Set theory and nominalisation, part 2

Eindhoven University of Technology

Department of Mathematics and Computing Science

Set Theory and Nominalisation
Part II

by

Fairouz Kamareddine

92/13

Computing Science Note 92/13
Eindhoven, July 1992

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. F. van Neerven
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

Set Theory and Nominalisation, Part II

Fairouz Kamareddine
Department of Mathematics and Computing Science
Eindhoven University of Technology
Den Dolech 2, postbus 513
5600 MB Eindhoven, the Netherlands
email: fairouz@info.win.tue.nl
tel: +31 40 474319

July 16, 1992

Abstract

In this paper we shall meet the application of Scott domains to nominalisation and explain its problem of predication. We claim that it is not possible to find a solution to such a problem within semantic domains without logic. Frege structures are more conclusive than a solution to domain equations and can be used as models for nominalisation. Hence we develop a type theory based on Frege structures and use it as a theory of nominalisation.

Keywords: *Frege structures, Nominalisation, Logic and Type freeness.*

1 Frege structures, a formal introduction

Having in part I informally introduced Frege structures, I shall here fill in all the technical details and show that Frege structures exist.

Consider $\mathbf{F_0}$, $\mathbf{F_1}, \ldots$, a family $\mathbf{F}$ of collections where $\mathbf{F_0}$ is a collection of objects, and $(\forall n > 0)[\mathbf{F_n}$ is a collection of n-ary functions from $\mathbf{F_0^n}$ to $\mathbf{F_0}]$.

Definition 1.1 *(An explicitly closed family) A family* $\mathbf{F}$ *as above is explicitly closed iff: For every expression* $e[x_1, \ldots, x_n]$ *of the metalanguage built up in the usual way from variables ranging over* $\mathbf{F_0}$ *and constants ranging over* $\cup_n \mathbf{F_n}$*, the n-place function denoted by* $< e[x_1, \ldots, x_n]/x_1, \ldots, x_n >$ *is in* $\mathbf{F_n}$*. More formally,* $\mathbf{F}$ *is explicitly closed iff 1, 2 and 3 below hold:*

1. Closure under constant functions: *For each* a *in* $\mathbf{F_0}$*, the function* f_a *is in* $\mathbf{F_1}$*, where* $(\forall x)[f_a(x) = a]$.
2. Closure under composition: *For each* f *in* $\mathbf{F_m}$*, for each* $g_1, \ldots, g_m$ *in* $\mathbf{F_k}$*,* $f(g_1, \ldots, g_m)$ *is in* $\mathbf{F_k}$ *where* $(f(g_1, \ldots, g_m))(x_1, \ldots, x_k) = f(g_1(x_1, \ldots, x_k), \ldots, g_m(x_1, \ldots, x_k))$.
3. Closure under projection: *For each* $n, i \geq 1, P_i^n$ *is in* $\mathbf{F_n}$ *where* $P_i^n(a_1, \ldots, a_n) = a_i$ *for each* a_i *in* $\mathbf{F_0}$ *and,* $1 \leq i \leq n$.

For example, if f and g are unary functions of $\mathbf{F}$ and h is a binary function of $\mathbf{F}$, then $< f \circ g(h(x_1, x_2))/x_1, x_2 >$ is a 2-ary function (i.e. in $\mathbf{F_2}$).

In what follows, we assume such a closed family and call it $\mathbf{F}$.

Definition 1.2 *($\mathbf{F}$-functional) A function* $D : \mathbf{F_{n_1}} \times \ldots \times \mathbf{F_{n_k}} \longrightarrow \mathbf{F_0}$ *is an* $\mathbf{F}$*-functional with respect to the explicitly closed family* $\mathbf{F}$*, iff:* $(\forall m \geq 0)(\forall f_1$ *in* $\mathbf{F_{m+n_1}}) \ldots (\forall f_k$ *in* $\mathbf{F_{m+n_k}})$ $[< D(< f_1(\bar{y}, \bar{x_1})/\bar{x_1} >, \ldots, < f_k(\bar{y}, \bar{x_k})/\bar{x_k} >)/\bar{y} >$ *is in* $\mathbf{F_m}]$

where $\bar{y}$ *is a list of* m*-variables and* $\bar{x_i}$ *is a list of* n_i *variables, for* $i = 1, \ldots, k$.

Note that if $f_1, .., f_k$ are 1-place functions and $D : \mathbf{F_1} \times \ldots \times \mathbf{F_1} \longrightarrow \mathbf{F_0}$ then $D(f_1, \ldots, f_k)$ is in $\mathbf{F_0}$. What is the intuitive meaning of $\mathbf{F}$-functionals? We know that an $\mathbf{F}$-functional is a functional, so that it operates on functions. But once we include functionals in the structure, we need to ensure that any expression which contains functionals should actually be in the structure. Assume for the sake of argument that $D : \mathbf{F_{n_1}} \times \ldots \times \mathbf{F_{n_k}} \longrightarrow \mathbf{F_0}$ is an $\mathbf{F}$-functional. Assume also that for some $m \geq 0$, f_i is in $\mathbf{F_{m+n_i}}$ for $i = 1, \ldots, k$. We know that according to the explicit closure, if $\bar{y}$ is a list of m-variables ranging over $\mathbf{F_0}$ and for each $i, \bar{x}_i$ is a list of n_i variables ranging over $\mathbf{F_0}$, then $< f_i(\bar{y}, \bar{x}_i)/\bar{x}_i >$ is an element of $\mathbf{F_{n_i}}$ for each i. Therefore it makes sense to talk of the expression $D(< f_1(\bar{y}, \bar{x}_1)/\bar{x}_1 >, \ldots, < f_k(\bar{y}, \bar{x}_k)/\bar{x}_k >)$. This expression however is open in $\bar{y}$ and if we abstract over $\bar{y}$ in this expression are we going to obtain an element of $\mathbf{F_m}$? Nothing so far in the structure ensures that this is the case, and we must therefore impose the constraint that these functionals should have such a property. A functional which has this property is called an $\mathbf{F}$-functional and now if D is an $\mathbf{F}$-functional then $[< D(< f_1(\bar{y}, \bar{x}_1)/\bar{x}_1 >, \ldots, < f_k(\bar{y}, \bar{x}_k)/\bar{x}_k >)/\bar{y} >$ is in $\mathbf{F_m}]$. Hence we extend the definition of explicit closure to the following:

Definition 1.3 *(A super explicitly closed family) Taking a family as above, we say that this family is super explicitly closed iff for every expression* $e[\xi_1, \ldots, \xi_m]$ *of the metalanguage,*

built up in the usual way from variables ranging over $\cup_n \mathbf{F_n}$ *and constants ranging over* $\cup_n \mathbf{F_n}$ *and over* **F***-functionals, the m-place function denoted by* $< e[\xi_1, \ldots, \xi_m]/xi_1, \ldots \xi_m >$ *is an* **F***-functional.*

This notion of explicit closure is going to provide us with the full comprehension principle we have been promising.

Theorem 1.4 *Any explicitly closed family which has variables for functions and objects, constants for objects, functions and* **F***-functionals, is a super explicitly closed family. (The proof is by an easy induction.)*

Example 1.5 *As an example of an explicitly closed family, consider* P_ω *as described previously. Define* $\mathbf{F_0}$ *to be the set of all subsets of* ω *(i.e.* P_ω*). Define, for each* $n \geq 0, \mathbf{F_n}$ *to be the set of all continuous functions from* $\mathbf{F_0^n} \longrightarrow \mathbf{F_0}$*. Using Part I, it can be easily seen that the constant functions, the projection functions, etc are continuous. It can also be seen that continuity is closed under composition and that any combination* $e[x_1, \ldots, x_n]$ *of variables for objects and constants for both functions and objects results in the function denoted by* $< e[x_1, \ldots, x_n]/x_1, \ldots, x_n >$ *being an element of* $\mathbf{F_n}$*. Therefore the family* $(\mathbf{F_n})_n$ *just obtained from* P_ω *(call it FE), is an explicitly closed family. Furthermore, FE is super explicitly closed as it can be proven not only that* $< e[x_1, \ldots, x_n]/x_1, \ldots, x_n >$ *denotes a continuous function but also that for any expression* $e[\xi_1, \ldots, \xi_n]$ *built in the usual way out of variables ranging over* $\cup_n \mathbf{F_n}$ *and constants ranging over both* $\cup_n \mathbf{F_n}$ *and* **F***-functionals,* $< e[\xi_1, \ldots, \xi_n]/\xi_1, \ldots, \xi_n >$ *denotes a continuous function.*

So far, we have only explicit closure on our structure. But that is not enough to give a logic on the structure. In what follows, we see how to obtain such a logic.

Assume an explicitly closed family **F** and a list of logical constants which are the following **F**-functionals:

$\neg : \mathbf{F_0} \longrightarrow \mathbf{F_0}$
$\vee, \wedge, \rightarrow, \doteq : \mathbf{F_0} \times \mathbf{F_0} \longrightarrow \mathbf{F_0}$
$\forall, \exists : \mathbf{F_1} \longrightarrow \mathbf{F_0}$

Definition 1.6 *(Logical system) A logical system on a super explicitly closed family* **F***, relative to a set of logical constants as above, is* $<$ **PROP**, **TRUTH** $>$ *the set of two collections of objects such that* **TRUTH** $\subseteq$ **PROP***. These two collections are closed under an adopted logical schemata for each logical constant. The logical schemata corresponds to the external logic and tells us, for each logical constant from the list, how to build new propositions out of other ones using the logical constant. It also gives the conditions of truth for the resulting proposition.*

THE LOGICAL SCHEMATA

- *NEGATION* If a is in **PROP** then $\neg a$ is in **PROP** and $\neg a$ is in **TRUTH** iff a is not in **TRUTH**.
- *CONJUNCTION* If a, b are in **PROP** then $(a \wedge b)$ is in **PROP** and $(a \wedge b)$ is in **TRUTH** iff a is in **TRUTH** and b is in **TRUTH**.
- *DISJUNCTION* If a, b are in **PROP** then $(a \vee b)$ is in **PROP** and $(a \vee b)$ is in **TRUTH** iff a is in **TRUTH** or b is in **TRUTH**.

- *IMPLICATION* If a is in **PROP** and b is in **PROP** provided that a is in **TRUTH** then $(a \rightarrow b)$ is in **PROP** and $(a \rightarrow b)$ is in **TRUTH** iff a is in **TRUTH** implies b is in **TRUTH**.
- *UNIVERSAL QUANTIFICATION* If f is a propositional function in $\mathbf{F_1}$ then $\forall f$[1] is in **PROP** and $\forall f$ is in **TRUTH** iff $f(a)$ is in **TRUTH** for all objects a.
- *EXISTENTIAL QUANTIFICATION* If f is a propositional function in $\mathbf{F_1}$ then $\exists f$ is in **PROP** and $\exists f$ is in **TRUTH** iff $f(a)$ is in **TRUTH** for some object a.
- *EQUALITY* If a, b are objects then $(a \doteq b)$ is in **PROP** and $(a \doteq b)$ is in **TRUTH** iff $a = b$[2].
- *BI-IMPLICATION* If a, b are in **PROP** then $(a \equiv b)$ is in **PROP** and $(a \equiv b)$ is in **TRUTH** iff (a is in **TRUTH** iff b is in **TRUTH**).

From now on, we shall use *a is true* for *a is in* **TRUTH**, *a is a proposition* for *a is in* **PROP** and *a is a set* for *a is in* **SET**. In short, a logical system builds a logic on our structure. But something is still missing: even, though we built the logical system on the top of an explicitly closed structure, where functional abstraction $< e[x_1, \ldots, x_n]/x_1, \ldots, x_n >$ and application $f(x)$ do exist, we still need a way of turning functions into objects (via λ) and of applying such objects to other objects (via **app**) so that $\mathbf{app}(\lambda f, x) = f(x)$. We do not want to gain logic yet lose the bijection between objects and functions. Therefore, our structure must have more in it. The next definition will tell us what.

Definition 1.7 *(λ-system) A λ-system on an explicitly closed family* $\mathbf{F}$ *is a pair of functionals* $< \lambda, \mathbf{app} >$ *such that:* $\lambda : \mathbf{F_1} \longrightarrow \mathbf{F_0}$ *and* $\mathbf{app} : \mathbf{F_0} \times \mathbf{F_0} \longrightarrow \mathbf{F_0}$ *satisfy:* $\mathbf{app}(\lambda x f[x], a) = f(a)$, *for each* f *in* $\mathbf{F_1}$ *and* a *in* $\mathbf{F_0}$.

Example 1.8 *If we take the system FE of Example 1.5, and if we define* $\lambda : \mathbf{F_1} \longrightarrow \mathbf{F_0}$ *as* $\lambda f = \{(n, m) : m \text{ is in } f(e_n)\}$ *where we take* (n, m) *to be* $1/2(n + m)(n + m + 1) + m$ *and define* $\mathbf{app} : \mathbf{F_0} \times \mathbf{F_0} \longrightarrow \mathbf{F_0}$ *as* $\mathbf{app}(a, b) = \{m : e_n \subseteq b \text{ for some } n, (n, m) \text{ is in } a\}$;
then $(\lambda, \mathbf{app})$ *forms a λ-system for FE.*

Proof: $\mathbf{app}(\lambda f, a) = \{m : e_n \subseteq a \text{ for some } n \text{ and } (n, m) \text{ is in } \lambda f\} =$
$\{m : e_n \subseteq a \text{ for some } n \text{ and } m \text{ is in } f(e_n)\} =$
$\{m \text{ in } f(e_n) : e_n \subseteq a\} = f(a)$ *by continuity.*

Therefore $(\lambda, \mathbf{app})$ *is a λ-system for FE. Actually, FE contains λ and* **app** *and so it is a λ-structure, but we leave this to the next definition.*

[1] Recall that $\forall, \exists$ and λ are functions from $\mathbf{F_1}$ to $\mathbf{F_0}$ and hence f does not necessarily contain any free variables. For example, $< x/x >$ is the identity function and contains no free variables, $\lambda < x/x >$ can be written as $\lambda x.x$. The same holds for $\forall < x/x >$ and $\exists < x/x >$. This might be confusing, as it might be asked if $< x/x >$ has no free variables, then what does $\lambda < x/x >$ mean? Despite the fact that $< x/x >$ has no free variables (let us denote $< x/x >$ by I), it is still an element of $\mathbf{F_1}$. I.e. it is still a function and we need to make it an object by nominalizing it. Therefore we turn it into an element of $\mathbf{F_0}$ by using λ. Now λI is in $\mathbf{F_0}$ and $\mathbf{app}(\lambda I, a) = I(a) = a$. In fact, λ does not abstract on free variables, it is $< / >$ which does so. λ just turns a function into an object preserving the comprehension axiom: $\mathbf{app}(\lambda f, x) = f(x)$. When we say $\lambda x.x$, we don't mean that λ abstracts over x in x, rather we mean that we first abstract via $< / >$ obtaining $< x/x >$ and then we look for the nominal of $< x/x >$

[2] Note the two equal signs, $\doteq$ and $=$. The first is a functional from $\mathbf{F_0} \times \mathbf{F_0}$ to $\mathbf{F_0}$ such that $a \doteq b$ is always a proposition and $a \doteq b$ is true iff we can prove from the rules of λ-calculus with logic that we are formulating that $a = b$. For example, we know from above that $\mathbf{app}(\lambda f, x) = f(x)$, hence the proposition $\mathbf{app}(\lambda f, x) \doteq f(x)$ is true.

Definition 1.9 *(λ-structure) A λ-structure is an explicitly closed family* **F** *which has a λ-system.*

Note that the λ-structure contains λ and **app** and that it is an explicitly closed family.

Example 1.10 *Now take the λ-system FE given in Example 1.8. FE is also a λ-structure having* $(\lambda, \mathbf{app})$ *as λ-system, because both λ and* **app** *are in FE, as FE is explicitly closed.*

Definition 1.11 *(Frege structures) A Frege structure is a logical system relative to a list of logical constants on an explicitly closed family* **F**, *together with a λ-system.*

Example 1.12 *As an example of a Frege structure, take the λ-structure FE given in Example 1.10 and which has a λ-system* $(\lambda, \mathbf{app})$. *Aczel (in [Aczel 1985]) showed that each λ-structure can be extended to a Frege structure. Therefore we now have an example of a Frege structure.*

Let us sketch the proof of how our particular λ-structure FE can be extended to a Frege structure. This will make the reader understand the notion of Frege structure, and get him used to working with it. Before proceeding, however, we must define two missing notions: that of an *independent* family of **F**-functionals and of a *primitive* **F**-functional.

Definition 1.13 *We say that a family of* **F***-functionals is* independent *iff for any two* **F***-functionals in the family, the range of values of those* **F***-functionals are disjoint.*

This implies that if F and G belong to an independent family of **F**-functionals, then for any $\bar{f}$ and $\bar{g}$ such that $F(\bar{f}) = G(\bar{g})$, we should definitely have $F = G$. From independence only we cannot conclude that $\bar{f} = \bar{g}$. For this we need primitivity and this is the next notion we define.

Definition 1.14 *We say that an* **F***-functional* $F : \mathbf{F_{n_1}} \times \ldots \times \mathbf{F_{n_k}} \longrightarrow \mathbf{F_0}$ *is* primitive *iff there exists a projection* P_i *in* $\mathbf{F_{n_i+1}}$ *for each* $1 \leq i \leq k$ *such that* $P_i(F(\bar{f}), \hat{a}) = f_i(\hat{a})$ *where* $\bar{f} = f_1, \ldots, f_k$ *is in* $\mathbf{F_{n_1}} \times \ldots \times \mathbf{F_{n_k}}$ *and* $\hat{a}$ *is in* $\mathbf{F_0^i}$.

The aim of primitive **F**-functionals is similar to injectivity; if we have $F(\bar{f}) = F(\bar{g})$ then we should be able to deduce $\bar{f} = \bar{g}$. It can be easily checked from the definition of **F**-primitiveness that this is the case.

The proof that we can extend any λ-structure into a Frege structure is based on two theorems. The first is one which asserts the existence of an independent family of primitive **F**-functionals on the λ-structure, which include the logical constants, $\wedge, \vee$ etc. It simply states that:

Theorem 1.15 *If for each natural number m we let* $(v_{m_1}, \ldots, v_{m_k})$ *be a finite sequence of natural numbers, then there is an independent family of primitive* **F***-functionals:*
$F_m : F_{v_{m_1}} \times \ldots \times F_{v_{m_k}} \longrightarrow \mathbf{F_0}$, *for* $m = 0, 1, 2, \ldots$.

The second is the well known fixed point theorem which applies to monotonic operators and helps us to find the logical schema of these logical constants. This theorem simply states the following:

Theorem 1.16 *If A is a partially ordered collection of objects such that every chain in A has a least upper bound then any monotonic operator Y from A to A has a fixed point. That is* $(\exists a \in A)[Y(a) = a]$.

Let us apply those two theorems to our FE and obtain out of it a Frege structure. Up to here, we know that the λ-structure FE exists and Theorem 1.15 enables us to find all the logical constants needed. What remains to turn it into a Frege structure is to find a logical system for the logical constants. This is the task of Theorem 1.16. The idea is to associate with each logical constant two predicates which will ultimately (after we get to the fixed point) give all the propositions obtained from the logical constant and all the truths respectively. The construction is well known mathematically and is similar to the one followed by Kripke in [Kripke 1963]. Now consider our λ-structure FE. We can be sure from Theorem 1.15 that we have a list of **F**-functionals which includes:

$\neg : \mathbf{F_0} \longrightarrow \mathbf{F_0}$

$\vee, \wedge, \rightarrow, \equiv, \doteq : \mathbf{F_0} \times \mathbf{F_0} \longrightarrow \mathbf{F_0}$

$\forall, \exists : \mathbf{F_1} \longrightarrow \mathbf{F_0}$

But we still need to make sure that they satisfy the closure properties we want to impose on them. I shall here try to make the construction a little easier than that described by Aczel (in [Aczel 1985]). To construct a logical schema for each constant, i.e. to define the whole logical system, we follow Aczel's intended construction but will carry an example with us at all times. The logical system is defined inductively. As the basis of the induction, we start with a pair $\chi_0 = (\chi_{0p}, \chi_{0t})$ such that $\chi_{0t} \subseteq \chi_{0p}$. Intuitively, χ_{0p} is the set of propositions at stage 0 and χ_{0t} is the set of truths at stage 0.

Example 1.17 *Let* $\chi_0 = (\chi_{0p}, \chi_{0t}) = (\{0,1\}, \{1\})$. *Note that both* $\{0,1\}$ *and* $\{1\}$ *are in* P_ω.

Before proceeding to the induction step, we must define a couple of auxiliary predicates which ensure that the logical constants map their arguments into appropriate values. That is, for each logical constant F, there is one predicate Φ_F which tests whether a particular tuple of arguments has the correct status of propositionhood, and a second predicate Ψ_F which states the conditions under which the tuple will be mapped into **TRUTH** by F. To see why we need this, recall the logical schema for negation that we presented under *NEGATION* above:

- (1) If a is in **PROP** then $\neg a$ is in **PROP**, and $\neg a$ is in **TRUTH** iff a is not in **TRUTH**.

This is an instance of a general logical schema for those functionals F in a Frege structure which correspond to truth-functional connectives:

- (2) If $\bar{f}$ is in $\mathbf{F_{n_1}} \times \ldots \times \mathbf{F_{n_k}}$ and $C'(F, \bar{f})$, then $F(\bar{f})$ is in **PROP**; and $F(\bar{f})$ is in **TRUTH** iff $C(F, \bar{f})$, where C expresses F's truth conditions and C' expresses F's propositionhood.

Now it is Φ_F which tests that the arguments $\bar{f}$ are in **PROP**, while Ψ_F does the work of C in (2).

Example 1.18 $\Phi_\neg$ *and* $\Psi_\neg$ *take arguments in* $(\cup\chi_i) \times \mathbf{F_0}$ *and*

$\Phi_\neg(\chi_0, x)$ *is:* x is in χ_{0p}

$\Psi_\neg(\chi_0, x)$ *is:* x is in not χ_{0t}

Thus, $\Phi_\neg(\chi_0, x)$ *is true of the set* $\chi_{0p} = \{0,1\}$, *and* $\Psi_\neg(\chi_0, x)$ *is true of all elements in* $\mathbf{F_0} \setminus \chi_{0t}$, *i.e. everything except the element 1.*

In order to carry out the induction step of the construction, we introduce a principle which determines how the propositions and truths at stage $i+1$ are built from the propositions and truths at stage i. The principle has two parts as follows:

Principle 1.19 *χ_{i+1p} is the collection of those $F(\bar{f})$ where F is a logical constant and $\Phi_F(\chi_i, \bar{f})$.*

Principle 1.20 *χ_{i+1t} is the collection of those $F(\bar{f})$ where F is a logical constant and both $\Psi_F(\chi_i, \bar{f})$ and $\Phi_F(\chi_i, \bar{f})$.*

In other words, given the pair (χ_{ip}, χ_{it}), we construct $(\chi_{i+1p}, \chi_{i+1t})$ in the following way:

first, χ_{i+1p} has to contain all and only those elements $F(\bar{f})$ such that $\bar{f}$ belongs to the propositions at stage i, i.e. it is in χ_{ip} according to $\Phi_F(\chi_i, \bar{f})$; and second, χ_{i+1t} must contain all and only those elements $F(\bar{f})$ such that $\bar{f}$ belongs to both the propositions and the truths at stage i, i.e. it is in χ_{ip} and χ_{it} according to $\Phi_F(\chi_i, \bar{f})$ and $\Psi_F(\chi_i, \bar{f})$. Notice that the principle guarantees that $\chi_{(i+1)t} \subseteq \chi_{(i+1)p}$.

Example 1.21 *We wish to build $\chi_1 = (\chi_{1p}, \chi_{1t})$ from $(\chi_{0p}, \chi_{0t}) = (\{0,1\}, \{1\})$. By Principle 1.19, χ_{1p} is the set of objects $\neg x$ such that $\Phi_\neg(\chi_0, x)$, i.e. it is the set $\{\neg 0, \neg 1\}$. By Principle 1.20, χ_{1t} is the set of objects $\neg x$ such that $\Phi_\neg(\chi_0, x)$ and $\Psi_\neg(\chi_0, x)$, i.e. such that x belongs to χ_{0p} but does not belong to χ_{0t}. The only thing which satisfies both these conditions is 0, so $\chi_{1t} = \{\neg 0\}$.*

Example 1.22 *$\Phi_\wedge$ and $\Psi_\wedge$ take arguments in $(\cup\chi_i) \times (\mathbf{F_0} \times \mathbf{F_0})$ and*
$\Phi_\wedge(\chi_0, (x,y))$ is: x and y are in χ_{0p}
$\Psi_\wedge(\chi_0, (x,y))$ is: x and y are in χ_{0t}

Thus, we can supplement the χ_{1p} of the previous example with the set of objects $\wedge(x,y)$ such that $(x,y) \subseteq \chi_{0p} \times \chi_{0p}$, i.e. the set $\{0 \wedge 0, 0 \wedge 1, 1 \wedge 0, \ldots\}$. Similarly, we add to χ_{1t} the set of objects $\wedge(x,y)$ such that $(x,y) \subseteq \chi_{0p} \times \chi_{0t}$, i.e. the set $\{1 \wedge 1\}$. Note that according to our example, the collection of objects in **TRUTH** at stage 1 is $\{1 \wedge 1, \neg 0\}$.

Note also that $\neg 0, 1 \wedge 1, 1 \vee 0$ are distinct objects, even though they are all in **TRUTH** and all have the same truth value in Frege's terms. If we wish, we could reconstruct Frege's notion of the True and the False by forming the relevant equivalence classes, but Frege structures give us an intensional ontology. This is justified on the grounds that objects with the same truth value, e.g. $\neg 0$ and $1 \wedge 1$ are equivalent in truth value but distinct.

We see that the pair is being enlarged at each step starting from the first step where we take $\chi_{0p} = \{0,1\}$ and $\chi_{0t} = \{1\}$, with the property that for each i we have: $\chi_{it} \subseteq \chi_{ip}$. Note that we are not imposing the condition that $\chi_{it} \subseteq \chi_{(i+1)t}$ or $\chi_{ip} \subseteq \chi_{(i+1)p}$; in fact our construction is monotonic in another sense which we shall see below. The aim is now to keep going up to a certain level α where $\chi_\alpha = (\chi_{\alpha p}, \chi_{\alpha t})$ is a logical system, because it is obvious that χ_i at the levels we met so far are not logical systems. Take for example χ_0 in our example above based on FE. Then χ_0 is not a logical system, as can be seen by taking the logical schema for $\neg$:

If a is a proposition then $\neg a$ is a proposition such that $\neg a$ is true iff $\neg a$ is not true.

χ_0 is not a logical system because 1 is in χ_{0p} (supposed to represent propositions) but $\neg 1$ is not in χ_{0p}. Nor is χ_1 a logical system because $\neg 1$ is in χ_{1p} but $\neg\neg 1$ is not in χ_{1p} and so on. To solve this problem, let us consider the fixed point (if it exists) of this construction. It

may be that the fixed point is a logical system and if so, we have succeeded. Before we prove that the fixed point is a logical system, let us remind ourselves again of the construction. The construction is built through an operator Y which takes us from level i to level $i+1$ in such a way that $Y(\chi_i) = \chi_{i+1}$, where $\chi_i = (\chi_{ip}, \chi_{it})$, $\chi_{i+1} = (\chi_{i+1p}, \chi_{i+1t})$, $\chi_{it} \subseteq \chi_{ip}$, $\chi_{i+1t} \subseteq \chi_{i+1p}$. Moreover χ_{i+1p} and χ_{i+1t} are obtained as follows:

For any **F**-functional F, χ_{i+1p} is the collection of those $F(\bar{f})$ where F is a logical constant and $\Phi_F(\chi_i, \bar{f})$ and χ_{i+1t} is the collection of those objects $F(\bar{f})$ where F is a logical constant and both $\Phi_F(\chi_i, \bar{f})$ and $\Psi_F(\chi_i, \bar{f})$. Now we prove that any χ such that $\chi = Y(\chi)$ is a logical system. To show that, we have to prove that for each logical constant F, the logical schemata of F holds in χ. Let F be a logical constant whose logical schema is as follows:

If $\bar{f}$ is in $\mathbf{F_{n_1}} \times \ldots \times \mathbf{F_{n_k}}$ and $\Phi_F(\chi, \bar{f})$, then $F(\bar{f})$ is in χ_p; and $F(\bar{f})$ is in χ_t iff $\Psi_F(\chi, \bar{f})$.

Let us prove that this schema holds in χ where χ is a fixed point, $\chi = (\chi_p, \chi_t)$ and $Y(\chi) = (\chi_p, \chi_t)$. Let $\bar{f}$ be in $\mathbf{F_{n_1}} \times \ldots \times \mathbf{F_n}$ where $\Phi_F(\chi, \bar{f})$. As $\Phi_F(\chi, \bar{f})$ then $F(\bar{f})$ is in χ'_p by definition, but $\chi'_p = \chi_p$ (because $\chi = Y(\chi)$), therefore $F(\bar{f})$ is in χ_p. Now let us prove that $F(\bar{f})$ is in χ_t iff $\Psi_F(\chi, \bar{f})$.

- ($\Longrightarrow$) If $F(\bar{f})$ is in χ'_t then $F(\bar{f})$ is in χ'_t. As $F(\bar{f})$ is in χ'_t then there exists an **F**-functional G and a sequence $\bar{g}$ in $\mathbf{F_{n_1}} \times \ldots \times \mathbf{F_{n_k}}$ such that $F(\bar{g}) = G(\bar{g})$ and $\Phi_G(\chi, \bar{g})$ and $\Psi_G(\chi, \bar{g})$ by definition. But the logical constants are independent. Therefore $F = G$ and as the family is primitive, $\bar{f} = \bar{g}$. Therefore we have from $\Psi_G(\chi, \bar{g})$ that $\Psi_F(\chi, \bar{g})$.
- $\Longleftarrow$ Suppose $\Psi_F(\chi, \bar{f})$, since also $\Phi_F(\chi, \bar{f})$ then $F(\bar{f})$ is in χ'_t; but $\chi'_t = \chi_t$, therefore $F(\bar{f})$ is in χ_t. □

This implies that the logical schema of F holds in χ. Now we know that if there exists a fixed point χ then this χ is a logical system. Let us find a fixed point.

We define an ordering $\leq$ on $(\chi_i)_i$ as follows: $\chi_i \leq \chi_{i+1}$ if

- $\chi_{ip} \subseteq \chi_{i+1p}$, and
- if x is in χ_{ip}, then x is in χ_{it} iff x is in χ_{i+1t}.

With this ordering we can show that Y is monotonic. Note that the levels can be any ordinal even a transfinite one, for if we are at a finite ordinal i we define $Y(\chi_i) := \chi_{i+1}$ as above. If we are at a limit ordinal j, we define $Y(\chi_j) = \cup\chi_i$ for $i < j$. Applying the fixed point theorem we get a fixed point of Y. The reason for this is of course the monotonicity of the operator Y, as we know that the ordering relation $\leq$ is a partial ordering on all those pairs.

2 Scott Domains and nominalisation

The ordering relation on Scott domains makes predication trivial. For, a predicate P is true of all the objects in the model iff it is true of the bottom element. Both semanticians and computer scientists however, share an interest in quantification and hence this problem of predication that faced Turner (in [Turner' 1984]) is a major issue for those interested in the semantics of either computer or natural languages and who base their work on Scott domains. The problem can be described as follows: Assume a language which has both *objects* and *functions* and assume that wffs are built out of other ones using $\wedge, \vee, \forall, \exists, \ldots$. If the model is

a Scott domain E_∞ then there is no problem interpreting anything which is not a quantified sentence, as the interpretations of all such things are continuous functions and hence belong to the model. Let us choose the following interpretation for the quantifiers $\forall$ and $\exists$

$$[[\forall x\phi]]_{gwt} = \begin{cases} 1 & \text{if for each } d \text{ in } D, [[\phi]]_{g[d/x]wt} = 1 \\ 0 & \text{if for some } d \text{ in } D, [[\phi]]_{g[d/x]wt} = 0 \\ \perp & \text{otherwise} \end{cases}$$

$$[[\exists x\phi]]_{gwt} = \begin{cases} 1 & \text{if for some } d \text{ in } D, [[\phi]]_{g[d/x]wt} = 1 \\ 0 & \text{if for each } d \text{ in } D, [[\phi]]_{g[d/x]wt} = 0 \\ \perp & \text{otherwise} \end{cases}$$

Then the following is a proof of the continuity of the quantifier clause for $\forall$. Assume by induction that we have $[[\phi]]$ is continuous where ϕ does not involve quantifiers. To prove the continuity of $[[\forall x\phi]]$ (i.e. to prove it in $[ASG \longrightarrow [S \longrightarrow EXT]]$ where ASG is the collection of assignment functions, S is the collection of states consisting of worlds and times and EXT is the extensional domain of values), we prove it continuous separately in each of its arguments, according to a theorem related to semantic domains.

Let us prove the continuity of $[[\forall x\phi]]$ for g in ASG. Take an ω-sequence $(g_n)_n$ and prove that: $[[\forall x\phi]]_{\cup g_n wt} = \cup[[\forall x\phi]]_{g_n wt}$.

- Assume $[[\forall x\phi]]_{\cup g_n wt} = 0 \Longleftrightarrow$ by definition,
  $(\exists d \in D)([[\phi]]_{\cup g_n[d/x]wt} = 0) \Longleftrightarrow$ by induction,
  $(\exists d \in D)(\cup[[\phi]]_{g_n[d/x]wt} = 0) \Longleftrightarrow$ by the structure of BOOL,
  $(\exists d \in D)(\exists n \in w)([[\phi]]_{g_n[d/x]wt} = 0) \Longleftrightarrow$ by logical laws,
  $(\exists n \in w)(\exists d \in D)([[\phi]]_{g_n[d/x]wt} = 0) \Longleftrightarrow$ by definition,
  $(\exists n \in w)(\forall x[[\phi]]_{g_n[d/x]wt} = 0) \Longleftrightarrow$ by the structure of Bool,
  $\cup[[\forall x\phi]]_{g_n wt} = 0$
- Assume $[[\forall x\phi]]_{\cup g_n wt} = 1 \Longleftrightarrow$ by definition,
  $(\forall d \in D)([[\phi]]_{\cup g_n[d/x]wt} = 1) \Longleftrightarrow$ by induction,
  $(\forall d \in D)(\cup[[\phi]]_{g_n[d/x]wt} = 1) \Longleftrightarrow$ by the structure of BOOL,
  $(\forall d \in D)(\exists n \in w)([[\phi]]_{g_n[d/x]wt} = 1) \Longleftrightarrow u \subseteq d$ and monotonicity,
  $(\exists n \in w)([[\phi]]_{g_n[d/x]wt} = 1) \Longleftrightarrow$ monotonicity,
  $(\exists n \in w)(\forall d \in D)([[\phi]]_{g_n[d/x]wt} = 1) \Longleftrightarrow$ by definition,
  $(\exists n \in w)(\forall x[[\phi]]_{g_n[d/x]wt} = 1) \Longleftrightarrow$ by the structure of Bool,
  $\cup[[\forall x\phi]]_{g_n wt} = 1$

Therefore $[[\forall x\phi]]$ is continuous.

Note that this interpretation of quantifiers is abandoned later by Turner (in [Turner' 1984]) and he decided to adopt the following clauses instead:

$$[[\forall x\phi]]_{gwt} = \begin{cases} 1 & \text{if for each } d \text{ in } E_\infty \setminus \cup E_n, [[\phi]]_{g[d/x]wt} = 1 \\ 0 & \text{if for some } d \text{ in } E_\infty \setminus \cup E_n, [[\phi]]_{g[d/x]wt} = 0 \\ \perp & \text{otherwise} \end{cases}$$

$$[[\exists x\phi]]_{gwt} = \begin{cases} 1 & \text{if for some } d \text{ in } E_\infty \setminus \cup E_n, [[\phi]]_{g[d/x]wt} = 1 \\ 0 & \text{if for each } d \text{ in } E_\infty \setminus E_n, [[\phi]]_{g[d/x]wt} = 0 \\ \perp & \text{otherwise} \end{cases}$$

Of course working with Scott domains, you have always to check for continuity and this is the case with the new clauses. It can easily be proved that continuity does in fact hold and so we can still think of Scott domains as models.

We now describe the problem which made Turner move from the first definition of quantifiers to the second one. By adopting the first definition, we had: $[[\forall x\phi]]_{gwt} = 1$ iff $(\forall d \in D)([[\phi]]_{g[d/x]wt} = 1)$.

As $[[\phi]]$ is continuous, therefore monotonic and as $u \subseteq d$ (where, as noted above, u is the undefined) for each d in D then we get: $(\forall d \in D)([[\phi]]_{g[d/x]wt} = 1)$ iff $[[\phi]]_{g[u/x]wt} = 1$.

This clause has serious consequences. I shall illustrate this by taking in the formal language an element u' which names u (I.e. $[[u']]_{gwt} = u$ always). Now see what happens if we take ϕ to be: $x = u'$. Applying the above clause we get:

$[[x = u']]_{g[u/x]wt} = 1$ iff $(\forall d \in D)([[x = u']]_{g[d/x]wt} = 1)$ which implies:

$u = u$ iff $(\forall d \in D)(d = u)$.

That is absurd. We have to do something about this and the first solution that one thinks of is to exclude the undefined element from the quantifier clause. Therefore, instead of letting d range over all of D, we let it range over $D*$ (i.e. $D \setminus \{u\}$). But now Scott domains can no longer be models under this interpretation, for we no longer have $[[\forall x\phi]]$ is continuous. If we go back to the proof of continuity given above, we see that we had to use the undefined element in order to prove continuity. Turner, realising this, exploits an important aspect of the structure of Scott domains. We explained earlier the existence of finite and infinite elements in E_∞ and said that for each element d of E_∞, d is the limit of $(e_n)_n$ where e_n belongs to E_n and each E_n is the domain of finite elements. The infinite (or ideal elements) are those which are in $E_\infty \setminus \cup E_n$. By restricting the quantification over these ideal elements only, we can prove again the closure of Scott models. However, by so restricting quantification, only infinite elements can be quantified over and finite elements are ignored.

3 Frege structures and nominalisation

Frege structures are not only a collection of collections of functions (as in the case of E_∞), but they also have a certain logic which works on them, and whose availability solves also the problem of Section 1.1 of part I. Therefore, Frege structures solve both problems of section 1 of part I. In a Frege structure, quantifiers and other connectives are built inductively step by step so that at the fixed point one gets all these logical constants. This availability of logic, makes Frege structures attractive candidates for the semantics of nominalisation. Their other advantage is the type theory that can be built inside them which accommodates self application. In fact, we mentioned in 2.2.1 of part I that the theory of types was not adequate to the semantics of nominalisation. The typing constraints according to Church's type theory are too restrictive for nominalisation and we need to have functions which can apply to themselves or to items of the same type. Abandoning Church's type theory does not imply getting rid of all the typed theories. We can still keep to typed languages but make the typing adequate to deal with nominalisation. This section will develop a type theory based on Frege structures such that for any two types σ, τ the type $< \sigma, \tau >$ is subsumed by the type σ. Some types will be circular or vacuous and they will be responsible for avoiding the paradoxes which

threaten theories that combine type freeness and logic. Basically, our method is to allow type freeness yet to restrict the abstraction of various formulae which belong to various types. Types can be basic or functional space types. Amongst the functional space types we have those types which are circular or vacuous. Abstraction is restricted to those formulae which when abstracted over will belong to a non circular, non vacuous type.

3.1 Polymorphic types

The set of types is the smallest set $\mathcal{T}$ such that

1. p, t, e are in $\mathcal{T}$ are all distinct.
2. If σ, τ are in $\mathcal{T}$ then $< \sigma, \tau >$ is in $\mathcal{T}$.

The types defined in 1 are basic types, p is the type of propositions, t is the type of those true propositions (which are many according to the intensional framework) and e is the type of objects. Of course not every object should be a proposition and not every proposition should be a truth. 2 gives the complex types. We impose a subsumption relation $\leq$ on the types as follows:

1. $\sigma \leq e$
2. $t \leq p$
3. $< \sigma, \tau > \leq \sigma$
   We also require that $\leq$ be a partial ordering and therefore impose the following additional conditions:
4. $\sigma \leq \sigma$
5. if $\sigma \leq \tau$ and $\tau \leq \sigma$, then $\sigma = \tau$
6. if $\sigma \leq \tau$ and $\tau \leq \rho$ then $\sigma \leq \rho$
7. if $\tau \leq \rho$, then $< \sigma, \tau > \leq < \sigma, \rho >$
8. if $\tau \leq p$, then $< \sigma, \tau > \leq << \sigma, \tau >, \tau >$

1- 6 are obvious. As an example of 7, take the propositional functions which are of type $< e, p >$; these functions are also of type $< e, e >$. 8 is there to capture those circular types. In fact we have the following lemma:

Lemma 3.1 *If $\tau \leq p$, then $< \sigma, \tau > = << \sigma, \tau >, \tau > = <<< \sigma, \tau >, \tau >, \tau > = \ldots$*
$< \ldots << \sigma, \tau >, \tau >, \ldots, \tau >$
Proof: obvious from 3, 5 and 8. □

When $\sigma \leq \tau$, we say that τ subsumes, or is a more general type than, σ; intuitively, it means that any expression which is of type σ is also of type τ. Note that e is the maximal element of the partial order, since it subsumes every type. We shall see that the subsumption relation plays a central role in polymorphism, and that there are models of such a typing system; that is, we will have functional domains $X \Rightarrow Y$ such that $(X \Rightarrow Y) \subseteq X$.

Our next task is to extend the definition of type so as to characterize the vacuous types, that is, the types which may be associated with empty domains. It is useful to first introduce the auxiliary notion of a p(ropositional)-chain type. This is defined inductively as follows:

Definition 3.2 *(P-Chain Type)*

1. *If* $\rho \leq p$ *and* $\tau = e$ *or* $\tau = p$ *or* $\tau = t$ *then* $< \tau, \rho >$ *is a p-chain type.*

2. *If* τ *is a p-chain type, and* $\rho \leq p$ *then* $< \tau, \rho >$ *is a p-chain type.*

Example 3.3 $< e,p >, < p,p >, < t,p >, << e,p >,p >$ *(which is equal to* $< e,p >$*),* $<< p,p >,p >$ *(which is equal to* $< p,p >$*),*$<< e,t >, < t,e >>$ *... are p-chain types. Moreover, whenever* σ *is a p-chain type, then so are* $< \sigma,t >, < \sigma,p >, < \sigma, < t,\tau >>$ *and* $< \sigma, < p,\tau >>$ *(for any type* τ*).*

Note however that the following are not p-chain types: $e, < e,e >, < e, < e,e >>, \ldots$

Vacuous types below will be associated with empty domains.

Definition 3.4 *(Vacuous Types)* σ *is a vacuous type iff:*

1. $\sigma =< \tau, \rho >$ *where* τ *and* ρ *are p-chain types, and neither* $\tau \leq p$ *nor* $\rho \leq p$ *or*

2. $\sigma =< \tau, \rho >$ *where* ρ *is a vacuous type, or*

3. $\sigma \leq \tau$*, where* τ *is vacuous.*

From 2 and 3 we can conclude that a function space $< \sigma,\tau >$ is vacuous if its domain σ is vacuous, using $< \sigma,\tau > \leq \sigma$.

Example 3.5 *The following instances of* $\sigma =< \tau, \rho >$ *are vacuous:*

- $\sigma =<< e,p >, < e,p >>$*, by clause 1, since* $\tau = \rho =< e,p >$ *and not* $< e,p > \leq p$.
- $\sigma =<<< e,t >, < t,e >>, << e,t >, < t,e >>>$

There are p-chain types which are not vacuous; for example $< e,p >$. There are types that are vacuous but not p-chains. For example $<< e,p >, < e,p >>$. There are types which are neither vacuous nor p-chains. For example, $e, < e,e >, \ldots$.

3.2 The Syntax of Tpol

The basic expressions of Tpol are as follows:

1. For each type σ, there exists an infinite number of constants. Constants of type σ are referred to as c_σ

2. For each type σ, there exists an infinite number of variables. Variables of type σ are referred to as u_σ.

Expressions of type σ, are defined recursively as follows:

1. $u_\sigma : \sigma$.

2. $c_\sigma : \sigma$.

3. If $\alpha : \tau, u : \sigma$ and $< \sigma,\tau >$ is a type which is not vacuous nor circular, then $\lambda u.\alpha :< \sigma,\tau >$.

4. If $\alpha :< \sigma, \tau >$ and $\beta : \sigma'$, where $\sigma' \leq \sigma$, then $app(\alpha, \beta) : \tau$.

5. If $\alpha : \sigma, \beta : \sigma'$ and $\sigma \leq \sigma'$, then $\alpha =_{\sigma'} \beta : p$.

Suppose $\phi : p$ and $\psi : p$ then

6. $\neg\phi : p$ and $\neg\phi : t$ iff not $(\phi : t)$.

7. $[\phi \vee \psi] : p$ and $[\phi \vee \psi] : t$ iff $\phi : t$ or $\psi : t$.

8. $[\phi \wedge \psi] : p$ and $[\phi \wedge \psi] : t$ iff $\phi : t$ and $\psi : t$.

9. $[\phi \supset \psi] : p$ and $[\phi \supset \psi] : t$ iff $\psi : t$ whenever $\phi : t$.

10. $[\phi \equiv \psi] : p$ and $[\phi \equiv \psi] : t$ iff $\psi : t$ iff $\phi : t$.

11. If $\phi : p$ and u is a variable of any type σ then $\forall u\phi : p$, and $\forall u\phi : t$ iff $\phi[a/u] : t$ for every constant $a : \sigma$.

12. If $\phi : p$ and u is a variable of any type σ then $\exists u\phi : p$, and $\exists u\phi : t$ iff $\phi[a/u] : t$ for some constant $a : \sigma$.

13. If $\sigma' \leq \sigma$, then $\alpha : \sigma'$ implies $\alpha : \sigma$.

Notice that we have placed a syntactic restriction of λ-abstraction to ensure that abstracts never have vacuous or circular types.

3.2.1 Axioms

In our system, self-application is only possible for those expressions which have a complex type; indeed, this is what is required by clause 4 of the syntax above.

- (α) $(\lambda x.\alpha) :< \sigma, \tau >= (\lambda y.\alpha[y/x]) :< \sigma, \tau >$, where y is not free in α.
- (β) $app((\lambda x.\alpha) :< \sigma, \tau >, \beta : \sigma') = \alpha[\beta/x] : \tau$, if $\sigma' \leq \sigma$
- (γ) If $\alpha_1 :< \sigma, \tau >= \alpha_2 :< \sigma, \tau >$ and $\beta_1 : \sigma = \beta_2 : \sigma$, then $app(\alpha_1, \beta_1) : \tau = app(\alpha_2, \beta_2) : \tau$
- (δ) If $(\alpha_1 = \alpha_2) : \sigma$ and $(\alpha_1 = \alpha_3) : \sigma$, then $(\alpha_2 = \alpha_3) : \sigma$
- (ε) If $app(\alpha_1, x) : \tau = app(\alpha_2, x) : \tau$, then $\alpha_1 :< \sigma, \tau >= \alpha_2 :< \sigma, \tau >$ where $x : \sigma$ is not free in α_1, α_2 or any other assumption.
- (ζ) $app(\lambda x.\alpha_1, \beta_2) = app(\lambda x'.\alpha_1, \beta_2)$ where $x : \sigma, x' : \sigma', \sigma' \leq \sigma$, and β_2 is any term of type σ'.
- (θ) $(\alpha : \sigma =_{\sigma'} \alpha : \sigma') \equiv (\alpha : \sigma =_{\sigma} a : \sigma')$ if $\sigma' \leq \sigma$.
- (ρ) $(\alpha : \sigma = \alpha : \sigma) : t$

The following version of η-conversion is derivable:
If $E :< \sigma, \sigma' >$ then $\lambda x.Ex :< \sigma, \sigma' >= E :< \sigma, \sigma' >$ for $x : \tau$ free in E and $\tau \leq\!< \sigma, \sigma' >$
Proof

$$\frac{\lambda x.Ex :< \tau, < \sigma, \sigma' >> \leq \tau \leq\!< \sigma, \sigma' > \qquad E :< \sigma, \sigma' > \qquad y : \sigma \text{ from } (\beta)}{\frac{(\lambda x.Ex)y : \sigma' = Ex[y/x] : \sigma' = Ey : \sigma' \text{ from } (\varepsilon)}{\lambda x.Ex :< \sigma, \sigma' >= E :< \sigma, \sigma' >}}$$

□

Axioms $(\alpha), (\beta), (\gamma)$ and (δ) are standard typed λ-calculus axioms. Axiom (ε) is the extensionality axiom. It says that if α_1 and α_2 give the same results for the same arguments, then they are equal. Axiom (ζ) says that if $f : A \longrightarrow B$ and if f/A' is the restriction of f to $A' \subseteq A$, then f and f/A' give the same results for all elements in A'. Axiom (θ) says that if $\alpha : \sigma'$ and if $\sigma' \leq \sigma$ then saying that α equals to itself in σ is the same as saying that α is equal to itself in σ'. Axiom (ρ) is the reflexivity of $=$.

3.2.2 Russell's and Curry's Paradoxes

Russell's paradox does not occur here because paradoxical expressions of the form $\lambda x.\neg app(x,x)$ are not well-formed. In fact, we have the following lemma:

Lemma 3.6 *If x is of type $< \sigma, p >$, then $\lambda x.\neg app(x,x)$ of type $<< \sigma, p >, p >$ is not well-formed.*

Proof According to the definition of meaningful expressions, it is enough to show that $<< \sigma, p >, p >$ is a circular type. This is obvious from Lemma 3.1. □

In fact, we have an even stronger lemma:

Lemma 3.7 *If x is of type $< \sigma, \tau >$, where $\tau \leq p$, then $\lambda x.\neg app(x,x)$ of type $<< \sigma, \tau >, p >$ is not well-formed.*

Proof Exactly as that of Lemma 3.6. □

With these lemmas, if $x :< \sigma, \tau >$, where $\tau \leq p$, then $app(x,x)$ is of type $\tau \leq p$. Hence $\neg app(x,x)$ is of type p. But $\lambda x.\neg app(x,x)$ is not well-formed in Tpol, due to clause 3 in the definition of the expressions of a type, since its type, namely $<< e, p >, p >$, is circular.

Curry's paradox comes from the presence of (DT), (MP) and β where (DT) and (MP) are as follows:

(DT) $\quad \Gamma \cup \{\phi\} \vdash \psi$ implies $\Gamma \vdash \phi \rightarrow \psi$,
(MP) $\quad \Gamma \vdash \phi \rightarrow \psi$ and $\Gamma \vdash \phi$ implies $\Gamma \vdash \psi$,

If we take a to be the formula $\lambda x.(app(x,x) \rightarrow \bot)$, then

1. $app(a,a) = app(a,a) \rightarrow \bot$ by β-conversion
2. $app(a,a) \vdash app(a,a)$, trivial
3. $app(a,a) \vdash app(a,a) \rightarrow \bot$ by 1
4. $app(a,a) \vdash \bot$ by (MP) applied to 2 and 3
5. $app(a,a) \rightarrow \bot$ by (DT)
6. $\vdash app(a,a)$ by 1
7. $\vdash \bot$ by (MP) applied to 5 and 6

However, our (DT) and (MP) have the following form:
(DT) $\quad \Gamma \cup \{\phi : t\} \vdash \psi : t$ implies $\Gamma \cup \{\phi : p\} \vdash (\phi \to \psi) : t$
(MP) $\quad \Gamma \vdash (\phi \to \psi) : t$ and $\Gamma \vdash \phi : t$ implies $\Gamma \vdash \psi : t$,

If we take a to be the formula $\lambda x.(app(x,x) \to \bot)$, then

1. $app(a,a) = app(a,a) \to \bot$ by β-conversion
2. $app(a,a) : t \vdash app(a,a) : t$, trivial
3. $app(a,a) : t \vdash (app(a,a) \to \bot) : t$ by 1
4. $app(a,a) : t \vdash \bot : t$ by (MP) applied to 2 and 3
5. $app(a,a) : p \vdash (app(a,a) \to \bot) : t$ by (DT)
6. $app(a,a) : p \vdash app(a,a) : t$ by 1
7. $app(a,a) : p \vdash \bot : t$ by (MP) applied to 5 and 6

However, we cannot show that $app(a,a) : p$. In fact $\lambda x.(app(x,x) \to \bot)$ is not well formed due to Lemma 3.6 above as its type is $<< \sigma', p >, p >$. This is because if x is of some type σ, since $app(x,x)$ has to be of type p, we can infer that σ must be of the form $< \sigma', p >$. From this it follows that a is of type $<< \sigma', p >, p >$, which is circular.

3.2.3 Models of Tpol

For the present paper we shall concentrate on $\mathbf{F_0}$, **PROP** and **SET** (where $\mathbf{PROP} \cap \mathbf{SET} = \emptyset$) and then we shall construct domains inside $\mathbf{F_0}$ which represent the types described in our theory Tpol.

Given domains X, Y already in the Frege structure, we build new domains as follows:
(DOM) $\quad X \Rightarrow Y = \{x \in X : \forall x' \in X[app(x,x') \in Y]\}$.

As a special case of (DOM), the domain $(\mathbf{F_0} \Rightarrow \mathbf{PROP}) = \mathbf{SET}$ inside $\mathbf{F_0}$ contains the nominals of propositional functions. Now let us see if the structure of types can be captured by the domains.

Lemma 3.8 *If X, Y are domains, then $(X \Rightarrow Y) \subseteq X$.*
Proof Obvious. □

Lemma 3.9 *If X and Y are domains built as above, then $Y \subseteq Y'$ implies $(X \Rightarrow Y) \subseteq (X \Rightarrow Y')$.*
Proof If $x \in X \Rightarrow Y$, then $\forall x' \in X, app(x,x') \in Y$, by (DOM). Since $Y \subseteq Y'$, it follows that $\forall x' \in X, app(x,x') \in Y'$ and so $x \in X \Rightarrow Y'$. □

Lemma 3.10 *If X and Y are domains built as above, then $X \subseteq X'$ implies $(X \cap (X' \Rightarrow Y)) \subseteq (X \Rightarrow Y)$.*
Proof If $x \in X \cap (X' \Rightarrow Y)$ then $x \in X$, and $x \in (X' \Rightarrow Y)$; by (DOM), $\forall x' \in X', app(x,x') \in Y$. Hence, we have both that $x \in X$ and, since $X \subseteq X'$, $\forall x' \in X, app(x,x') \in Y$. Therefore $x \in X \Rightarrow Y$. □

We now inductively define a relation $\leq$ between arbitrary domains X and the domain **SET**. This relation is related to the notion of a p-chain type which we defined earlier. The relation $X \leq \mathbf{SET}$ holds iff

1. $X = \mathbf{SET}$, or
2. $X = (X' \Rightarrow Y')$ where $X' \leq \mathbf{SET}$ and $Y' \leq \mathbf{PROP}$.

We say that a domain X is *inductively predicable* iff $X \leq \mathbf{SET}$.

Lemma 3.11 *If $X \leq \mathbf{SET}$ then $X \subseteq \mathbf{SET}$.*

Proof The proof is by an easy induction. If $X = \mathbf{SET}$ then the property holds. Assume by induction that the property holds up to X', and show that the property holds for $X = (X' \Rightarrow Y)$ where $X' \leq \mathbf{SET}$. By Lemma 3.9, $(X' \Rightarrow Y) \subseteq X'$, and since $X' \subseteq \mathbf{SET}$ by inductive hypothesis, we have by transitivity that $X \subseteq \mathbf{SET}$. □

The following lemma informs us that if X, Y are inductively predicable then $X \Rightarrow Y$ is empty. When we give the denotation of our various types, we will find that the domains associated with vacuous types are always empty.

Lemma 3.12 $\mathbf{SET} \Rightarrow X$ *is empty whenever* $X \leq \mathbf{SET}$.

Proof The proof is by induction on X:

1. *If $X = \mathbf{SET}$ then $\mathbf{SET} \Rightarrow \mathbf{SET}$ is empty, for the following reason. Suppose x is in $\mathbf{SET} \Rightarrow \mathbf{SET}$. Then for every $x' \in \mathbf{SET}$, $app(x, x') \in \mathbf{SET}$. But $app(x, x')$ is also in $\mathbf{PROP}$, by the definition of x being a $\mathbf{SET}$. Hence, $\mathbf{PROP} \cap \mathbf{SET}$ is not empty. Contradiction.*

2. *Assume $\mathbf{SET} \Rightarrow X$ is empty for $X \leq \mathbf{SET}$, and show that the domain $Y = \mathbf{SET} \Rightarrow (X \Rightarrow Y)$ is empty. Suppose Y is not empty, then if x is in $\mathbf{SET} \Rightarrow (X \Rightarrow Y)$, then for any x' in $\mathbf{SET}$, $app(x, x')$ is in $X \Rightarrow Y$. Hence $app(x, x') \in X$ for any $x' \in \mathbf{SET}$. Hence x is in $\mathbf{SET} \Rightarrow X$ which is empty. Contradiction.* □

Theorem 3.13 *$X \Rightarrow Y$ is empty for $X, Y \leq \mathbf{SET}$.*

Proof The proof is by induction on $X \leq \mathbf{SET}$. If $X = \mathbf{SET}$ then the theorem holds according to Lemma 3.12. Assume the property holds for $X' \leq \mathbf{SET}$, that is, the domain $X' \Rightarrow Y$ is empty for any $Y \leq \mathbf{SET}$; we must show that $(X' \Rightarrow Y') \Rightarrow Y$ is empty for $Y' \subseteq \mathbf{PROP}$. If Z is not empty, i.e. there is some a in $(X' \Rightarrow Y') \Rightarrow Y$, then a is also in $X' \Rightarrow Y'$ and for all x in $X' \Rightarrow Y'$, $app(a, x)$ is in $Y' \subseteq \mathbf{PROP}$. But for all x in $X' \Rightarrow Y'$, $app(a, x)$ is in $Y' \subseteq \mathbf{SET}$. Hence $app(a, x)$ is in $\mathbf{PROP} \cap \mathbf{SET}$ which is empty, absurd. □

Example 3.14 *The following domains are empty:*

- $\mathbf{SET} \Rightarrow \mathbf{SET}$
- $\mathbf{SET} \Rightarrow (\mathbf{SET} \Rightarrow \mathbf{PROP})$
- $(\mathbf{SET} \Rightarrow \mathbf{PROP}) \Rightarrow \mathbf{SET}$ *and*
- *every domain built recursively out of the above three using $\Rightarrow$.*

3.2.4 Semantics of Types

A model M for Tpol is a quadriple $< F, \Rightarrow, C, D >$, where

1. F is a Frege structure in which $\mathbf{PROP} \cap \mathbf{SET} = \emptyset$,
2. $\Rightarrow$ is as defined above by (DOM),
3. The function D which maps types into domains of M is defined as follows:
   - $D_e = \mathbf{F_0}$,
   - $D_p = \mathbf{PROP}$,
   - $D_t = \mathbf{TRUTH}$,
   - $D_{<\sigma,\tau>} = D_\sigma \Rightarrow D_\tau$, where $< \sigma, \tau >$ is non-vacuous.
4. C is an interpretation function which takes any constant of type σ to an object in D_σ.

We also assume the existence of an assigment function g which takes any variable of a non-vacuous type σ to an object in D_σ.

Lemma 3.15 *$D_{<\sigma,\tau>} = (D_\sigma \Rightarrow D_\tau) \subseteq D_\sigma$ where $< \sigma, \tau >$ is non vacuous.*
Proof Obvious by Lemma 3.8. □

Lemma 3.16 *If $D_\tau \subseteq D_\rho$ then $(D_\sigma \Rightarrow D_\tau) \subseteq (D_\sigma \Rightarrow D_\rho)$.*
Proof If a is in $(D_\sigma \Rightarrow D_\tau)$ then $(a \in D_\sigma)$ and $[(\forall x \in D_\sigma)(app(a,x) \in D_\tau)]$ then $(a \in D_\sigma)$ and $[(\forall x \in D_\sigma)(app(a,x) \in D_\rho)]$. □

Lemma 3.17 *If $\sigma \leq \tau$ then $D_\sigma \subseteq D_\tau$.*
Proof by induction on $\sigma \leq \tau$.

1. *If $\sigma = e$ then $\tau = e$ and $D_\sigma = D_\tau$.*
2. *If $\sigma = t$ and $\tau = p$ then $D_\sigma = \mathbf{TRUTH}$ and $D_\tau = \mathbf{PROP}$.*
3. *If $\sigma =< \tau, \rho >$ then $D_\sigma = D_{<\tau,\rho>} \subseteq D_\tau$ by Lemma 3.15.*
4. *If $\tau \leq p$ then $D_{<\sigma,\tau>} \subseteq D_{<<\sigma,\tau>,\tau>}$*
   Proof $(D_\sigma \Rightarrow D_\tau) \subseteq D_\sigma$. Hence by Lemma 3.10,
   $(D_\sigma \Rightarrow D_\tau) \cap (D_\sigma \Rightarrow D_\tau) \subseteq (D_\sigma \Rightarrow D_\tau) \Rightarrow D_\tau$.
   Hence $(D_\sigma \Rightarrow D_\tau) \subseteq (D_\sigma \Rightarrow D_\tau) \Rightarrow D_\tau$.
   Assume that $\tau \leq \rho$ implies $D_\tau \subseteq D_\rho$. Then $D_{<\sigma,\tau>} \subseteq D_{<\sigma,\rho>}$, by Lemma 3.16.
   Note that due to Lemma 3.16, if $< \sigma, \tau >$ is circular, then $D_{\sigma,\tau>} = D_{<<\sigma,\tau>,\tau>}$. □

Lemma 3.18 *If σ is a p-chain type and not $\sigma \leq p$ then $D_\sigma \leq \mathbf{SET}$.*
Proof The proof is by induction on σ.
If $\sigma =< e, \tau >$ where $\tau \leq p$ then $D_\sigma \subseteq \mathbf{F_0} \Rightarrow \mathbf{PROP} = \mathbf{SET} \leq \mathbf{SET}$.
Take $\sigma =< \tau, \tau' >$, where σ is not $\leq p$ and τ is a p-chain type and property holds for τ.

- case 1 *not $\tau \leq p$, then $D_{<\tau,\tau'>} = D_\tau \Rightarrow D_{\tau'}$ where $D_{\tau'} \subseteq \mathbf{PROP}$ and $D_\tau \leq \mathbf{SET}$ by inductive hypothesis. Hence $D_{<\tau,\tau'>} \leq \mathbf{SET}$.*

- case 2 $\tau \leq p$ *then* $< \tau, \tau' > \leq \tau \leq p$. *But it is not the case that* $< \tau, \tau' > \leq p$ *absurd. Hence* τ *is not* $\leq p$. □

Lemma 3.19 *If* σ *is vacuous then* D_σ *is empty.*

Proof If σ *is vacuous, then* $\sigma =< \tau, \rho >$ *where* τ *and* ρ *are p-chain types not* $\leq p$ *or either* τ *or* ρ *is vacuous. If either is vacuous then nothing to prove. Else,* $D_\sigma = D_\tau \Rightarrow D_\rho$ *where* $D_\tau, D_\rho \leq \mathbf{SET}$ *according to Lemma 3.18. Hence by Theorem 3.13,* D_σ *is empty.* □

4 COMPARISON AND CONCLUSION

In this part, we showed that Frege structures provide a solution to both problems; we provided a type theory where any function belongs to its domain and hence the theory is a suitable framework for nominalisation. Now we assess further the advantages one obtains with Frege structures. We start with type freeness and the fact that **SET** is isomorphic to Propositional functions $\mathbf{F_0} \longrightarrow \mathbf{PROP}$ and that $\mathbf{SET} \subseteq \mathbf{F_0}$. Also, we have the two following functionals:

$\| \|_1 : \mathbf{SET} \longrightarrow PF_1$

$\lambda : PF_1 \longrightarrow \mathbf{SET}$.

If we assume that the interpretation of verbs takes place in $\mathbf{F_i}$ for $i \geq 1$ and thus that $[[walk]]$ is in $\mathbf{F_1}$, then we get: $[[to\ walk]]g = \lambda.[[walk]]_g$.

Now it is straightforward to interpret things like *to walk hurts*, for: $[[to\ walk\ hurts]]_g = [[hurt]]_g([[to\ walk]]g) = [[hurt]]_g(\lambda.[[walk]]_g)$.

The advantage of what we just offered lies in the elegance of classifying the denotation of our items. With Montague's and Turner's approaches, one has always to check whether the denotation of an item is in the right domain. With our approach, we do not need to check whether $[[to\ walk]]_g$ is in $\mathbf{F_0}$ or not using some confusing domain equations. All we had to say was that $[[walk]]_g$ is in $\mathbf{F_1}$; therefore $\lambda[[walk]]_g$ is in $\mathbf{F_0}$. This actually seems to be an encouraging advantage about Frege structures: nominalisation and self reference are a natural process inside the Frege structure. It also seems that we have *real application*, unlike in Scott domains where application is only through the isomorphic embedding. This is because instead of interpreting things as above into $\mathbf{F_i}$, for $i \geq 0$, we can restrict everything to $\mathbf{F_0}$ obtaining $[[fun\ is\ fun]]_g = \mathbf{pred}([[fun]]_g, [[fun]]_g)$.

Therefore it seems that by using Frege structures we get the following advantages over Scott domains,

1. Real self application
2. Less cumbersome checking for the right typing than that involved with Scott domains. It is mainly checking the propositionhood of various items to obtain the type of the resulting item.
3. No redundant semantic types
4. Nominalisation seems to flow naturally
5. Quantification

For the sake of completeness, we mention a new approach to a theory of properties proposed by Turner (in [Turner 1987]) which abandons completely the use of Scott domains. Turner's new theory is one which starts from Frege's comprehension principle and restricts

it in such a way that the paradox is no longer derivable. Turner starts with a first order theory which has a pairing system and adds to this theory a new operator p (to serve as the predication operator) together with the lambda operator. Then in this case, if one assumes full classical logic and Frege's comprehension principle, one will certainly derive the paradox; for, take $a = \lambda x.\neg p(x,x)$, then $p(a,a) \leftrightarrow \neg p(x,x)[a/x] \leftrightarrow \neg p(a,a)$. Contradiction.

Of course, the problem does not come from contraction, i.e. $p(\lambda x.A,t) \rightarrow A(t,x)$ is always true. But the converse implication (i.e. expansion) is problematic. This is due to negation, i.e. if A is atomic then we can accept $A(t,x) \rightarrow p(\lambda x.A,t)$. But we cannot accept it when A is like Russell's property, an atomic term proceeded by a negation sign. This is exactly what guides Turner in setting his theory. For the theory now will have the following axioms replacing Frege's comprehension principle:

$(E1)$ $\quad A(t,x) \rightarrow p(\lambda x.A,t)$ when A is atomic.

(R) $\quad p(\lambda x.A,t) \rightarrow A(t,x)$.

(I) $\quad p(\lambda x.p(\lambda y.A,t),u) \rightarrow p(\lambda y.p(\lambda x.A,u),t)$

Now the abandonment of Frege's full comprehension axiom will impose the use of two logics, one inside the predication operator in addition to the usual one for wffs. This is due to the fact that breaking the equivalence between $p(\lambda x.A,t)$ and $A(t,x)$ will disconnect the reasoning about wffs and properties. To build models for T above, one uses the fixed point operator to turn an ordinary model of the first order theory into a model which will validate in it as many instances of the comprehension axiom as possible. It will of course validate only the safe instances whereas the paradoxical ones will oscillate in truth-values. The inductive step to build the model should be obvious. As an example, one can start with the first order model, and an operator PI which is empty at the beginning. Then at the next step, extend PI to also contain the pairs $< [[\lambda x.A]], [[t]]_{gM} >$ such that $[[A]]_{g[[[t]]_g/x]} = 1$ and so on until one gets a limit ordinal χ where PI then is to have in it all the pairs $< e,d >$ such that for some ordinal smaller than this χ, $< e,d >$ belongs to all the intermediate PI's. Now we no longer have a full comprehension principle and we cannot do with properties what we can do with formulae. But there are still a great deal of things that one can identify between properties and wffs; for example, from $P(\lambda x.A,t)$ and $P(\lambda x.B,t)$ one can derive $p(\lambda x.A \wedge B,t)$. Turner showed however that theories of Frege structures are weaker than his theory of properties which is a fact that may stand to our advantage for the following reasons. Firstly, Turner can prove at least as much in his theory as one can in a theory based on Frege structures. Secondly, Turner is paying a price for the strength of his theory — mainly his use of two logics (internal and external) rather than one only. On balance it seems better to use a theory based on Frege structures for properties. Doing so gains the advantages of Turner without the complications.

From the point of view of typing, whereas I use a type free theory, Cocchiarella uses a second order one. There are however some similarities and differences in these two ways of typing that I would like to illustrate. According to axiom (9) under 1.1.2 of Part I, we have $ME_n \subseteq ME_0$ for all $n > 1$, where ME_n are the meaningful expressions of any type n. For us, we have that $ME_n \subseteq ME_0$ for any $1 \leq n$ but the pictures of both approaches are quite different. According to our approach these types are related to each other in a chain like way. That is $ME_n \subseteq MEn-1 \ldots \subseteq ME_0$. For Cocchiarella we have that each $ME_n \subseteq ME_0$ for $n > 1$, yet no relation exists between ME_n and ME_m for $n \neq m$. Also for Cocchiarella, propositions are not included in objects, even though they can be embedded in ME_0 by axiom (8) under the same paragraph. Hence Cocchiarella's whole structure can be understood as a collection of objects, which has a denumerably infinite number of subcollections called

functions but where propositions are outside the domain of objects and can be mapped into it. This structure for Cocchiarella is not a structure of types in the sense that we have in the typing structure in [Kamareddine 1988]. In fact everything that Cocchiarella has so far we have; as can be seen in [Kamareddine 1988], a Frege structure is $\mathbf{F_0}, \ldots, \mathbf{F_n}$, where $\mathbf{F_0}$, is the collection of objects, $\mathbf{F_k}$ is the collection of k-ary functions and each of these $\mathbf{F_k}$, can be embedded in $\mathbf{F_0}$, by λ_k. What we have in addition is a typing system constructed inside $\mathbf{F_0}$, which cannot be found in Cocchiarella's theory. Also, our system is first order in that the quantification over objects and functions is the same, whereas Cocchiarella's system is second order.

5 Acknowledgments

The author is indebted to the anonymous referees whose suggestions played a crucial role in improving both the form and the contents of the paper.

References

[Aczel 1980] P. Aczel, Frege structures and the notions of proposition, truth and set; in: J. Barwise, ed., *The Kleene Synposium*, Studies in Logic 101, North-Holland, NewYork, 1980, pp. 31-60.

[Aczel 1984] P. Aczel, *Non-well founded sets*, CSLI Lecture notes No 14, 1984.

[Aczel 1985] P. Aczel, Properties and propositional functions, privately circulated note, Manchester University, 1985.

[Barendregt 1981] H. Barendregt, *The lambda calculus: its syntax and semantics*, North Holland, 1981.

[Barendregt' 1981] H. Barendregt, The type free lambda calculus, *Handbook of Mathematical logic*, Ed. J.Barwise, North Holland, 1981, pp. 1091-1132.

[Barwise 1987] J. Barwise and J. Etchemendy, *The liar: An Essay on Truth and Circular Propositions*, Oxford University Press, Oxford, 1987.

[Beeson 1987] M. Beeson, *Foundations of constructive Mathematics*, Springer-Verlag, Berlin, 1987.

[Bealer 1982] G. *Bealer, Quality and concept*, Clarendon press, Oxford, 1982.

[Boolos 1971] G. Boolos, The iterative conception of set, *Journal of Philosophy LXVIII*, 1971, pp. 215-231.

[Church 1940] A. Church, A formulation of the simple theory of types, *Journal of Symbolic Logic 5*, 1940, pp. 56-68.

[Cocchiarella 1984] N. Cocchiarella, Frege's Double Correlation Thesis and Quine's set theories NF and ML, *Journal of Philosophical Logic 13*, 1984.

[Cocchiarella 1986] N. Cocchiarella, Conceptualism, Ramified Logic and Nominalised predicates, *Topoi 5*, 1986, pp. 78-87.

[Cocchiarella' 1986] N. Cocchiarella, Philosophical Perspectives on Formal theories of Predication, *Handbook of Philosophical Logic 4*, 1986.

[Feferman 1975] S. Feferman, A language and axioms for explicit Mathematics, *Algebra & Logic*, lecture notes in mathematics 450, 1975, pp. 87-139.

[Feferman 1979] S. Feferman, Constructive theories of functions and classes, Ed. M.Boffa, *Logic Colloquium '78*, North-Holland, Amsterdam, 1979, pp. 159-224.

[Feferman 1981] S. Feferman, Working foundations, A revised version of a paper presented to the workshop The present state of the problem of foundations of Mathematics, Florence, 1981.

[Feferman' 1981] S. Feferman, A theory of variable types, privately circulated note, Stanford University, 1981.

[Feferman 1982] S. Feferman, Inductively presented systems and the formalisation of Meta-Mathematics, Eds. D. Lascar and J. Smiley, *Logic Colloquium '80*, North-Holland, Amsterdam, 1982, pp. 95-128.

[Feferman 1983] S. Feferman, Intensional Mathematics, *Logic Colloquium '83.*

[Feferman' 1983] S. Feferman, Between constructive and classical Mathematics, *Logic Colloquium '83.*

[Feferman 1984] S. Feferman, Intensionality in Mathematics, *Symposium on intensions and set theory*, Meeting of the Pacific Division of the American Philosophical Association, 1984.

[Feferman' 1984] S. Feferman, Towards useful type-free theories I, *Journal of Symbolic Logic 49*, 1984, pp. 75-111.

[Fraenkel 1966] A. Fraenkel, *Set theory and Logic*, Addison-Wesley, U.S.A, 1966.

[Fraenkel 1973] A. Fraenkel, Y. Bar-Hillel and A. Levy, *Foundations of set theory*, North-Holland, Amsterdam, 1973.

[Frege 1970] G.Frege, *Translations from the philosophical writings of Frege*, Ed. P. Geach and M. Black, Basil Blackwell, Oxford, 1970.

[Kamareddine 1988] F. Kamareddine, A polymorphic type theory, with Ewan Klein, Talk given at Titisee conference on unification, 1988.

[Kamareddine 1989] F. Kamareddine, *Semantics in a Frege Structure*, PhD thesis, University of Edinburgh, 1988.

[Kleene 1952] S. Kleene, *Introduction to Metamathematics*, D.Van Nostrand co, Princeton, 1952.

[Kripke 1963] S. Kripke, Semantical considerations on Modal Logic, Ed. L. Linsky, *Reference and Modality*, 1963.

[Kripke 1975] S. Kripke, Outline of a theory of Truth, *The Journal of Philosophy LXXII*, 1975, pp. 690-716.

[Meyer 1981] A. Meyer, What is a model of the Lambda Calculus? Unpublished ms., M.I.T. Lab., Computer Science, 1981.

[Parsons 1979] T. Parsons, The theory of types and ordinary language, Eds. S. Davies and M. Mithun, *Linguistics, Philosophy and Montague Grammar*, 1979, University of Texas Press, Austin.

[Poincaré 1900] H. Poincaré, Du role de l'intuition et de la logique en mathematiques, C.R. du II Congr. Intern. des Math., Paris 1900, pp. 200-202.

[Quine 1969] W. Quine, *Set theory and its Logic*, Belknap Press, Harvard, 1969.

[Russell 1908] B. Russell, Mathematical logic as based on the theory of types, *American Journal of of Math. 30*, 1908, pp. 222-262.

[Scott 1976] D. Scott, Data types as Lattices, Technical Monograph PRG-5, *Siam Journal on Computing 5*, 1976, pp. 522-587.

[Scott 1975] D. Scott, Combinators and classes, Ed. Böhm, *λ-calculus and Computer Science theory*, Lecture notes in Computer Science 37, Springer, 1975.

[Thomason 1974] R. Thomason, *Formal Philosophy; Selected papers by Richmond Montague*, Yale University, 1974.

[Turner 1984] R.Turner, Three Theories of Nominalized Predicates, *Studia Logica XLIV2*, 1984, pp. 165-186.

[Turner' 1984] R. Turner, Nominalization and Scott's Domains II, *Notre Dame Journal of Formal Logic 26*, 1984.

[Turner 1987] R. Turner, A theory of properties, *Journal of Symbolic Logic 52*, 1987.

In this series appeared:

90/1	W.P.de Roever- H.Barringer- C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, p. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems, p. 17.
90/16	F.S. de Boer C. Palamidessi	A fully abstract model for concurrent logic languages, p. p. 23.
90/17	F.S. de Boer C. Palamidessi	On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29.

90/18	J.Coenen E.v.d.Sluis E.v.d.Velden	Design and implementation aspects of remote procedure calls, p. 15.
90/19	M.M. de Brouwer P.A.C. Verkoulen	Two Case Studies in ExSpect, p. 24.
90/20	M.Rem	The Nature of Delay-Insensitive Computing, p.18.
90/21	K.M. van Hee P.A.C. Verkoulen	Data, Process and Behaviour Modelling in an integrated specification framework, p. 37.
91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.

91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.

91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.
91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.
92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.