its elements belong to. In order to do that we shall introduce the types $(\Sigma y \in U_0)\, T_0(y)$ and $(\Sigma y \in U_1)\, T_1(y)$, called respectively D_0 and D_1, which we can get by the rules of ML in the following way (i=0,1):

$$\frac{U_i\ \mathrm{tp} \qquad \overset{[y \in U_i]}{T_i(y)\ \mathrm{tp}}}{(\Sigma y \in U_i)\, T_i(y)\ \mathrm{tp}}$$

In this figure 'tp' is a shorthand for 'type', and T_i is the decoding-function which, given a name (of a certain type) belonging to the universe U_i, outputs the corresponding type. Therefore, D_i is the set of pairs $<y,x>$ in which y is the name of a type in U_i, and x is an object of type $T_i(y)$. It is worth noting that D_0 is isomorphic to a subset of D_1, i.e. if $<y,x> \in D_0$ then $<\tau(y),x> \in D_1$ ('τ' being the function which carries names along universes: this allows the inference from '$a \in U_0$' to '$\tau(a) \in U_1$'). Further, let us introduce the following new (and easily seen to be sound, because perfectly in accordance with the meaning of judgements of the form "$(\Sigma y \in U_i)\, T_i(y)$ tp") inference rules:

$$\frac{<a,b> \in D_i}{b \in T_i(a)} \quad \text{and} \quad \frac{b \in A}{<\mathrm{name}(A)/\tau(\mathrm{name}(A)),b> \in D_i}$$

Moreover, we observe that U_1 contains a name for D_0: in fact we have:

$$\cfrac{\cfrac{\cfrac{\cfrac{u_0 \in U_1 \qquad \overset{[y \in T_1(u_0)]}{\tau(y) \in U_1}}{\sigma(u_0,(y)\tau(y)) \in U_1}}{T_1(\sigma(u_0,(y)\tau(y)))\ \mathrm{tp}}}{(\Sigma y \in T_1(u_0))\, T_1(\tau(y))\ \mathrm{tp}}}{(\Sigma y \in U_0)\, T_0(y)\ \mathrm{tp}}$$

and the last type obtained — whose name is $\sigma(u_0,(y)\tau(y))$ — is the very type D_0.

Thanks to the Σ-rules (that is to say, to the notion of "such that"), D_0 will make it possible to translate into ML the elementary (i.e., not depending on type variables) classes of FT. In the course of the translation, however, it comes out a previously not perceived distinction falling within the domain of elementary classes themselves. For we will have to distinguish FT-classes in the two categories of *homogeneous* and *non-homogeneous* classes. An FT-class belongs to the former category if it is such that all the objects falling within its translation belong to one and a same ML-type. Classes belonging to the latter category are instead classes for which this is not the case; i.e., their elements, if "seen" from the point of view of ML, have different types. It is to take due account of this fact that we added the two universes U_0 and U_1.

A second general problem arises from the availability of partial functions in IOC_Λ and from the consequent adoption of the Logic of Partial Terms. In ML, on the contrary, every operation is totally defined and it may be applied to an object only under the hypothesis that the object is of the suitable type. As a result of this discrepancy, the translation function will have to provide also a check for the coherence of types. Actually we shall exploit what we can call *intermediate* operations (denoted by outlined characters). They work directly on the pairs <name of a type, object of the type named>, and their task is to check the coherence of the type-assignments. If such coherence doesn't subsist, the intermediate operations will yield the pair $<\perp,c>$, where $\perp$ is, for example, $I(\mathbb{N},0,1)$: granted we will never encounter a c $\in \perp$, an operation which produces the pair $<\perp,c>$ corresponds to an "abort"-command. We observe that intermediate operations will allow us to define the translation of an FT-term in direct dependence on the translations of the terms which that FT-term is going to be applied to; i.e., directly on pairs belonging to D_i. In fact, as we will see, they are just shorthands for easily to be found ML-derivations.

We conclude this introduction recording some notational conventions we shall adopt in this paper: elements of D_i will be denoted by small greek letters; if $y \in U_i$, the type named by y, $T_i(y)$, will be denoted by Y; if "Y tp" holds, $Y^\wedge$ will denote its name in U_i (i=0,1).

§ 2. *Constant types.*

We are now ready to introduce the inductive clauses of the translation (-)* from FT into ML which constitutes the main body of the theorem we are going to prove. In order to simplify such definition, let's first consider as our source theory, FT, the subtheory of IOC_Λ which results through the omission of type variables. As a result of this restriction, **for the time being we limit our considerations to range over the set D_0.** Following the syntactical specifications of [3], § 2.2, we define (-)*-translations for individual terms, type terms, and formulas (when possible, we will be parsimonious in the use of brackets, provided that no ambiguity arises.).
Beginning from the

i) Individual terms

we remind that we intend to translate them as pairs belonging to D_0:

a) $0^* = <n,0>$ ($\in D_0$), where $T_0(n) = \mathbb{N}$ (0 is the unique individual constant of FT).

b) $a^* = \omega \in D_0$, a being an individual variable of FT.

c) If $s^*=<k,j>$ and $t^*=<r,m>$, then the pairing function P is translated as follows:

$$(P(s,t))^* = \mathbb{P}(s^*,t^*)$$
$$= \mathbb{P}(<k,j>,<r,m>) = <b,<j,m>>$$
$$\text{with } b = \sigma(k,(x)r(x)).$$

Remark. $\mathbb{P}$ is what we called an *intermediate* operation: given the pairs s* and t* as above, it stands for the following derivation in ML (which is as one expects once he is told that b must have the shape previously displayed):

$$\dfrac{\dfrac{\dfrac{\dfrac{<k,j> \in D_0}{j \in T_0(k)} \qquad \dfrac{<r,m> \in D_0}{m \in T_0(r)}}{<j,m> \in (\Sigma x \in T_0(k))T_0(r)} \qquad \dfrac{\dfrac{<k,j> \in D_0}{k \in U_0} \qquad \dfrac{<r,m> \in D_0}{r \in U_0}}{(\Sigma x \in T_0(k))T_0(r) = T_0(\sigma(k,(x)r(x))}}{<j,m> \in T_0(\sigma(k,(x)r(x)))}}{<\sigma(k,(x)r(x)),<j,m>> \in D_0}$$

Projection functions are easily described following the last definition:

d) $(P_1(t))^* = \mathbb{P}_1(t^*)$

$$= \mathbb{P}_1(<k,j>) = \begin{cases} <b,p(j)>, & \text{if } k = \sigma(b,(x)r(x)) \\ <\perp,c>, & \text{otherwise} \end{cases}$$

e) $(P_2(t))^* = \mathbb{P}_2(t^*)$

$$= \mathbb{P}_2(<k,j>) = \begin{cases} <r(p(j)),q(j)>, & \text{if } k = \sigma(b,(x)r(x)) \\ <\perp,c>, & \text{otherwise} \end{cases}$$

In this definition p and q are the projection functions of ML. We note that while the pairing operation is always defined, projections are partial operations. Here we can clearly see what the job of the intermediate operations consists in. Granted the "partial" nature of projections, we cannot give the *-translation of the term $P_i(t)$ outright. It needs a beforehand check of the coherence of the types, and this is the task of the intermediate operation $\mathbb{P}_i$.

f) $(D(s,t,v))^* = \begin{cases} R(q(s^*),t^*,(x,\omega)e(x,\omega)), & \text{if } p(s^*) = n \\ <\perp,c>, & \text{otherwise} \end{cases}$

In the translation of the Definition-by-cases-operator of FT we use the operation R introduced by the $\mathbb{N}$E-rule in ML, and denote by $(x,\omega)e(x,\omega)$ the constant function which gives v* as value. It is worth seeing the construction of the term in ML, with all its assumptions displayed:

$$\frac{q(s^*) \in \mathbb{N} \qquad t^* \in D_0 \qquad \begin{matrix}[x \in \mathbb{N}, \omega \in D_0]\\ [v^*\equiv]\ e(x,\omega) \in D_0\end{matrix}}{R(q(s^*),t^*,(x,\omega)e(x,\omega)) \in D_0}$$

g) $(st)^* = \mathbb{A}p(s^*,t^*)$

$$=\mathbb{A}p(<r,m>,<k,j>)=\begin{cases} <b,Ap(m,j)>, & \text{if } r=<\pi(a,(x)b(x))> \text{ and } k=a \\ <\bot,c>, & \text{otherwise}\end{cases}$$

We temporarily delay the treatment of $(\lambda x{:}A)t$ (and, obviously, of $(s\bullet T)$, and $(\Lambda X.t)$) as it is first necessary to define the *-translation for type-terms, abstracts, and formulas.

ii) type-terms

Since we have omitted type variables, we only have the constant class $\mathbb{N}$ of natural numbers, and the classes we can form from that one by separation. The translation rule for $\mathbb{N}$ is simply:

a) $(\mathbb{N}_{FT})^* = <u_0,n>$ ($\in D_1$), which we dub $\mathbb{N}_{ML}$, or simply $\mathbb{N}$ if there is no danger of confusion.

iii) Abstracts and formulas

Abstracts and formulas have to be treated simultaneously, for on the one hand an abstract has the shape $\{z|\phi(z)\}$, where ϕ is a formula; and the definition of the set of formulas, on the other hand, is based on atoms such as $(s=t)$ or $(t \in A)$, where A is a type-term, possibly defined by abstraction. While giving the translation rules, we will check that to every theorem of FT a non-void type of ML will correspond.

a) $(s\downarrow)^* = (\Sigma\omega \in D_0)I(D_0,\omega,s^*)$.

If the term s is defined, then the type $(\Sigma\omega \in D_0)I(D_0,\omega,s^*)$ contains the pair $<s^*,d>$, where d is the canonical proof for $s^*=s^*$, while it is void if $(s\downarrow)$ doesn't hold in FT. For in this case $s^*=<\bot,c>$, which is different from every element of D_0.

b) $(s=t)^* = I(D_0,s^*,t^*)$.

The key-rôle played by the type D_0 appears from the last clause: in fact, it is a peculiar feature of ITT that I-rules can be applied only to objects belonging to one and the same type. With D_0 available, we can write an equality as a proposition. That is to say, given any two objects we can ask about their equality by joining either object with (the name of) its type and then by asking if the resulting pairs are equal in D_0. One immediately observes that the translation $I(D_0,s^*,t^*)$ contains a canonical proof of $(s=t)^*$ iff $(s=t)$ holds in FT.

c) $(s \in A)^* = \phi_A{}^*(s^*)$, where $A=\{z|\phi_A(z)\}$.

Of course, the formula ϕ_A will ultimately consist of equations and statements of membership relations concerning $\mathbb{N}$, this being the only constant class. Consequently, for the clause c) to be well-defined, we must give explicitly the translation rule concerning $\mathbb{N}$. Accordingly, we define $(s \in \mathbb{N})^*$, i.e. $\phi_{\mathbb{N}}{}^*(s^*)$, as $I(U_0,n,p(s^*))$. Since a pair has the shape $<n,x>$ iff it is the translation of a numeral of FT, the formula $(s \in \mathbb{N})$ holds in FT iff the first component of s^* is n (we recall that $T_0(n) = \mathbb{N}$), i.e. iff $I(U_0,n,p(s^*))$ is an inhabited type. We must proceed differently in the translation of the formula $(s \in A)$ because, as we shall see later, the name of the translation A^* doesn't belong, in general, to U_0, but to U_1.

Obviously, in $(s \in A)^*$ the type $\phi_A{}^*$ is the translation in ML of the abstraction condition which occurs in the definition of A in FT, and the procedure adopted rests on the comprehension axiom of FT:

$$\forall y\, (y \in \{z|\phi[z]\} \leftrightarrow \phi[y]).$$

A term t of FT belongs to the class A iff $\phi_A(t)$ is satisfied in FT, iff — by induction hypothesis — $\phi_A{}^*(t^*)$ is an inhabited type of ML.

Having provided the *-translation for atomic formulas, the translation of other formulas doesn't present any particular difficulty:

d) $(\phi \vee \varphi)^* = (\phi^* + \varphi^*)$;

e) $(\phi \,\&\, \varphi)^* = (\phi^* \times \varphi^*)$;

f) $(\phi \to \varphi)^* = (\phi^* \to \varphi^*)$;

g) $(\neg\phi)^* = (\phi^* \to \bot)$;

h) $(\forall x\, \phi)^* = (\Pi\omega \in D_0)\, \phi^*(\omega)$;

i) $(\exists x\, \phi)^* = (\Sigma\omega \in D_0)\, \phi^*(\omega)$.

Finally, we can give the general scheme for handling the abstraction procedure of FT:

j) if $A = \{z|\phi_A(z)\}$, then $(A)^*$ is $(\Sigma\omega \in D_0)\phi_A{}^*(\omega)$,

which we will refer to simply by A^*. Roughly speaking, we can say that the translation of a class A of FT singles out the type of those pairs whose first projection belongs to D_0, and whose second projection is a canonical proof that the previous element satisfies the *-translation of the formula separating A. It is worth noting that $A^{*\wedge}$, the name of the translated class, doesn't belong to U_0, but to U_1. (We could now define $(\mathbb{N})^*$ also as $(\Sigma\omega \in D_0)I(U_0,n,p(\omega))$ or as $(\Sigma\omega \in D_1)I(U_1,\tau(n),p(\omega)))$.

<u>Examples</u> Some examples will be useful in clarifying the mechanism of the translation (cfr. § 3.1. of [3]):

a) $(V)^* = (\{x|x=x\})^* = (\Sigma\omega \in D_0)I(D_0,\omega,\omega)$;

b) $(\{a_1,a_2\})^* = (\{x|x=a_1 \vee x=a_2\})^* = (\Sigma\omega \in D_0)\, (I(D_0,\omega,(a_1)^* + I(D_0,\omega,(a_2)^*))$;

we proceed analogously if $n > 2$.

c) $(-A)^* = (\{x|x \notin A\})^* = (\Sigma\omega \in D_0)(\phi_A{}^*(\omega) \to \bot)$;

d) $(A\cup B)^* = (\{x \mid x \in A \vee x \in B\})^* = (\Sigma\omega \in D_0)(\phi_A{}^*(\omega) + \phi_B{}^*(\omega))$;

e) $(A\times B)^* = (z \mid \exists x \exists y(z=(x,y) \;\&\; x \in A \;\&\; y \in B)\})^*$

$:= (\Sigma\omega \in D_0)((\Sigma\alpha \in D_0)(\Sigma\beta \in D_0)(\phi_A{}^*(\alpha) \times \phi_B{}^*(\beta) \times I(D_0,\omega, \mathbb{P}(\alpha,\beta)))))$;

f) $(A \rightarrow B)^* = (\{z \mid \forall x(x \in A \rightarrow zx \in B)\})^* =$

$= (\Sigma\omega \in D_0)((\Pi\alpha \in D_0)(\phi_A{}^*(\alpha) \rightarrow \phi_B{}^*(\mathbb{A}\mathrm{p}(\omega,\alpha))))$;

g) $(\cap_{x \in A} T[x])^* = (\{y \mid \forall x(x \in A \rightarrow y \in T[x]\})^* =$

$= (\Sigma\omega \in D_0)((\Pi\alpha \in D_0)(\phi_A{}^*(\alpha) \rightarrow \phi^*{}_{T[\alpha]}(\omega)))$;

h) $(\Pi_{x \in A} T[x])^* = (\{z \mid \forall x(x \in A \rightarrow zx \in T[x]\})^* =$

$= (\Sigma\omega \in D_0)((\Pi\alpha \in D_0)(\phi_A{}^*(\alpha) \rightarrow \phi^*{}_{T[\alpha]}(\mathbb{A}\mathrm{p}(\omega,\alpha))))$;

§ 2.1.

Let us now tackle the problem of giving the definition for the still missing individual term of FT: $((\lambda x{:}A)t(x))^*$. With reference to this problem, we assume that A^* is $(\Sigma\omega \in D_0)\phi_A{}^*(\omega)$, with $A^{*\wedge} \in U_1$, and that $(t(x))^*$ is $\langle k,j \rangle$, or, more explicitly, $j(x) \in K(x)$ (we remind that $t(x)$ means that x may, but it need not, occur in t). We have to make some preliminary remarks. The derivation in ML of the judgement $j(x) \in K(x)$ necessarily depends on some type-statements concerning x (if x actually occurs in t). That is to say that the derivation must begin with one or two assumptions, of the form $[x \in Y]$ and $[y \in U_0]$ or just $[x \in B]$. This situation reflects the need to distinguish the case in which the λ-term which is finally obtained by discharging the assumption(s) is an uniform across types function (as, for example, the identity function) or not (as, for example, the successor function, belonging to the type $\mathbb{N} \rightarrow \mathbb{N}$). This fact means that in ML we can face the following two situations:

a)

$$\dfrac{\dfrac{\begin{array}{c}[y \in U_0,\ x \in \mathcal{T}_0(y)]\\ [\text{i.e.,}\langle y,x\rangle=\omega \in D_0\,]\\ j(x) \in K(x)\\ [\text{i.e.,}\langle k(\omega),j(\omega)\rangle \in D_0]\end{array}}{(\lambda x)j(x) \in (\Pi x \in Y)\,K(x)}}{(\lambda y)((\lambda x)j(x)) \in (\Pi y \in U_0)((\Pi x \in Y)\,K(x))}$$

b)

$$\dfrac{\begin{array}{c}[x \in B]\\ j(x) \in K(x)\end{array}}{(\lambda x)j(x) \in (\Pi x \in A)\,K(x)}$$

It is precisely the assumption(s) discharged that we have to pay attention to in order to stay in U_0 (as we know, $A^{*\wedge} \in U_1$). In fact, the assumption(s) points to the function's maximum possible definition domain, subsequently restricted to A^* by the λ-term we are going to translate. To cope with this problem we introduce a further notion, that of the Type-Set of A^*, denoted by $TS(A^*)$, and which is the set to which belong the first projections of the pairs satisfying $\phi_A{}^*$ (and hence members of D_0). Shortly (in § 2.2) we will show how to obtain a purely syntactical definition of $TS(A^*)$, resting for the completion of the argument on the previous informal explanation. We shall say that A is a *homogeneous* class if $TS(A^*)$ is a singleton; *non-homogeneous* otherwise.

When applying the Π-introduction rule we must check the Type-Set of A^*. If A is an homogeneous class (reflecting the previously displayed case b)), then $TS(A^*)$ is a singleton and we can give the translation a well-defined type. Otherwise, having to handle a derivation shaped as in a), we can only discharge the assumption $[x \in \mathcal{T}_0(y)]$, while the other assumption of derivation a), $[y \in U_0]$, still survives. In other words, what the translation amounts to in the latter case is the output of a *parametric* module. That is to say, it will be the job of the application operator $\mathbb{A}\mathrm{p}$ to give properly a type to the "generic" function before applying it to an object. As a result of this situation we get the following general definition:

$((\lambda x{:}A)\ t(x))^* =$

$$= \lambda(TS(A^*),t^*) = \begin{cases} \langle \pi(a,(x)k(x)),(\lambda x)j(x)\rangle, & \text{if } TS(A^*)=\{a\}\\ \langle \pi(y,(x)k(x)),(\lambda x)j(x)\rangle, & \text{otherwise}\end{cases}$$

The reason for using, in the latter case, the variable $y \in U_0$ is that we must wait until we have to apply the function before giving it a type.

According to the translation rule for the λ-operator, the previously given definition of the $*$-translation of an applicative term (st) (cfr. § 2.i)g)) has to be completed in the following way. Let s be $(\lambda x{:}A)\ c(x)$, and suppose the $*$-translation of s is $\langle b,(\lambda x)j(x)\rangle$. As we know, b can be either $\pi(a,(x)k(x))$, which means that A is an homogeneous class and that $TS(A^*) = \{a\}$, or $\pi(y,(x)k(x))$. The first task the intermediate operation $\mathbb{A}\mathrm{p}$ is entrusted with is properly typing s^*. In the former case s^* is a constant, and hence nothing essentially new must be added. If the latter alternative of the definition holds, it means that s is a uniform across types function. In this case, $\mathbb{A}\mathrm{p}$ must first discharge the assumption $[y \in U_0]$, producing $(\lambda y)s^*$. The second step will amount to apply $(\lambda y)s^*$ to the name corresponding to the first projection of t^*. In this way we specialize the generic function $\pi(y,(x)k(x))$, being then allowed to performe the proper application operation.

More in detail, we can say that if the latter alternative holds, then the derivation in ML of $(s)^*$, i.e., of

$$\langle \pi(y,(x)k(x)),(\lambda x)j(x)\rangle \in D_0,$$

depends on the assumption $[y \in U_0]$. Consequently, the first step manages to discharge this hypothesis through an application of the $\prod$-introduction rule. What we obtain in this way is the term:

$$(\lambda y)(\langle\pi(y,(x)k(x)),(\lambda x)j(x)\rangle) \in U_0 \to D_0$$

which then has to be applied to $p(t^*)$, say a, which is an element of U_0. In this way a type has been assigned to b, and we obtain a term $(\lambda x)j(x) \in (\prod x \in A)\ K(x)$. At this point we have to proceed in the usual way applying the last term obtained to $q(t^*)$. We can give the derivation the following compact form:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}[y \in U_0]\\ \vdots\\ (\lambda x)j(x) \in (\prod x \in Y)\ K(x)\\ \vdots\end{array}}{[\alpha(y)\equiv]\langle\pi(y,(x)k(x)),(\lambda x)j(x)\rangle \in D_0}}{(\lambda y)\alpha(y) \in U_0 \to D_0} \qquad \dfrac{t^* \in D_0}{[a\equiv]p(t^*) \in U_0}}{\mathbb{A}p\ ((\lambda y)\alpha(y),a) \in D_0}}{\langle\pi(a,(x)k(x)),(\lambda x)j(x)\rangle \in D_0}}{(\lambda x)j(x) \in (\prod x \in A)K(x)} \qquad [d\equiv]q(t^*) \in A}{\mathbb{A}p\ ((\lambda x)j(x),d) \in K(d)}}{\langle k,j\rangle \in D_0}$$

As this construction makes it clear, localizing the generic function s* to a given type must be made implicitly (that is to say, through the intermediate operation $\mathbb{A}p$). In fact, the object $(\lambda y)\alpha(y)$ doesn't belong to D_0.

Taking into due account the shape of the translation rule given for a λ-term, we can finally restate the *-translation rule for the application as follows:

g') if $s^*=\langle\pi(\beta,(x)b(x)),(\lambda x)m(x)\rangle$, and if $t^*=\langle k,j\rangle$, then

$$(st)^*= \mathbb{A}p\ (s^*,t^*)=\begin{cases}\langle b(j),m(j)\rangle, & \text{if } \beta=k \text{ (a constant value) or } \beta=y\\ \langle\bot,c\rangle, & \text{otherwise.}\end{cases}$$

§ 2.2. *The notion of Type-Set.*

Let A be a class of FT and let A* denote, as usual, $(\Sigma\omega \in D_0)\ \phi_A{}^*(\omega)$. From the previously given informal definition it follows that if TS(A*) contains a name $a \in U_0$, then at least an object $x \in \mathcal{T}_0(a)$ exists such that the type $\phi_A{}^*(\langle a,x\rangle)$ is non void. We note that the proposition $\phi_A{}^*(\omega)$ depends on the variable ω by means, ultimately, of expressions of the following kinds:

(i) $I(U_0,p(t(\omega)),a)$,

(ii) $I(D_0,t(\omega),\alpha)$

where the term $t(\omega)$ depends on ω. Obviously, the α occurring in (ii) need not be a constant: typically it may occur within a subexpression like

$$(\Sigma\alpha \in D_0)\ \phi_B{}^*(\alpha)$$

or

$$(\prod\alpha \in D_0)\ \phi_B{}^*(\alpha).$$

We will indicate with (ii) the case in which α is a constant, and with (iii) the other case. Moreover, each one of the propositions (i)-(iii) can appear in a negated (even more than once) form, like, for example:

(ii)' $I(D_0,t(\omega),\alpha) \to \bot$.

First, we observe that an easy lemma to prove is that if (i) holds, then we must have $t(\omega)\equiv\omega$ and $a\equiv n$. In fact, the restrictions we imposed on FT entail that only stratified formulas (i.e., of the form "$s \in A$") concur to the definition of an FT-class. In general, the definition of the class A can have whatever logical complexity, but it will ultimately consist of equation- and membership-expressions involving $\mathbb{N}$, the only one constant class. But if case (i) holds, then the translation of A doesn't have any logical complexity, and contains only equations between the first projection of the pair $t(\omega)$ and the name of a type. Finally, from the above and from the way we have defined the translation of the membership relation, we have that the type named must be $\mathbb{N}$ (i.e., it must be $a\equiv n$) and that the pair $t(\omega)$ must be the translation of an FT-numeral (i.e., $t(\omega)\equiv\omega$).

Given $A^*=(\Sigma\omega \in D_0)\ \phi_A{}^*(w)$, for each I-type occurring in $\phi_A{}^*(\omega)$ which actually depends on ω, we build a subset of U_0, whose definition is given by cases on the structure of the term $t(\omega)$. Let's call $IS_i(A)$, the set corresponding to the i^{th} (starting from the leftmost one) I-type taken into account. We have:

1) $t(\omega) \equiv \omega$

in case (i), we have $a \in IS_i(A)$,

in case (ii), we have $p(\alpha) \in IS_i(A)$,

in case (iii), we have $TS(B^*) \subseteq IS_i(A)$.

2.1) $t(\omega) \equiv P(\omega,\beta)$ (cfr. Def. 2.i) c))

in case (ii), if $p(\alpha)$ is, say, $\sigma(\beta,(x)c(x))$, then $b \in IS_i(A)$,

in case (iii), we have $\{b \in U_0 | d=\sigma(b,(x)c(x)), d \in TS(B^*)\} \subseteq IS_i(A)$.

2.2) $t(\omega) \equiv P(\beta,\omega)$ (...)

in case (ii), if $p(\alpha)$ is $\sigma(c,(x)b(x))$, then $b \in IS_i(A)$,

in case (iii), we have $\{b \in U_0 | d=\sigma(c,(x)b(x)), d \in TS(B^*)\} \subseteq IS_i(A)$.

We proceed analogously in the other cases.

TS(A*) can be obtained from the $IS_i(A)$'s in many ways — possibly producing a type-set larger than necessary. For simplicity's sake, we choose to define TS(A*) as the union of all the $IS_i(A)$'s, with only a further elaboration for the $IS_i(A)$'s corresponding to I-types which are negated or occur within negated types. We have to take account of two possibilities, depending on either the I-rules considered are referred to D_0 or to U_0. In the second case, the type-set is given by the usual set-theoretical operation $\setminus$. In the first one, we introduce a new operation, say $\int$, which is active only in presence of a complement symbol, as follows:

$$A \int B = A$$
$$C\setminus(A \int B) = (C\setminus A) \cup B.$$

An example will clear up what the problem consists in. Let's consider:

$$A = \{x | x=0\} \text{ and } B = \{x | x \in \mathbb{N}\}$$

Obviously, $TS(A^*)=TS(B^*)=\{n\}$ holds. If we now consider complements, we have:

$$-A = \{x | x \neq 0\} \text{ and } -B = \{x | x \notin \mathbb{N}\}$$

The type-set of (-B)* — which is obtained by making reference to U_0 — is $U_0\setminus\{n\}$. The type-set of (-A)* — defined with respect to D_0 — is $U_0 \int \{n\} = U_0$. Twice iterated negations output $\{n\}$ in the first case, and $U_0\setminus(U_0 \int \{n\}) = (U_0\setminus U_0) \cup \{n\} = \{n\}$ in the second one.

§ 3.Type variables.

The abstractions so far considered fall within the theory EOC_Λ of Feferman [3] in which every application of the comprehension axiom must concern stratified formulas which contain no bound type variables. We can drop this restriction by allowing quantification on a type variable, **provided it doesn't depend, in its turn, on a type variable** (as, for example, $(\Lambda X:(\prod Y.T(Y)).C(X))$ with T variable). As a result of this stipulation, FT comes now to denote a proper subtheory of IOC_Λ and it would be interesting to single out precisely the portion of IOC_Λ that we actually translate into ML. We will not dwell on this subject, limiting ourselves to suggest that special attention could be focused, for instance, on the relations between FT and the semi-predicative core of IOC_Λ that Feferman discusses in [3] (cfr. also Longo [5]).

Owing to the presence of type-variables, and because constant classes of FT are mapped on ML-types whose names belong to U_1, we have to make recourse to the domain D_1 (see §

1). All the translation rules given in §2 still hold — *mutatis mutandis* — also for D_1. For example, we will have:

a) $0^* = <\tau(n),0>$.

b) $x^* = <y,x>$ ($\in D_1$).

...

c) We make class variables X, Y, Z,..., range over elements of U_1.

...

Rule c) holds a separate position because the translation of a class variable does not abide by the usual way; that is to say, it is not a pair. The reason is that otherwise we would obtain an object, say $<u_1,y>$ which does not belong to D_1. Using U_1 as quantification domain, it will be possible to translate universal and existential quantifications on types from FT to ML; and thanks to the restriction imposed, abstractions can be translated making recourse only to U_0-variables. Equality betweeen classes will be expressed in ML through the I-rules of ML applied to the names (belonging to U_1) of the translated classes.

Concerning λ-abstraction and application, it is easily seen that the definitions of homogeneous and non-homogeneous class, and of Type-Set of a class, can be immediately transferred to level 1. The new terms, which are to be dealt with now, are Λ-abstraction and $\bullet$-application. We remind that the former allows abstraction on type variable, by which polymorphic classes can be produced. The latter, in its turn, has the task to localize at a given type a polymorphic function. In IOC_Λ the two terms are connected by the following axiom:

$$((\Lambda X)t[X]) \bullet Y \cong t[Y].$$

For these terms we give the following translation rules. Let t* be <r,m>, then

$$((\Lambda X)t(X))^* = <\pi(u_0,(y)r(y)),(\lambda y)m(y)>;$$

and

$$(s\bullet T)^* = \begin{cases} \mathbb{A}\mathrm{p}\ (s^*,<u_0,a>), & \text{if } TS(T^*)=\{\tau(a)\} \\ \mathbb{A}\mathrm{p}\ (s^*,<u_0,y>), & \text{for y variable, otherwise.} \end{cases}$$

The first rule is self-evident: if t(y) is an FT-term depending on a type variable, and if the pair which translates t(y) corresponds to the ML-object $m(y) \in R(y)$, then the translation of $(\Lambda X)t(X)$ outputs the function $(\lambda y)m(y)$ belonging to the type $(\prod y \in U_0)R(y)$. We emphasize that, thanks to the Type-Set notion, the translation of a Λ-term involves $\prod$-rules referred only to U_0, with the consequence that the resulting objects are elements of D_1. This would have been impossible if we had had to take into account the universe U_1.

In the second rule, the name a, or the variable y, is coupled with its type-name, u_0, because inputs for the intermediate operation $\mathbb{A}\mathrm{p}$ must be pairs belonging to D_1 (when the variable y occurs, we obtain a parametric module). What is worth emphasizing is that the second rule is well-defined iff the Type-Set TS(T*) contains names belonging to U_1 via the coding-function τ. This condition, however, is clearly satisfied, because type variables are bound to range on constant types which — as we saw in §.2 — are translatable as subtypes of D_0: consequently, names belonging to their Type-Set already occur in U_0.

We can now define:

a) $(\forall X\ \phi)^* = (\Pi x \in U_i)\ \phi^*$

b) $(\exists X\ \phi)^* = (\Sigma x \in U_i)\ \phi^*$

where i is 0 if the formula to be translated occurs within the separation condition of an abstraction, and i is 1 otherwise.

Finally, we can add a last remark to justify the choice of U_0 as appropriate range for type variables in the first case, in spite of the fact that, as we have pointed out, the name of the *-translation of a class usually belongs to U_1. Concerning this, it is enough to remind that type variables are restricted to range over constant types, which are already named in U_0, as can be easily seen. For instance, the class of even natural numbers, which, according to the general rule, is translated into a type having its name in U_1, can be expressed in U_0 by the type $(\Sigma m \in \mathbb{N})((\Sigma n \in \mathbb{N})I(\mathbb{N},m,2n))$.

§.3. Appendix

The language of FT has a bunch of individual variables a,b,c, ..., a bunch of type variables A,B,C, ..., an individual constant 0 and a type constant $\mathbb{N}$. The class of individual terms, type terms and formulas are given by means of a simultaneous inductive definition:

individual terms::= $0 \mid a,b,c, \ldots \mid st \mid s{\bullet}T \mid (\lambda x{:}S)T \mid \Lambda X.T \mid P(s,t) \mid P_i(t)\ (i=1,2) \mid D(s,t_1,t_2)$

type terms::= $\mathbb{N} \mid A,B,C,\ldots \mid \{x \mid \phi(x)\}$, for ϕ stratified

atomic formulas::= $s\!\downarrow \mid s=t \mid s \in T$

formulas::= atomic formulas $\mid \neg\,\phi \mid \phi\ \&\ \varphi \mid \phi \vee \varphi \mid \phi \to \varphi \mid \forall x\ \phi \mid \exists x\ \phi \mid \forall X\ \phi \mid \exists X\ \phi$.

Leaving aside axioms and rules relating to the Logic of Partial Terms, we record the axioms of IOC_Λ, which are arranged in the following I-V classes:

I. Abstraction-Reduction

(i) $y \in X \to ((\lambda x{:}X)t[x])y \cong t[y]$

(ii) $(\Lambda X.t[X]) \bullet Y \cong t[Y]$

where $(s \cong t)$ is an abbreviation for $[(s\!\downarrow \vee\ t\!\downarrow) \to s=t]$

II. Pairing, Projections

(i) $P(x,y) \neq 0$

(ii) $P_1(x,y)=x\ \&\ P_2(x,y)=y$

III. Comprehension Axiom

$\forall y\,(y \in \{z \mid \phi[z]\} \leftrightarrow \phi[y])$, for each ϕ stratified

IV. Natural Numbers

(i) $0 \in \mathbb{N}\ \&\ \forall x(x \in \mathbb{N} \to succ(x) \in \mathbb{N})$

(ii) $0 \in X\ \&\ \forall x(x \in X \to succ(x) \in X) \to \mathbb{N} \subseteq X$

V. Definition by Cases on $\mathbb{N}$

(i) $x=0 \to D(x,y_1,y_2)=y_1$

(ii) $x \in \mathbb{N}\ \&\ x\neq 0 \to D(x,y_1,y_2)=y_2$

The axioms of EOC_Λ are the same except that in the Comprehension Axiom the formula ϕ is required to be elementary stratified, which means that the formula contains no bound type variables.

The features of the theory ML.which are mainly relevant for this paper can be restricted to those involving the following two forms of judgements:

"A tp" (A is a type or A is a proposition)

and

"$a \in A$" (a is an element of the type A or a is a proof of the proposition A).

In fact, the body of ML consists in a set of operations which can be performed on types according to the following rules (which, at the same time, give the meaning of the two forms of judgements:

(i) the Cartesian Product Π of a family of types

Π-formation

$$\begin{array}{c}[x \in A]\\ \dfrac{A\ tp \qquad B(x)\ tp}{(\Pi x \in A)B(x)\ tp}\end{array}$$

Π-introduction

$$\begin{array}{c}[x \in A]\\ \dfrac{b(x) \in B(x)}{(\lambda x)b(x) \in (\Pi x \in A)B(x)}\end{array}$$

Π-elimination

$$\dfrac{c \in (\Pi x \in A)B(x) \qquad a \in A}{Ap(c,a) \in B(a)}$$

Π-equality

$$\begin{array}{c}\qquad\qquad\qquad [x \in A]\\ \dfrac{a \in A \qquad\qquad b(x) \in B(x)}{Ap((\lambda x)b(x),a)=b(a) \in B(a)}\end{array}$$

(ii) The Disjoint Union Σ of a family of types

Σ-formation

$$\begin{array}{c}[x \in A]\\ \dfrac{A\ tp \qquad B(x)\ tp}{(\Sigma x \in A)B(x)\ tp}\end{array}$$

Σ-introduction

$$\dfrac{a \in A \qquad b \in B(a)}{\langle a,b\rangle \in (\Sigma x \in A)B(x)}$$

Σ-elimination

$$\begin{array}{c}[x\in A, y\in B]\\[4pt] \dfrac{c\in(\Sigma x\in A)B(x) \quad d(x,y)\in C(<x,y>)}{E(c,(x,y)d(x,y))\in C(c)}\end{array}$$

Σ-equality

$$\begin{array}{c}[x\in A, y\in B]\\[4pt] \dfrac{a\in A \quad b\in B(a) \quad d(x,y)\in C(<x,y>)}{E(<a,b>,(x,y)d(x,y))=d(a,b)\in C(<xa,b>)}\end{array}$$

By putting

$$p(c)=E(c,(x,y)x) \qquad \text{and} \qquad q(c)=E(c,(x,y)y)$$

Σ-elimination gives the usual left and right projections.

Besides other important interpretations, the type $(\Sigma x\in A)B(x)$ has the meaning of the set of all a in A *such that* B(a) holds. That is to say that Σ-rules play the role of the usual Comprehension Axiom.

(iii) The Disjoint Union + of two types

+-formation

$$\frac{A\ tp \quad B\ tp}{A+B\ tp}$$

+-introduction

$$\frac{a\in A}{i(a)\in A+B} \qquad \frac{b\in B}{j(b)\in A+B}$$

+-elimination

$$\begin{array}{c}[x\in A] \qquad\qquad [y\in B]\\[4pt] \dfrac{z\in A+B \quad d(x)\in C(i(x)) \quad e(y)\in C(j(y))}{(\mathbf{D}\ x,y)(z,d(x),e(y))\in C(z)}\end{array}$$

+-equality

$$\begin{array}{c}[x\in A] \qquad\qquad [y\in B]\\[4pt] \dfrac{a\in A \quad d(x)\in C(i(x)) \quad e(y)\in C(j(y))}{(\mathbf{D}\ x,y)(i(a),d(x),e(y))=d(a)\in C(i(a))}\end{array} \qquad \begin{array}{c}[x\in A] \qquad\qquad [y\in B]\\[4pt] \dfrac{b\in B \quad d(x)\in C(i(x)) \quad e(y)\in C(j(y))}{(\mathbf{D}\ x,y)(j(b),d(x),e(y))=e(b)\in C(j(b))}\end{array}$$

(iv) Identity

I-formation

$$\frac{A\ tp \quad a\in A \quad b\in A}{I(A,a,b)\ tp}$$

I-introduction

$$\frac{a=b\in A}{r\in I(A,a,b)}$$

I-elimination

$$\frac{c\in I(A,a,b)}{a=b\in A}$$

I-equality

$$\frac{c\in I(A,a,b)}{c=r\in I(A,a,b)}$$

where the judgement "a=b∈ A" means that a and b are equal objects (proofs) of the type (proposition) A; and r is a canonical object (proof) of the type (proposition) I(A,a,b).

(v) Natural Numbers

N-formation

$$\mathbb{N}\ tp$$

N-introduction

$$0\in\mathbb{N} \qquad \frac{a\in\mathbb{N}}{succ(a)\in\mathbb{N}}$$

N-elimination

$$\begin{array}{c}[x\in\mathbb{N}, y\in C(x)]\\[4pt] \dfrac{c\in\mathbb{N} \quad d\in C(0) \quad e(x,y)\in C(succ(x))}{\mathbf{Rec}(c,d,(x,y)e(x,y))\in C(c)}\end{array}$$

N-equality

$$\begin{array}{c}[x\in\mathbb{N}, y\in C(x)]\\[4pt] \dfrac{d\in C(0) \quad e(x,y)\in C(succ(x))}{\mathbf{Rec}(0,d,(x,y)e(x,y))=d\in C(0)}\end{array}$$

and

$$\begin{array}{c}[x\in\mathbb{N}, y\in C(x)]\\[4pt] \dfrac{a\in\mathbb{N} \quad d\in C(0) \quad e(x,y)\in C(succ(x))}{\mathbf{Rec}(succ(a),d,(x,y)e(x,y))=e(a,\mathbf{Rec}(a,d,(x,y)e(x,y)))\in C(succ(a))}\end{array}$$

(vi) Universes

Following the Tarski-formulation, we introduce an Universe U_0 whose elements are indices or names of types. Canonical elements of U_0 are constructed by means of new operations (denoted by lower case greek letters) which reflect on names the operations Π,Σ,... on types. We limit ourselves to few examples.

U_0-formation

$$U_0\ tp \qquad\qquad \frac{a\in U_0}{T_0(a)\ tp}$$

U_0-introduction

$$\frac{a \in U_0 \quad \overset{[x \in T_0(a)]}{b(x) \in U_0}}{\pi(a,(x)b(x)) \in U_0} \qquad \frac{a \in U_0 \quad \overset{[x \in T_0(a)]}{b(x) \in U_0}}{T_0(\pi(a,(x)b(x))) = (\prod x \in T_0(a))T_0(b(x))}$$

$$\frac{a \in U_0 \quad \overset{[x \in T_0(a)]}{b(x) \in U_0}}{\sigma(a,(x)b(x)) \in U_0} \qquad \frac{a \in U_0 \quad \overset{[x \in T_0(a)]}{b(x) \in U_0}}{T_0(\sigma(a,(x)b(x))) = (\sum x \in T_0(a))T_0(b(x))}$$

The process can be iterated, introducing a new Universe U_1 (and in fact any U_n for n natural numbers; but for our purposes the first two Universes are enough) with similar formation and introduction rules plus the following new introduction rules:

$$u_0 \in U_1 \qquad T_1(u_0) = U_0$$

$$\frac{a \in U_0}{\tau(a) \in U_1} \qquad \frac{a \in U_0}{T_1(\tau(a)) = T_0(a)}.$$

REFERENCES

[1] M. Beeson, *Foundations of Constructive Mathematics*, Springer, Berlin, 1985.

[2] — , *Proving Programs and Programming Proofs*, in R. Marcus et al. (eds.) VII Int. Cong. for Logic Methodology and Philosophy of Science, North-Holland, Amsterdam, 1988, 51-82.

[3] S. Feferman, *Polymorphic typed lambda-calculus in a type-free axiomatic framework*, in W. Sieg (ed.), Logic and computation, Am. Math. Soc., 1990, 101-136.

[4] G. Longo, *From type-structures to Type Theories*. Notes for a graduate course at Carnegie Mellow University, 1988.

[5] — , *Some Aspects of Impredicativity*. Notes on Weyl's Philosophy of Mathematics and Today's Type Theory, in H.D. Ebbinghaus et al. (eds.), Logic Colloquium '87, North-Holland, Amsterdam, 1989, 241-274.

[6] P. Martin-Löf, *Constructive Mathematics and Computer Programming*, in L.J.Cohen et al. (eds.), VI Int. Cong. for Logic Methodology and Philosophy of Science, North-Holland, Amsterdam, 1982, 153-175.

[7] — , *Intuitionistic Theory of Types*, Bibliopolis, Napoli, 1984.

[8] A.S. Troelstra, D. van Dalen, *Constructivism in Mathematics*, North Holland, Amsterdam, 1988.