Werkexemplaar

Elimination of Σ-types by means of introspective types

Maarten M Fokkinga, 8 febr 1988

We extend the (term and) type formation rules of the (higher order) typed λ-calculi with the rule:

> $Rx{:}M.\,N$
> $(Rx.M)$ is a type, whenever x is a term variable and N, M ~~is a~~ are types.

This type is called the introspective type, because x stands for the term itself to which this type is assigned. Consequently, one of the type inference rules reads: from $N{:}(Rx.M)$ conclude $N{:}\,M[x{:}{=}N]$.

We show that the strong Σ-types (with the two projections π_1 and π_2) can be translated into Π-types plus the introspective type; (in this translation there is no need for Type:Type).

1. Introduction

There is a strong interest in typed λ-calculi for at least two reasons. On the one hand there is the proposition-as-type interpretation that leads to consider suitable typed λ-calculi as a formalism for constructive mathematics, cf. [Coquand & Huet 1985], [Martin-Löf 1975], [de Bruijn 1980]. On the other hand typed λ-calculi form a model of modern and future programming languages, with facilities like abstract data types and modules, cf.[†] [Fokkinga 1983], [Fokkinga 1987], [Burstall & Lampson 1984], [Cardelli 1986]. Quite essential in both approaches is the presence of dependent types; like $(\Pi x{:}A.\ B)$ for the type of functions that map x in A into elements of B (where B may depend on x).

The dependent type $\Sigma x{:}A.B$ —where B may depend on x— is the type of pairs $\langle a,b\rangle$ with a in A and b in $B[x:=a]$. From the logical side, this type is strongly related to $\exists x{:}A.B$ (just as $\Pi x{:}A.B$ is related to $\forall x{:}A.B$). From the programming side, $\Sigma x{:}A.B$ is related to abstract types, cf. [Fokkinga 1987]. Given a pair $\langle a,b\rangle$ of type $\Sigma x{:}A.B$, the projections $\pi_1\langle a,b\rangle$ and $\pi_2\langle a,b\rangle$ yield a respectively b. It seems impossible to express Σ, π_1, π_2 and $\langle\rangle$ as terms not using $\Sigma, \pi_1, \pi_2, \langle\rangle$.

---

† [Reynolds 1974]

In order to do the translation of $\Sigma, \pi_1, \pi_2, \langle\rangle$ we introduce another type formation rule:

$Rx{:}A.B$ $\qquad$ types A, B

$Rx.A$ is a type, for any variable x and type A.

This type is called the <u>introspective</u> type, because x stands for the term to which the type is assigned. Consequently one of the type inference rules reads

$(Rx{:}A \cdot B)$

from $N{:}(Rx \cdot A)$ conclude $N{:}\,A[x:=N]$.

It turns out, and will be shown below, that $\Sigma, \pi_1, \pi_2, \langle\rangle$ can be expressed (without $\Sigma, \pi_1, \pi_2, \langle\rangle$) using only Π- and introspective types.

We have invented introspective types solely for this purpose. It remains to be seen whether they are of any further use, and what desirable properties are lost by adding introspective types to a typed λ-calculus.

This note is a formalisation of [Fokkinga 1984] in which introspective types were presented originally.

2. The typed λ-calculus

We shall keep the typing system as conservative as possible: we refrain from stipulating $*:*$ (where $*$ is interpreted as the type of types), because we can do with the simpler $*:**$ (and now $*$ is the type of types, but $*$ itself is typed with $**$). We shall first define the set of terms, and then give the type inference rules.

Postulate a countably infinite set of so-called variables. We let x, y, z vary over variables.

Terms. Terms are inductively defined as follows.

$*$, $**$ are terms;

let x be a variable and M, N be terms, then

x, $(\lambda x.M)$, $(\Pi x{:}M.\,N)$, $(M\,N)$, $\langle M,N\rangle$, $(\Sigma x{:}M.\,N)$, $\pi_1 M$, $\pi_2 M$

are terms.

We let M, N, A, B vary over terms; in particular A, B vary over terms for which we have assumed or can infer that $A:*$ and $B:*$ (see below).

The notions of free variable FV and substitution $:=$ are defined in the usual way; in this respect $\Pi x{:}M.N$, $\Sigma x{:}M.N$ (and $Rx{:}M.N$) behave exactly like $\lambda x.N$.

One might write $(A \to B)$ for $(\Pi x{:}A.\,B)$ whenever $x \notin FV(B)$, but we shall not do so.

Notice that in $(\lambda x.M)$ there is no type indicated for x; ~~that~~ a term may have several types and we shall use ~~that!~~ this property!

Type inference. Here we list the rules for inferring formulae $M:N$ in the style of natural deduction.

(*) $\quad * : **$

(Π*) $\quad \dfrac{A:\square \qquad \begin{matrix}[x:A]\\ B:*\end{matrix}}{(\Pi x:A.\ B):*}$ $\qquad$ (Π**) $\quad \dfrac{A:\square \qquad \begin{matrix}[x:A]\\ B:**\end{matrix}}{(\Pi x:A.\ B):**}$ $\qquad \square \in \{*, **\}$

(λ) $\quad \dfrac{A:\square \qquad \begin{matrix}[x:A]\\ M:B\end{matrix}}{(\lambda x.\ M):(\Pi x:A.\ B)}$ $\qquad \square \in \{*, **\}$

provided that in the right hand side subderivation $\gamma, x:A \vdash M:B$ there are no assumptions $y:C$ with $x \in FV(C)$

(@) $\quad \dfrac{M:(\Pi x:A.\ B) \qquad N:A}{(M\,N):\ B[x:=N]}$

(:β) $\quad \dfrac{M:A \qquad A =_\beta A' \qquad A':\square}{M:\ A'}$ $\qquad \square \in \{*, **\}$

(Σ*) $\quad \dfrac{A:\square \qquad \begin{matrix}[x:A]\\ B:*\end{matrix}}{(\Sigma x:A.\ B):*}$ $\qquad \square \in \{*, **\}$

(⟨⟩) $\quad \dfrac{M:A \qquad N:B[x:=M]}{\langle M,N\rangle:(\Sigma x:A.\ B)}$

(π_1) $\quad \dfrac{M:(\Sigma x:A.B)}{\pi_1 M:\ A}$ $\qquad$ (π_2) $\quad \dfrac{M:(\Sigma x:A.\ B)}{\pi_2 M:\ B[x:=\pi_1 M]}$

We shall not consider a rule $(\Sigma{**})$ that is related to $(\Sigma *)$ as $(\Pi{**})$ is related to $(\Pi *)$; it wouldn't presumably present ~~no~~ any problems. As an alternative to the duplications of the rules (one for $\square = *$ and one for $\square = **$; also one $(\Pi *)$ and one $(\Pi{**})$), one might think of adding the rule

$$(**)\quad \frac{A : *}{A : **}$$

and then having $**$ everywhere $*$ occurs now (except in rule $(*)$, $(**)$). ? $\Rightarrow (\Sigma{**})$ $\Rightarrow (R{**})$ ~~erasing all rules in which a formula "M:*" occurs, (~~ ~~replacing "$\square \in \{*, **\}$" everywhere by "$\square \in \{**\}$" (and deleting rule $(\Pi *)$, $(\Sigma *)$)~~. We shall not investigate this idea. The interested reader may consult [Coquand 1986, Section 6.1, 6.2].

Extension with introspective types

We add to the term formation rules

$(Rx{:}A.\ B)$ is a term, for variable x and terms A, B,

and to the type inference rules

$$(R*)\quad \frac{A : \square \qquad \begin{matrix}[x : A] \\ B : *\end{matrix}}{(Rx{:}A.\ B) : *} \qquad \square \in \{*, **\}$$

$$(R\text{-intro})\quad \frac{A : * \quad \begin{matrix}[x{:}A] \\ B : *\end{matrix} \qquad M : A \qquad M : B[x := M]}{M : (Rx : A.\ B)}$$

$$(R\text{-elim})\quad \frac{M : (Rx : A.\ B)}{M : B[x := M]}$$

We have given introspective types the same shape as Π- and Σ-types: the type of the bound variable is explicitly indicated. Presumably this is not at all necessary; we could have defined $(Rx.B)$ to be a term rather than $(Rx:A.B)$; however, we shall use x within B in no other way than suggested by type A.

We conjecture, moreover, that no properties of the type inference system are lost if the three left most premisses of rule (R-intro) are omitted: we have added them for savety's sake. In this way it <u>seems</u> clear that, for example, M strongly normalizes because $M:A$ already guarantees that.

3 Expressing Sums in other terms

We construct terms Sum, Pair, Fst, Snd that behave exactly like $\Sigma, \langle\rangle, \pi_1, \pi_2$. Formally, let a and b be variables, and let Γ be the assumption list

$$a:*, \quad b:(\Pi x:a.*)$$

then

$$\Gamma \vdash \mathrm{Sum} : *$$
$$\Gamma \vdash \mathrm{Pair} : \Pi x:a.\ \Pi y:bx.\ \mathrm{Sum}$$
$$\Gamma \vdash \mathrm{Fst} : \Pi z:\mathrm{Sum}.\ a$$
$$\Gamma \vdash \mathrm{Snd} : \Pi z:\mathrm{Sum}.\ b\,(\mathrm{Fst}\ z)$$

and

$$\left.\begin{array}{l} \mathrm{Fst}\ (\mathrm{Pair}\ M\ N) =_\beta M \\ \mathrm{Fst}\ (\mathrm{Pair}\ M\ N) =_\beta N \end{array}\right\} \text{ provided } \Gamma \vdash M:a,\ N: b\,M$$

The proof is given in the next section; here we confine ourselves to the intuitive, informal development.

As an aid to the reader we list here our convention as regards to naming:

a, b, x, y, z, f, g are variables;

$a:*$;

$b:(\Pi x:a.*)$, so that $b x:*$ if $x:a$;

$x:a$;

$y:bx$;

$F \equiv \Pi x:a.*$ will be the type of f, $f:F$;

$G \equiv \Pi x:a, y:bx.\ fx$ will be a type of g, $g:G$;

$G' \equiv \Pi x:a, y:bx.\ A$ will be a type of g, $g:G'$;

$z:\mathrm{Sum}$.

We would like to construct the terms Pair, Fst, Snd and Sum by adapting and typing the conventional coding of pairs. So our first attempt is

$$
\begin{aligned}
\text{Pair} &\approx \lambda x\, y.\ \lambda g.\ \ g\, x\, y \\
&:\ \Pi x{:}a\ \ y{:}bx.\ \ \Pi g{:}(\Pi x{:}a\ \ y{:}bx.\ ??).\ \ \cancel{\text{Sum}}\ ?? \\
\text{Fst} &\approx \lambda z.\ \ z\ (\lambda x\, y.\ x) \\
&:\ \Pi z{:}\text{Sum}.\ \ a \\
\text{Snd} &\approx \lambda z.\ \ z\ (\lambda x\, y.\ y) \\
&:\ \Pi z{:}\text{Sum}.\ \ b\ (\text{Fst}\ z) \\
\text{Sum} &\equiv\ ???
\end{aligned}
$$

The first problem is the result type of g: ?? has to be a when g equals $(\lambda x\, y.\, x)$, and ?? has to be ~~by~~ bx when g equals $(\lambda x\, y.\, y)$. To express this type uniformly, we add another parameter, f, that forecasts this type: when g equals $(\lambda x\, y.\, x)$ take f to be $(\lambda x.a)$, and when g equals $(\lambda x\, y.\, y)$ take f to be $(\lambda x.\, bx)$. In both cases $F \equiv \Pi x{:}a.\ast$ is the type of f, so that f itself doesnot introduce further problems. We have (with $G \equiv \Pi x{:}a\ \ y{:}bx.\ fx$),

$$
\begin{aligned}
\text{Pair} &\approx \lambda x\ y.\ \ \lambda f\ g.\ \ g\ x\ y \\
&:\ \Pi x{:}a\ \ y{:}bx.\ \ \underbrace{\Pi f{:}F\ \ g{:}G.\ \ \cancel{\text{Sum}} \to f x}_{\approx\ \text{Sum}} \\
\text{Fst} &\approx \lambda z.\ \ z\ (\lambda x.a)\ (\lambda x\, y.\, x) \\
&:\ \Pi z{:}\text{Sum}.\ \ a \\
\text{Snd} &\approx \lambda z.\ \ z\ (\lambda x.\ bx)\ (\lambda x\, y.\, y) \\
&:\ \Pi z{:}\text{Sum}.\ \ b\ (\text{Fst}\ z) \\
\text{Sum} &\equiv\ ???
\end{aligned}
$$

The next problem is to find a suitable term for Sum, that can be used in both Pair and Fst, Snd. Within Pair we might take Sum $\equiv$ $(\Pi f{:}F\ g{:}G.\ f\ x)$ as the type of $P \equiv (\lambda f\ g.\ g\ x\ y)$, but then x occurs free in Sum, so Sum cannot be used outside Pair. However we know that x is to be (Fst P), so using introspective types we can eliminate x:

$$\text{Sum} \equiv \ \mathrm{R}p{:}\,??\,.\ \Pi f{:}F\ g{:}G.\ f\ (\text{Fst}\ p)$$

We are left with the problem of designing a type for p. (Even if the format of introspective types is $(\mathrm{R}p.\,N)$ rather than $(\mathrm{R}p{:}M.\ N)$ we still have to find M in order to infer a typing for Sum.) Clearly we cannot use Sum as a type for p, because the term Sum would then have a circular definition – it would be an infinite term. It happens that we can take

$$\Pi f{:}F\ \ g{:}G'.\ a \qquad \text{where } G' \equiv \Pi x{:}a\ \ y{:}b x.\ a$$

as the type for p within Sum;† (Notice that ~~Fst indeed~~ the occurrence of Fst within Sum is not assigned the type $(\Pi z{:}\text{Sum}.\ a)$, but instead $\Pi p{:}(\Pi f{:}F\ g{:}G'.\ a).\ a$. This is a legal type for Fst indeed.)

---

† it is also a legal type for P within Pair.

Summing up, we have found

$$\begin{aligned}
\mathit{Sum} &\equiv R p{:}(\Pi f{:}F\ g{:}G'.\ a).\ \Pi f{:}F\ g{:}G.\ f\ (\mathit{Fst}\ p)\\
\mathit{Pair} &\equiv \lambda x\ y.\ \lambda f\ g.\ g\ x\ y\\
\mathit{Fst} &\equiv \lambda z.\ z\ (\lambda x.\ a)\ (\lambda x\ y.\ x)\\
\mathit{Snd} &\equiv \lambda z.\ z\ (\lambda x.\ b x)\ (\lambda x\ y.\ y)
\end{aligned}$$

By abstracting from a and b we even get

$\lambda a\ b.\ \mathit{Sum}$

$\lambda a\ b.\ \mathit{Pair}$

$\lambda a\ b.\ \mathit{Fst}$

$\lambda a\ b.\ \mathit{Snd}$

Using these, it is easy to give a "syntactic homomorphic", "convertibility-isomorphic" "λ-calculus transformation" that eliminates Σ, $\langle\rangle$, π_1 and π_2 in exchange for $\Pi, \lambda, @, R$. (Cf. [Fokkinga 1987 b] for the notions of "syntactic homomorphic", "convertibility-isomorphic", "λ-calculus transformation".)

4. Proofs of the correct behaviour of Sum, Pair, Fst, Snd

1. Thm Fst (Pair M N) $\twoheadrightarrow_\beta$ M, Snd (Pair M N) $\twoheadrightarrow_\beta$ N.

Proof Easy.

In the sequel we shall frequently use some of the following assumptions and abbreviations.

$a : *$	$F \equiv \Pi x{:}a.\,*$
$b : (\Pi x{:}a.\,*)$	$G \equiv \Pi x{:}a\ y{:}bx.\ fx$
$x : a$	$G' \equiv \Pi x{:}a\ y{:}bx.\ a$
$y : bx$	
$f : F$	$P \equiv \lambda f\,g.\ g\,x\,y$

A list of assumptions will be abbreviated by the left-hand side variables; e.g. $abxy$, ab and so on.

2. Lemma There exist derivations

D1: $ab \vdash F : **$

D2: $ab \vdash G' : *$

D3: $abf \vdash G : *$

D4: $ab \vdash \lambda x.a : F$

D5: $ab \vdash \lambda x\,y.\,x : G[f := \lambda x.a]$

D6: $ab \vdash \lambda x\,y.\,x : G'[f := \lambda x.a] \equiv G'$

D7: $ab \vdash \lambda x.\,bx : F$

D8: $ab \vdash \lambda x\,y.\,y : G[f := \lambda x.\,bx]$

D9: $abxy \vdash (\Pi f{:}F\ g{:}G.\ fx) : *$

D10: $ab \vdash (\Pi f{:}F\ g{:}G'.\ a) : *$

Proof. Straightforward. Notice that the only place where (Π**) is used, is D1 (and the places where D1 itself is invoked). Further, (Π* with $\square$ = **) is used in both D9 and D10.

3. Lemma $\quad ab \vdash Fst : (\Pi p : (\Pi f{:}F\ g{:}G'.\ a).\ a)$

Proof

$$
\begin{prooftree}
\AxiomC{$D10$}
\AxiomC{$[s : (\Pi f{:}F\ g{:}G'.\ a)]^{1}$}
\AxiomC{$D4$}
\RightLabel{@}
\BinaryInfC{$s\,(\lambda x.a) : (\Pi g{:}G'.\ a)$}
\AxiomC{$D6$}
\RightLabel{@}
\BinaryInfC{$s\,(\lambda x.a)\,(\lambda x\,y.\,y) : a$}
\RightLabel{$\lambda\,[s]^{1}$}
\BinaryInfC{$\lambda s.\ s\,(\lambda x.a)\,(\lambda x\,y.\,y) : \Pi s : (\Pi f{:}F\ g{:}G'.\ a).\ a$}
\end{prooftree}
$$

4. Theorem $\quad ab \vdash Sum : *$

Proof.

$$
\begin{prooftree}
\AxiomC{$D10$}
\AxiomC{$D1$}
\AxiomC{$[f]^{2}$}
\noLine
\UnaryInfC{$D3$}
\AxiomC{$[f : (\Pi x{:}a.\,*)]^{2}$}
\AxiomC{Lemma 3}
\dashedLine
\UnaryInfC{$Fst : (\Pi p : (\Pi f{:}F\ g{:}G'.\ a).\ a)$}
\AxiomC{$[p : (\Pi f{:}F\ g{:}G'.\ a)]^{3}$}
\RightLabel{@}
\BinaryInfC{$(Fst\ p) : a$}
\BinaryInfC{$f\,(Fst\ p) : *$}
\RightLabel{$\Pi *\,[g]^{1}$}
\BinaryInfC{$(\Pi g{:}G.\ f\,(Fst\ p)) : *$}
\RightLabel{$\Pi *\,[f]^{2}$}
\BinaryInfC{$(\Pi f{:}F\ g{:}G.\ f\,(Fst\ p)) : *$}
\RightLabel{$R *\,[p]^{3}$}
\BinaryInfC{$Sum : *$}
\end{prooftree}
$$

5. Lemma $\quad abxyf \vdash f\,(\mathrm{Fst}\ P) : *$

Proof

$$[g : G']^1 \qquad x : a$$
$$\overline{g\,x : (\Pi y{:}bx.\,a) \qquad y : bx}\ @$$
$$\overline{\quad g\,x\,y : a \quad}\ @$$
$$D2 \qquad g\,x\,y : a$$
$$\overline{(\lambda g.\ g\,x\,y) : (\Pi g{:}G'.\ a)}\ \lambda[g]^1$$
$$D_1 \qquad (\lambda g.\ g\,x\,y) : (\Pi g{:}G'.\ a)$$
$$\overline{P \equiv (\lambda f\ g.\ g\,x\,y) : (\Pi f{:}F\ g{:}G'.\ a)}\ \lambda[f]^2$$
$$\mathrm{Lemma\,3} \qquad P \equiv (\lambda f\ g.\ g\,x\,y) : (\Pi f{:}F\ g{:}G'.\ a)$$
$$\overline{(\mathrm{Fst}\ P) : a}\ @$$
$$f : F \qquad (\mathrm{Fst}\ P) : a$$
$$\overline{f\,(\mathrm{Fst}\ P) : *}\ @$$

6. Theorem $\quad ab \vdash \mathrm{Pair} : \ \Pi x{:}a\ y{:}bx.\ \mathrm{Sum}$

Proof

$$[g : G]^1 \qquad [x : a]^4$$
$$\overline{g\,x : \Pi y{:}bx.\,f x \qquad [y : bx]^3}$$
$$\overline{\quad g\,x\,y : f x \quad}$$
$$g\,x\,y : f x \qquad f x =_\beta f\,(\mathrm{Fst}\ P) \quad \underset{\mathrm{Lemma\,5}}{[x, y, f]}$$
$$\overline{g\,x\,y : f\,(\mathrm{Fst}\ P)}$$
$$\overset{[f]^2}{D3} \qquad g\,x\,y : f\,(\mathrm{Fst}\ P)$$
$$\overline{(\lambda g.\ g\,x\,y) : \Pi g{:}G.\ f\,(\mathrm{Fst}\ P)}\ \lambda[g]^1$$
$$D1 \qquad (\lambda g.\ g\,x\,y) : \Pi g{:}G.\ f\,(\mathrm{Fst}\ P)$$
$$\overline{P \equiv (\lambda f\ g.\ x y) : \Pi f{:}F\ g{:}G.\ f\,(\mathrm{Fst}\ P)}\ \lambda[f]^2$$
$$D, D', D'' \qquad P \equiv (\lambda f\ g.\ x y) : \Pi f{:}F\ g{:}G.\ f\,(\mathrm{Fst}\ P)$$
$$\overline{P : \mathrm{Sum}}\ \mathrm{Rintro}$$

$$\dfrac{b : \Pi x{:}a.*\quad [y : a]^4}{bx : *} \qquad P : \mathrm{Sum}$$
$$\overline{(\lambda y.\,P) : (\Pi y{:}bx.\ \mathrm{Sum})}\ \lambda[y]^3$$
$$a : * \qquad (\lambda y.\,P) : (\Pi y{:}bx.\ \mathrm{Sum})$$
$$\overline{(\lambda x\ y.\,P) : (\Pi x{:}a\ y{:}bx.\ \mathrm{Sum})}\ \lambda[x]^4$$

where D'': $ab \vdash P : (\Pi f{:}F\ g{:}G'.\ a)$ is shown within Lemma 5

and D: $ab \vdash (\Pi f{:}F\ g{:}G'.\ a) : *$ and D': $abp \vdash (\Pi f{:}F\ g{:}G'.\ f\,(\mathrm{Fst}\ p)) : *$

are already contained in $ab \vdash \mathrm{Sum} : *$ (Theorem 4).

7. Theorem $\quad ab \vdash \mathrm{Fst} : \Pi z{:}\mathrm{Sum}.\, a$

Proof

$$
\begin{array}{l}
\qquad [z : \mathrm{Sum}]^1 \\
\hline \text{(Relim)} \\
z : \Pi f{:}F\ g{:}G.\ f\,(\mathrm{Fst}\ z) \qquad \underline{D4} \\
\hline @ \\
z\,(\lambda x.a) : (\Pi g{:}G.\ f\,(\mathrm{Fst}\ z))[f := \lambda x.a] \qquad \underline{D5} \\
\hline @ \\
z\,(\lambda x.a)\,(\lambda x y.x) : f\,(\mathrm{Fst}\ z)[f := \lambda x.a]\,D \quad =_\beta \quad a \qquad a{:}* \\
\hline (:\beta) \\
\mathrm{Thm4} \qquad z\,(\lambda x.a)\,(\lambda x y.x) : a \\
\hline \lambda[z]^1 \\
\lambda z.\ z\,(\lambda x.a)\,(\lambda x y.x) : (\Pi z{:}\mathrm{Sum}.\ a)
\end{array}
$$

8. Theorem $\quad ab \vdash \mathrm{Snd} : \Pi z{:}\mathrm{Sum}.\, b\,(\mathrm{Fst}\ z)$

Proof

$$
\begin{array}{l}
\qquad [z : \mathrm{Sum}]^1 \\
\hline \text{Relim} \\
z : \Pi f{:}F\ g{:}G.\ f\,(\mathrm{Fst}\ z) \qquad D7 \\
\hline @ \\
z\,(\lambda x.bx) : (\Pi g{:}G.\ f\,(\mathrm{Fst}\ z))[f := \lambda x.bx] \qquad \underline{D8} \\
\hline @ \\
z\,(\lambda x.bx)\,(\lambda x y.y) : (f\,(\mathrm{Fst}\ z))[f := \lambda x.bx] \quad =_\beta \quad b\,(\mathrm{Fst}\ z) \qquad \overset{[z]}{D} \\
\hline (:\beta) \\
\mathrm{Thm4} \qquad z\,(\lambda x.bx)\,(\lambda x y.y) : b\,(\mathrm{Fst}\ z) \\
\hline \lambda[z]^1 \\
\lambda z.\ z\,(\lambda x.bx)\,(\lambda x y.y) : \Pi z{:}\mathrm{Sum}.\ b\,(\mathrm{Fst}\ z)
\end{array}
$$

where D: $\quad abz \vdash b\,(\mathrm{Fst}\ z) : *\quad$ is as follows:

$$
\begin{array}{c}
\qquad\qquad \underline{\mathrm{Thm7}} \qquad z{:}\mathrm{Sum} \\
\qquad\qquad \overline{\qquad\qquad\qquad\qquad} @ \\
b : \Pi x{:}a.* \qquad\qquad \mathrm{Fst}\ z : a \\
\hline @ \\
b\,(\mathrm{Fst}\ z) : *
\end{array}
$$

References

de Bruijn, N.G., A survey of the project Automath. In J.P. Seldin & J.R. Hindley (eds): To H.B. Curry – Essays on Combinatory Logic, Lambda Calculus and Formalism. Academic Press, London, 1980

Cardelli, L. & Wegner, P., On understanding types, data abstraction and polymorphism. Comp. Surveys 17 (1985) 471-522

Burstall, R. & Lampson, B., A Kernal language for abstract data types and modules. In: (eds) G. Kahn, D.B. MacQueen, G. Plotkin: Semantics of Data Types. LNCS 173 (1984) 1-50.

Coquand, Th & Huet, G: Constructions – a higher order proof system for mechanizing mathematics. EUROCAL 85, LNCS 203 (1985) 151-184.

Coquand, Th: An analysis of Girard's Paradox. Techn. Report No. 531, INRIA, 1986.

Fokkinga, M.M.: Programming Language Concepts – The Lambda Calculus Approach. In: (eds) P.R.J. Asveld & A. Nijholt: Essays on concepts, formalisms, and tools. CWI-Tract 42, 1987.

Fokkinga, M.M.: Modellering van lijsten in 2de orde getypte λ-calculi. Manuscript, dec 1987.

Martin-Löf, P.: An intuitionistic theory of types: predicative part. In: Logic Colloquium 1973, pp 73-118. North-Holland, 1975.

Reynolds, J.C.: Towards a theory of type structure. LNCS 19 (1974) pp 408-425.

Fokkinga, M.M.: Nog een uitbreiding vd SVP'-typering. Manuscript, 1984.