NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

CMU-CS-84-[illegible] 157

Typing and convergence in the Lambda Calculus

Daniel Leivant

Department of Computer Science
Carnegie-Mellon University

Abstract. We use a perception of typing in the λ-Calculus, according to which the typing of an expression E expresses semantic properties of E in models over λ-expressions, to exhibit natural and uniform proofs of generalizations of theorems of Girard [Gir71,72] and of Coppo, Dezani and Veneri [CDV81] about the solvability, normalizability and strong normalizability of suitably typed expressions.

We also generalize the theorems of [CDV81] which go in the opposite direction, showing the typeability of solvable and of normalizable expressions.

Introduction and background.

Simple and second-order types in programming and in the Lambda Calculus.

Types have been used in programming languages mainly as a device to enforce disciplined programming; in particular — to guarantee that the composition of procedures is meaningful. A reasonable expectation is that typing should guarantee the absence of infinite looping of procedure calls, provided of course recursive calls are not used.

To state this property precisely one first distils, out of the given programming language, its non-recursive procedure call mechanism. The result is a typed lambda calculus. Then one proves that all expressions in that lambda calculus convert to a normal form along any reduction sequence. This property is referred to as the *Strong Normalization Property* of the given programming language (and of the corresponding typed lambda calculus). The *Strong Normalization Theorem*, for a given programming language or lambda calculus, is simply the statement that the Strong Normalization Property is true of that language.

Languages such as PASCAL and ALGOL68 have relatively simple type disciplines, and their procedure call mechanism is represented by the simply typed lambda calculus, or fragments thereof [Mor68, FLO83§4]. The Strong Normalization Theorem for the simply typed lambda calculus is rather straightforward [Tai67, Mor68, San67, And71, FLO83§6.1].

A new dimension was introduced to the subject of typing with the invention of second-order and higher-order type disciplines [Gir71,72, Rey74, FLO83]. Generic (parametric) types are motivated by the pragmatic wish to avoid repeated definition of the same procedure for different types. Once generic procedures are used, conceptual consistency dictates that they also be allowed as arguments of other procedures, the latter should again be permitted as arguments, and so on. This gives rise to the full second-order type discipline of Girard and Reynolds. The Lambda Calculus supplemented with the second-order type discipline is referred to as the *Second-Order*, or *Polymorphic*, Lambda Calculus.

Normalization in the Second-Order Lambda Calculus.

The Normalization Theorem for the Second-Order Lambda Calculus, while true, is not as easily proved as normalization for the simply typed calculus. In fact, it can not be proved even in Second-Order Arithmetic [Lei79]. A combinatorially isomorphic problem, that of strong normalization for Second-Order Logic, was an outstanding open problem in Mathematical Logic in the 50's and 60's.

Initially, a Normal Form Theorem was proved [Tai66, Tak67, Pra67,68], whereby every proof in second-order logic can be converted to a normal form by *some* reduction sequence. The full Strong Normalization Theorem was proved by Girard [Gir71,72], by injecting second-order reasoning into the combinatorial method of Tait [Tai67]. (Independently, Harvey Friedman injected second-order reasoning to a different syntactic method, Kleene's slash, with related results [Fri73].) Girard's method was then applied and refined, in different guises, to various formal calculi [Pra71, Mar71, Ste72, Tai75, FLO83].

Two philosophies of typing.

The observation that objects that can be typed are guaranteed to convert to normal form goes back, for Combinatory Logic, to Curry and Feys [CF58]. However, they were thinking of a type not as an inherent property of an object, but as a semantic notion, where one object can be assigned different types. This *semantical* perception of typing has been pursued and developed, for example, in [Cop80, Pot80, CDV81, MS82, BCD83, MPS84, BM84]. It is motivated by a wish to study the functionality properties of (untyped) λ-expressions, and is quite different from the view underlying the *Typed* Lambda Calculus. In the type calculus each object carries its type, a perception we dub the *ontological* discipline. To say that "E *is of type* $\tau \rightarrow \sigma$" implies, in the ontological discipline, that the domain of E consists precisely of all objects of type τ. In the semantical discipline, the same statement implies that the domain of E merely contains all objects of type τ, which E maps to objects of type σ.

Ontological typing arises not only in programming languages, but also in connection with Proof Theory. In Church's Theory of Types [Chu40], logic is coded directly in the Typed Lambda Calculus. Related to this is the formula-as-type/ derivation-as-λ-expression isomorphism discovered by Howard and others [CF58, How80, deB70]. The two approaches are unified in [Mar82].

In spite of the fundamental difference between the ontological and the semantical viewpoints of typing, the computational aspects of the two disciplines are related [Lei83,§6 below]. A typing calculus for the ontological discipline is in fact isomorphically embeddable in a typing calculus for the semantical discipline.

Characterization of convergence properties by type systems.

Within the semantical discipline it makes sense to consider implications that are inverse to the Normalization Theorems: can we assign a type to every strongly normalizable λ-expression? More generally, we may set forth a broader notion of typing, and consider the question of whether a certain convergence property of λ-expressions, such as solvability or normalizability, implies typability. This kind of question was considered by Coppo, Dezani and Veneri [CDV81]. They showed that for a suitable notion of types, typing of a λ-expression is guaranteed by its solvability. They also showed that a more restrictive notion of typing is guaranteed by normalizability. In fact, these notions of type characterize exactly the solvable and the normalizable λ-expressions, respectively.

The results of this paper.

The purpose of this note is to integrate and generalize the techniques and results relating typing to convergence properties in the Lambda Calculus. The main ideas we build on are those of Tait [Tai67], Girard [Gir71,72], and Coppo, Dezani and Pottinger [Cop80 ,CD80, Pot80, CDV81].

Our basic results are all developed within the semantical framework. One advantage is that this framework permits a richer notion of type, and therefore more general results. This choice is fecund, however, even if one is only interested in ontological typing. The semantical framework allows a more natural and transparent development of the normalization proofs which so far have been presented within the ontological discipline. This is mostly because in the semantical approach, combinatorial properties, such as Tait's "computability" [Tai67, Gir71, Pra71, Ste72], can be viewed as semantic properties of expressions in models built over λ-expressions. In addition, the semantical approach suppresses irrelevant details.

The relations between typing and convergence properties are summarized in theorems 5.7, 5.10 and 5.12 below. Some specific new results we obtain are analogues and generalizations of the results of [CDV81]. For example, we show that a λ-expression is strongly normalizable iff it has a type which can be derived without reference to the universal type ω. Also, we show that a λ-expression is solvable whenever it can be assigned a type where ω does not appear strictly positively, even if that type is second-order. We point out that the Coppo-Dezani-Veneri characterization of the normalizable expressions fails for second-order types, in contrast to the characterizations of solvable expressions and of strongly normalizable expressions. Thus, strong normalizability and solvability are properties which are, in a sense, more "stable" than normalizability.

Finally, we show in §6 how the Strong Normalization Theorem for the Second-Order Lambda Calculus falls out as a corollary of the corresponding Strong Normalization Theorem in the semantical discipline.

1. Polymorphic Typing.

Polymorphic types are defined inductively as follows (compare [Rey74, Gir72, FLO83, MS82]).

$R \in TV$ — type variables (a countable set)

$\tau \in T$ — polymorphic type expressions

$\tau ::= R \mid \omega \mid \tau_1 \to \tau_2 \mid \tau_1 \wedge \tau_2 \mid \forall R\tau$

A type is *first-order* if it has no occurrence of $\forall$, *proper* if it has no occurrence of ω, $\wedge$-*free* if it has no occurrence of $\wedge$. An occurrence of a type expression σ in τ is *positive* [*negative*] if it is in the negative (=left side) scope of an even [odd, respectively] number of instances of $\to$ in τ. The occurrence is *strictly positively* if it is in the negative scope of no $\to$. A type expression τ is [*n-, p-, sp-*] *proper* if ω does not occur in τ [negatively, positively, strictly positively, respectively].

A *type statement* is an expression $E{:}\tau$ where E is a λ-expression, and τ is a type expression. Following de Bruijn [deB70] we use the term *context* for a finite function C from λ-calculus variables to type expressions, which we sometimes write as a list $x_1{:}\tau_1, \ldots, x_k{:}\tau_k$ of type statements. A *typing* is an expression $C \vdash E{:}\tau$, where C is a context, and $E{:}\tau$ is a type statement.

The typing $C \vdash E:\tau$ is *initial* if τ is ω, or if E is a variable and $C(E)=\tau$. A *type deduction* is a derivation of a typing from initial typings, using the following *inference rules*. The *structural* inference rules are $\rightarrow I$ and $\rightarrow E$ (I for "introduction", E for "elimination"):

$$\frac{C,x:\sigma \vdash E:\rho}{C \vdash \lambda x.E:\sigma \rightarrow \rho} \qquad \frac{C \vdash E:\sigma \rightarrow \rho \quad C \vdash F:\sigma}{C \vdash EF:\rho}$$

The remaining rules are *stationary*, in that they refer to a fixed λ-expression. These rules are $\wedge I$, $\wedge E$, $\forall I$ and $\forall E$:

$$\frac{C \vdash E_1:\tau_1 \quad C \vdash E_2:\tau_2}{C \vdash E:\tau_1 \wedge \tau_2} \qquad \frac{C \vdash E:\tau_1 \wedge \tau_2}{C \vdash E:\tau_i} \ (i=1 \text{ or } 2)$$

$$\frac{C \vdash E:\sigma}{C \vdash E:\forall R\,\sigma} \ (R \text{ not free in } C) \qquad \frac{C \vdash E:\forall R\,\sigma}{C \vdash E:\sigma[\rho/R]} \ (\rho \text{ free for } R \text{ in } \sigma)$$

In the rule $\forall E$ the substituted type expression ρ is the *eigen-type* of the inference. A derivation is *essentially first-order* if only type-variables are used as eigen-types. A typing $C \vdash E:\tau$ is *properly derived* if it has a derivation with proper types only. It is *sp-proper* if τ is sp-proper. It is *p-proper* if τ is p-proper and every type $C(x)$ is n-proper. If P is a property of typings, we say that E *has a* P *typing* if there is a typing $C \vdash E:\tau$ with property P.

2. The semantic meaning of typing.

We now show that types can be understood as coding semantic properties of λ-expressions in second-order models over λ-expressions. Sets of λ-expressions we refer to are:

$\Lambda \equiv$	the set of all λ-expressions
$\Lambda_I \equiv$	the set of λI-expressions
$\mathbf{N} \equiv$	the set of normalizable λ-expressions
$\mathbf{N}_I \equiv$	the set of normalizable λI-expressions
$\mathbf{S} \equiv$	the set of strongly normalizable λ-expressions
$\mathbf{L} \equiv$	the set of solvable λ-expressions

We define a family of second-order formulas φ^τ, $\tau \in \mathbf{T}$, by induction on τ. $\varphi^\tau[E]$ asserts that the λ-expression E behaves functionally as a mapping of type τ. We use $X,\ldots$ as individual logical variables, and $R,\ldots$ as unary predicate variables.

$$\varphi^{R}[E] \equiv R(E)$$
$$\varphi^{\omega}[E] \equiv \text{true}$$
$$\varphi^{\sigma\to\rho}[E] \equiv \forall X(\varphi^{\sigma}[X] \to \varphi^{\rho}[EX])$$
$$\varphi^{\sigma\wedge\rho}[E] \equiv \varphi^{\sigma}[E]\wedge\varphi^{\rho}[E]$$
$$\varphi^{\forall R.\sigma}[E] \equiv \forall R\,\varphi^{\sigma}[E]$$

Lemma 2.1. $\varphi^{\sigma[\rho/R]}$ is syntactically identical to $\varphi^{\sigma}[\varphi^{\rho}/R]$.

Proof. Straightforward induction on σ.■

Remark. The mapping $\sigma \Rightarrow \varphi^{\sigma}$ is right inverse to the mapping $\varphi \Rightarrow \tau(\varphi)$ of [Lei83b], i.e. $\tau(\varphi^{\sigma})\equiv\sigma$. We have here a duality between the (modified) notion of "formula as type" [How80, Lei83b], and of the notion above of "type as formula".

We wish to consider models of monadic second order formulas, where individual elements are λ-expressions. The models we have in mind are of the kind defined by Henkin [Hen], i.e., pairs (U,C), where U is a set of λ-expressions (the universe of the models), and C is a collection of subsets of U. (We identify monadic predicates with their extensions.)

One natural condition on both U and the elements of C is that they be closed under a weak form of β-equality[1]. We say that a set $Q\subseteq U$ is *extensional in* U if it is closed under $=_{\beta}$ in U; i.e.,

- if $E\in Q$ and E' comes from E by replacing an occurrence of $(\lambda xF)G$ by $F[G/x]$, then $E'\in Q$;
- if $E\in Q$ and E' comes from E by replacing $F[G/x]$ by $(\lambda xF)G$, where $G\in U$, then $E'\in Q$.

We say that Q is simply *extensional* if Q is extensional in itself. For example, N and L are extensional in Λ; Λ_I is extensional, but not extensional in Λ; N_I is extensional in Λ_I, but not in Λ; S is extensional, but not extensional in Λ. In a model we shall require that U be an extensional set, and that each set in C be extensional in U.

Another natural condition on the underlying universe U of a model is that it be closed under taking subexpressions.[2]

In summary, a pair (U,C) is a *model* if U is an extensional set closed under taking subexpressions, and C is a non-empty collection of subsets of U extensional in U.

Suppose (U,C) is a model, φ a monadic second-order formula. We give the standard meaning to the semantic satisfaction relation $U,C,\eta\models\varphi$, where η is a valuation of free individual variables as members of U, and of free predicate variables as elements of C. We also define $[\tau]_{UC\eta} \equiv \{E\in U \mid U,C,\eta\models\varphi^{\tau}[E]\}$. When U and C are clear from the context we use the abbreviations $\eta\models\varphi$ and $[\tau]_{\eta}$.

1) Note that strong normalizability is not preserved under unrestricted β-equality. However, for our results about solvable and normalizable expressions we may require that models be closed under full β-equality.
2) Actually, a weaker conditions will be used: if $EF\in U$, then $E'\in U$.

Lemma 2.2. Suppose (U,C) is a model, η a valuation in (U,C). Then $[\tau]_{UC_\eta}$ is extensional in U, for each type τ.

Proof. Straightforward induction on τ.■

Let T_0 be a set of types. A model (U,C) is *complete* [*for* T_0] if $[\tau]_{UC_\eta} \in C$ for every type expression τ [in T_0] and every valuation η in (U,C).

Lemma 2.3 (Semantic typing). (i) Suppose (Λ,C) is a complete model. If the typing $\{y_i:\sigma_i\}_i \vdash E:\tau$ is derived, then $\wedge_i \varphi^{\sigma_i}(Y_i) \to \varphi^\tau(E[\vec{Y}])$ is valid in (Λ,C), i.e.

$$\eta \models \wedge_i \varphi^{\sigma_i}(Y_i) \to \varphi^\tau(E[\vec{Y}]),$$

for all valuations η in (Λ,C). (We write $E[\vec{Y}]$ for $E[Y_1/y_1 \cdots Y_k/y_k]$.)
(ii) Suppose (U,C) is a model complete for proper types. If the typing $\{y_i:\sigma_i\}_i \vdash E:\tau$ is properly derived, then $\wedge_i \varphi^{\sigma_i}(Y_i) \to \varphi^\tau(E[\vec{Y}])$ is valid in (U,C).

Proof. By induction on the derivation Δ of $\{y_i:\sigma_i\}_i \vdash E:\tau$.

If Δ consists of an initial typing, and $E \equiv y_i$, then the lemma is trivial. If $\tau \equiv \omega$ (for case (i) of the lemma), then $\varphi^\tau \equiv$ true, and the lemma is trivial again.

Suppose that the last inference of Δ is $\to I$, as displayed in §1, and $\tau \equiv \sigma \to \rho$. By induction assumption

$$\eta[F/X] \models \wedge_i \varphi^{\sigma_i}(Y_i) \wedge \varphi^\sigma(F) \to \varphi^\rho(E[\vec{Y},F]),$$

for all $F \in U$. So

$$\eta \models \wedge_i \varphi^{\sigma_i}(Y_i) \to \forall X(\varphi^\sigma(X) \to \varphi^\rho(E[\vec{Y},X]).$$

By lemma 2.2 $\eta \models \varphi^\rho(E[\vec{Y},X]) \to \varphi^\rho((\lambda x.E[\vec{Y}])X)$. Also, since U is closed under taking subexpressions, $(\lambda x.E[\vec{Y}])X \in U$ implies $\lambda x E[\vec{Y}] \in U$. So

$$\eta \models \wedge_i \varphi^{\sigma_i} \to \varphi^\tau(\lambda x.E).$$

Suppose that the last inference of Δ is $\to E$, as displayed §1. By induction assumption

$$\eta \models \wedge_i \varphi^{\sigma_i}(Y_i) \to \varphi^{\sigma\to\rho}(E[\vec{Y}]) \quad \text{and} \quad \eta \models \wedge_i \varphi^{\sigma_i}(Y_i) \to \varphi^\sigma(F[\vec{Y}]).$$

So

$$\eta \models \wedge_i \varphi^{\sigma_i}(Y_i) \to \varphi^\rho(EF[\vec{Y}]),$$

by the definition of $\varphi^{\sigma\to\rho}$.

Suppose that the last inference of Δ is $\forall E$. By induction assumption

$$\eta \models \wedge_i \varphi^{\sigma_i}(Y_i) \to \varphi^{\forall R\sigma}(E[\vec{Y}]),$$

i.e.

$$\eta \models \wedge_i \varphi^{\sigma_i}(Y_i) \to \forall R\, \varphi^\sigma(E[\vec{Y}]).$$

But we assume that (U,C) is complete, so

$$\eta \models \forall R\, \varphi^\sigma(E) \to \varphi^\sigma(E)[\varphi^\rho/R].$$

Moreover, by lemma 2.2. $\varphi^\sigma(E)[\varphi^\rho/R] \equiv \varphi^{\sigma[\rho/R]}(E)$. Altogether we get

$$\eta \models \wedge_i \varphi^{\sigma_i}(Y_i) \to \varphi^{\sigma[\rho/R]}(E[\vec{Y}]).$$

The cases for the rules $\forall I$, $\wedge I$ and $\wedge E$ are trivial.■

In fact the proof establishes a slightly stronger fact.

Lemma 2.4. Suppose that Δ is a derivation of the typing $\{y_i:\sigma_i\}_i \vdash E:\tau$. If (U,C) is a model complete for the set of eigen-types in Δ, then $\wedge_i \varphi^{\sigma_i}(Y_i) \rightarrow \varphi^{\tau}(E[\bar{Y}])$ is valid in (U,C).■

3. Model constructions.

We describe two straightforward model constructions. The first is a trivial construction, which we use in the proof of theorem 4.7 below. The second yields complete models, which we use in the proofs of theorems 4.1 and 4.3.

Lemma 3.1. Suppose U is an extensional set closed under subexpressions, and suppose that V is extensional in U. Let $V_U \equiv \{V,U\}$. Then (U,V_U) is a model.

Proof. Immediate from the conditions.■

For example, $(\Lambda,\{N,\Lambda\})$ is a model.

Let U be a set, z a λ-variable. Define $zU^* \equiv \{zE_1 \cdots E_k \mid E_i \in U, k \geq 0\}$, and $C_U \equiv \{Q \mid zU^* \subseteq Q \subseteq U, R \text{ is extensional in } U\}$.

Lemma 3.2. Suppose U is extensional and closed under taking subexpressions. Assume, moreover, that $zU^* \subseteq U$. Then (U,C_U) is a complete model.

Proof. That (U,C_U) is a model is immediate from the conditions (the last condition guarantees that $C_U \neq \emptyset$). To show that (U,C_U) is complete we prove, by induction on proper types τ, that $zU^* \subseteq [\tau]_\eta$, for each valuation η in (U,C_U). Let η be such a valuation.

If τ is a type variable R, then $[\tau]_\eta = \eta(R) \in C_U$.

Suppose $\tau \equiv \sigma \rightarrow \rho$. To show $zU^* \subseteq [\tau]_\eta$, consider an arbitrary $z\bar{E} \in zU^*$. We have $z\bar{E}F \in zU^*$ for any $F \in U$ trivially, and $zU^* \subseteq [\rho]_\eta$ by induction assumption. Thus $\eta \models \varphi^{\tau}(z\bar{E})$, i.e. $z\bar{E} \in [\tau]_\eta$.

The cases where τ is conjunctive or universal are trivial.■

Examples. (S,C_S) and (Λ,C_Λ) are complete by lemma 3.2.

4. Convergence properties implied by typing.

We show that typing properties of λ-expressions imply certain convergence properties. Namely,

- every expression with a properly derived typing is strongly normalizable;
- every expression with an sp-proper typing is solvable; and
- every expression with a first order-derived, p-proper typing, is normalizable.

For the first of these results we use the model (S,C_S).

Theorem 4.1. (Girard [Gir]). If E has a properly derived typing then E is strongly normalizable.

Proof. Assume $B \vdash E:\tau$ is properly derived. It suffices to consider closed expressions E, because $E \in S$ iff $\lambda \vec{x}.E \in S$. Assume the theorem's premise (with $B \equiv \emptyset$). Then, by lemma 2.3, $E \in [\tau]_{UC_\tau}$ for any complete model (U,C), and any valuation η therein. Thus, $E \in [\tau]_{SC_{S\tau}}$, by lemma 3.2. Taking an arbitrary η in (S,C_S), $E \in [\tau]_{SC_{S\tau}} \subseteq S$.■

Remark. The theorem fails if only the derived typing is required to be proper, but not the entire derivation:

Example. There is a derived typing $\vdash E:\tau$, where τ is proper and E is not even normalizable. Let Ω be some non-normalizable expression, and consider $E \equiv \lambda z.z\Omega$. It is easy to see that $E:\beta \to Q$ is derivable, where $\beta \equiv \forall R.R$ and Q is a type variable.■

Next, we consider expressions that can be assigned an sp-proper typing. The model we use here is (Λ,C_Λ), which by 3.2 is complete. Note that $L \in C_\Lambda$, since L is extensional in Λ.

Lemma 4.2. Let ζ be the valuation in (Λ,C_Λ) which assigns L to all predicate variables. Suppose that τ is an sp-proper type. Then $[\tau]_\zeta \subseteq L$.

Proof. By induction on sp-proper types. If τ is a type variable R, then $[\tau]_\zeta \equiv \zeta(R) \equiv L$.

Suppose $\tau \equiv \sigma \to \rho$. To show $[\tau]_\zeta \subseteq L$ assume $E \in [\tau]_\zeta$. Since (Λ,C_Λ) is complete, $[\sigma]_\zeta \in C_\Lambda$, hence $z\Lambda^* \subseteq [\sigma]_\zeta$, and $z \in [\sigma]_\zeta$. Thus $Ez \in [\rho]_\zeta$, and by induction assumption $Ez \in L$, since ρ is sp-proper. Hence $E \in L$.

Suppose $\tau \equiv \forall R.\sigma$. Then $[\tau]_\zeta \equiv \bigcap\{[\sigma]_\theta \mid \theta \equiv \zeta[Q/R], Q \in C_\Lambda\} \subseteq [\sigma]_\zeta$, and the latter is a subset of L by induction assumption.

The case where τ is conjunctive is trivial.■

Theorem 4.3. If E has an sp-proper typing then E is solvable.

Proof. It suffices to consider closed expressions E. Assume $\vdash E:\tau$ is derived, where τ is sp-proper. By lemma 2.2 it follows that $E \in [\tau]_\eta$ for any closed model (Λ,C) and η therein. By lemma 4.2 $[\tau]_\zeta \subseteq L$, so $E \in L$.■

A similar statement is proved in [CDV81], but where only first-order types are treated.

Finally, we show that if E has a p-proper typing then E is normalizable, *provided* the typing derivation is essentially first-order (as defined in §1).

The model we use here is (Λ,N_Λ).

Lemma 4.4. Let τ be first-order and proper. Let ζ be a valuation in (Λ,N_Λ) which assigns N to all predicate variables. Then $zN^* \subseteq [\tau]_\zeta \subseteq N$.

Proof. By induction on proper types. If τ is a type variable R, then $[\tau]_\zeta \equiv \zeta(R) \equiv N$.

Suppose $\tau \equiv \sigma \to \rho$. To show $[\tau]_\zeta \subseteq N$ assume $E \in [\tau]_\zeta$. By induction assumption, $z \in [\sigma]_\zeta$, hence $Ez \in [\rho]_\zeta$, and by induction assumption $Ez \in N$. Thus $E \in N$.

To show $zN^* \subseteq [\tau]_\zeta$, let $z\vec{E} \in zN^*$. For any $F \in [\sigma]_\zeta$ we have $F \in N$, by induction assumption, so $z\vec{E}F \in zN^*$. By induction assumption $zN^* \subseteq [\rho]_\zeta$, so $z\vec{E}F \in [\rho]_\zeta$. Thus $z\vec{E} \in [\tau]_\zeta$.

The case where τ is conjunctive is trivial.■

Remark. Unlike the proof of 4.2, we need in 4.4 the lower bound condition $zN^* \subseteq [\tau]_\zeta$. This breaks down if $\tau \equiv \forall R.\sigma$.

Lemma 4.5. Let ζ be a valuation in (U,C), which agrees with a valuation η, except that $\zeta(R) \supseteq \eta(R)$ for a particular type-variable R. If R does not occur positively in τ then $[\tau]_\zeta \subseteq [\tau]_\eta$.

Proof. By induction on τ, simultaneously with the dual statement: If R does not occur negatively in τ then $[\tau]_\zeta \supseteq [\tau]_\eta$.■

Lemma 4.6. Assume τ is p-proper, ζ a valuation in a model (Λ,C). There is a proper type θ and a valuation κ such that $[\tau]_\zeta = [\theta]_\kappa \subseteq [\theta]_\zeta$.

Proof. Let θ be τ with ω replaced throughout by some fresh type variable R. Let $\kappa \equiv \zeta[\Lambda / R]$. Then $[\tau]_\zeta = [\theta]_\kappa$ by a trivial induction on τ, and $[\theta]_\kappa \subseteq [\theta]_\zeta$ by 4.5.■

Theorem 4.7. (Coppo-Dezanni-Venneri [CDV81]). If E has a p-proper typing, essentially first-order derived, then E is normalizable.

Proof. It suffices to consider closed E. If $\vdash E:\tau$ has an essentially first-order derivation, where τ is p-proper, then $\varphi^\tau(E)$ is valid in (Λ,N_Λ), by lemma 2.4. and lemma 3.1, i.e. $E \in [\tau]_\eta$ for every valuation η in (Λ,N_Λ). In particular, $E \in [\tau]_\zeta$, where ζ is as above. By 4.6 $[\tau]_\zeta \subseteq [\theta]_\zeta$, and by 4.4 $[\theta]_\zeta \subseteq N$. So $E \in N$.■

By the Example above the restriction in the theorem to essentially first order derivations is essential. Note that if the condition that the derivation be first-order is replaced by the condition that *all* types in the derivation be p-proper (sub-types not being counted), then no initial typing $B \vdash E:\omega$ can occur in the derivation. All occurrences of ω can then be replaced by some fresh type variable, and all types become proper. We are then back to the case of theorem 4.1.

5. Convergence properties implying typing; characterization theorems.

All expressions have trivially the type ω. We show here that expressions satisfying certain convergence properties can be assigned more interesting types [CDV81]. Namely,

- every solvable expression E has an sp-proper typing;
- every strongly normalizable expression has a properly derived first-order typing; and
- every normalizable expression has a p-proper first-order typing.

Lemma 5.1. If E is in head normal form then E has a first-order sp-proper typing.

Proof. Clearly $z:\omega \to \omega \to \cdots \to \omega \to R \vdash zF_1 \cdots F_k:R$ is derived, where there are k occurrences of ω, and R is a type variable. The lemma follows.■

Let C_1 and C_2 be contexts. We write $C_1 \wedge C_2$ for the context C that agrees with C_1 and C_2 for the arguments on which C_1 and C_2 do not disagree, and $C(x) = C_1(x) \wedge C_2(x)$ if $C_1(x)$ and $C_2(x)$ are both defined and distinct.

Lemma 5.2. If Δ derives $C \vdash E:\tau$, C' is any context, then $C \wedge C' \vdash E:\tau$ is derived by a derivation Δ' identical to Δ, except possibly for inference of typings initial in Δ from typings initial in Δ', by instances of $\wedge E$.

Proof. Trivial induction on Δ.■

Lemma 5.3. Assume $C \vdash E[F/x]:\tau$ is derived by Δ. Then either

(1) $C \vdash E:\tau$ is derived, and the derivation is proper if Δ is proper; or

(2) there is a type σ such that $C \wedge \{x:\sigma\} \vdash E:\tau$ and $C \vdash F:\sigma$ are derived. Moreover, if Δ is proper then so are these two derivations.

Proof. By induction on the derivation Δ of $C \vdash E[F/x]:\tau$. (1) applies if x is not free in E. (2) applies otherwise, where σ is the conjunction of all types assigned to F in Δ (for occurrences of F substituted for x in $E[F/x]$).■

Lemma 5.4. If $C \vdash E[F/x]:\tau$ is derived, then so is $C \vdash (\lambda x.E)F:\tau$. Moreover, if the former is properly derived, and $C' \vdash F:\rho$ is properly derived (for some ρ), then $C \wedge C' \vdash (\lambda x.E)F:\tau$ is properly derived.

Proof. Suppose that Δ derives $C \vdash E[F/x]:\tau$. If case (1) of 5.3 applies, then, by $\rightarrow I$, $C \vdash \lambda x E:\alpha \rightarrow \tau$ is derived for any type α. Let $\alpha \equiv \omega$ in the general case, $\alpha \equiv \rho$ if the extra lemma assumption holds, and use $\rightarrow I$.

If case (2) of 5.3 applies, then $C \vdash \lambda x E:\sigma \rightarrow \tau$ and $C \vdash F:\sigma$, so $C \vdash (\lambda x E)F:\tau$ is derived. Moreover, if Δ is proper then the last derivation is proper, by 5.3.■

Lemma 5.5. Suppose E β-converts to E' (in one step), and $C \vdash E':\tau$ is derived. Then $C \vdash E:\tau$ is derived.

Moreover, suppose that $C \vdash E':\tau$ is properly derived. If E' comes from E by replacing an occurrence H of $(\lambda x.F)G$ by $F[G/x]$, and G has a properly derived typing, then $C' \vdash E:\tau$ is properly derived for some C'.

Proof. By straightforward induction on the depth of H in E, using 5.4 for the base case.■

Theorem 5.6. [CDV81]. If a λ-expression E is solvable then it has an sp-proper typing.

Proof. By induction on the length of the shortest β-conversion leading from E to an expression in head normal form. The induction basis is lemma 5.1. The induction step is lemma 5.5.■

Corollary 5.7. The following conditions are equivalent.

1. E is solvable.
2. E has an sp-proper typing.
3. E has an sp-proper first-order typing.

Proof. By 5.6 (1) implies (3). (3) implies (2) trivially. (2) implies (1) by 4.3. ■

Lemma 5.8. If E is normal then there is a typing $C \vdash E:\tau$ derived using proper first-order types only.

Proof. By induction on E. The case E is a variable is trivial.

Suppose $E \equiv \lambda x.F$. By induction assumption there is a derivation Δ, using proper first-order types only, for the typing $C,x:\sigma \vdash F:\rho$. Applying $\rightarrow I$ we get such a derivation for $C \vdash \lambda x.F:\sigma \rightarrow \rho$.

Suppose $E \equiv zF_1 \cdots F_k$, z a λ-variable. By induction assumption there are proper first-order derivations Δ_i deriving typings $C_i \vdash F_i:\sigma_i$, $i=1 \cdots k$. Define a context C by $C(x)=C_1(x) \wedge \cdots \wedge C_k(x)$ for x other than z, $C(z)=C_1(z) \wedge \cdots \wedge C_k(z) \wedge (\sigma_1 \rightarrow \cdots \rightarrow \sigma_k \rightarrow R)$, where R is fresh. By lemma 5.2 there are proper first-order derivations deriving $C \vdash F_i:\sigma_i$, $i=1 \cdots k$. Applying $\wedge E$ and $\rightarrow E$ yields a derivation of $C \vdash E:R$ ■

Theorem 5.9. If a λ-expression E is strongly normalizable then it has a first-order properly derived typing.

Proof. By induction on the reduction tree of E. The induction basis is lemma 5.8. The induction step is lemma 5.4. ■

Corollary 5.10. The following conditions are equivalent.

1. E is strongly normalizable.
2. E has a properly derived typing.
3. E has a typing derived by a proper first-order derivation.

Proof. By 5.8 (1) implies (3). (3) implies (2) trivially. (2) implies (1) by 4.1. ■

Theorem 5.11. [CDV81]. If a λ-expression E is normalizable then it has a first-order derived p-proper typing.

Proof. By induction on the length of the shortest β-conversion leading from E to an expression in normal form. The induction basis is lemma 5.8. The induction step is lemma 5.4. ■

Corollary 5.12. The following conditions are equivalent.

1. E is normalizable.
2. E has a first-order derived p-proper typing.
3. E has a first-order derived proper typing.

Proof. By 5.11 (1) implies (3). (3) implies (2) trivially. (2) implies (1) by 4.7. ■

6. Strong normalization in the ontological discipline.

Let us now see that every expression of the Second-Order Lambda Calculus is strongly normalizable, as a corollary of the analogous result for the untyped calculus (theorem 4.1). The idea is simply this. On the one hand, every typed expression E can be viewed as an untyped expression E^u decorated with its own proper type inference (Proposition 6.1). On the other hand, a sequence S of reductions on a typed expression E is isomorphic to a sequence of reductions on E^u, possibly with intermittent stretches of type reductions. The first one is finite by 4.1, and the latter ones are finite because type reductions shorten the syntax tree of a typed λ-expression. Thus S must be finite.

The Second-Order Lambda Calculus is defined in [Rey74, Gir72, FLO83, Lei83a]. In this calculus, like in any other typed version of the Lambda Calculus, all expressions (and subexpressions) carry their types, and an expression contains, in fact, its type derivation. In particular, each expression carries exactly *one* type, excluding thereby use of ω, or of type-conjunction. Since the type derivation of an expression is in fact contained in it, a type-derivation for an expression is merely a reconstruction of the typing information from the structure of the expression. This feature is best illustrated by the treatment of type abstraction, for which the inference rules are analogous to the rules $\rightarrow I$ and $\rightarrow E$ for λ-abstraction and application:

$$\frac{\vdash E:\sigma}{\vdash \Lambda R.E:\Delta R\,\sigma} \qquad \frac{\vdash E:\Delta R\,\sigma}{\vdash E\rho:\sigma[\rho/R]} \quad (\rho \text{ free for } R \text{ in } \sigma)$$

Using our terminology from §1, all type-inference rules for the Typed Calculus are thus structural, none is stationary.

Clearly, from the combinatorial viewpoint of type deduction, the type inference system obtained here is merely a notational variant of the system described in §1 for the semantical discipline, restricted to its proper and ∧-free fragment. For an expression E of the Second-Order Lambda Calculus, let E^u be the underlying untyped expression. That is, E^u is defined inductively by

- $(x_i^\tau)^u \equiv x_{\tau,i}$,
- $(\lambda x_i^\sigma.E^\rho)^u \equiv \lambda x_{\sigma,i}.E^u$,
- $(EF)^u \equiv E^u F^u$,
- $(\Lambda R.E)^u \equiv E^u$,
- $(E\tau)^u \equiv E^u$.

The following proposition follows immediately from the definition of the inference rules.

Proposition 6.1. [Lei83a]. Let E be an expression of the Second-Order Lambda Calculus. The untyped λ-expression E^u has a proper and ∧-free typing.■

Lemma 6.2. Let $E \equiv (\lambda x^\sigma G^\rho)H^\sigma$ be an expression of the Second-Order Lambda Calculus, $F \equiv G[H/x]$. Then E^u β-converts to F^u (in the untyped calculus).

Proof. Trivial from the definition of the mapping $E \Rightarrow E^u$.■

Lemma 6.3. Let E be an expression of the Second-Order Lambda Calculus, and suppose that E β-converts (in one step) to F. Then E^u β-converts to F^u.

Proof. By induction on the depth in E of the converted redex, using 6.2 for the induction basis.■

For an expression E of the Second-Order Lambda Calculus let αE be the number of object-applications and type-applications in E. That is, αE is defined inductively by

- $\alpha(x_i^\tau) \equiv 0$,
- $\alpha(\lambda x^\sigma.E^\rho) \equiv \alpha(E^\rho)$
- $\alpha(EF) \equiv \alpha E + \alpha F + 1$,
- $\alpha(\Lambda R.E) \equiv \alpha E$,
- $\alpha(E\tau) \equiv \alpha E + 1$.

Recall that in the second-order calculus one has type-β-conversions: $(\Lambda R.E^\tau)\sigma$ converts to $E^\tau[\sigma/R]$.

Lemma 6.4. Let E be an expression of the Second-Order Lambda Calculus. If E converts to F by a type-β-reduction, then $\alpha F = \alpha E - 1$.

Proof. By induction on the depth of the converted redex in E. The induction basis in proved by induction on expressions.■

One last trivial lemma is:

Lemma 6.5. If E converts to F by a type-β-conversion then $F^u \equiv E^u$.

Proof. As for 6.4.■

We can now put the pieces together:

Theorem 6.6. [Gir71,72] Every expression E of the Second-Order Lambda Calculus is strongly normalizable (within that calculus).

Proof. Let E be an expression of the Second-Order Lambda-Calculus. Let E^u be defined as above. By 6.1 and 4.1 E^u is strongly normalizable. The theorem is proved by (main) induction on the reduction tree of E^u, and secondary induction on αE.

Suppose E converts to F. If this is a λ-conversion, then E^u β-converts to F^u, by 6.3, so by induction assumption F is strongly normalizable. If the conversion is a type-β-conversion, then $F^u \equiv E^u$ by 6.5, and $\alpha F < \alpha E$, by 6.4. So F is strongly normalizable by the secondary induction assumption.

Since every expression to which E converts is strongly normalizable, then E must be strongly normalizable.■

References.

[And71] Peter B. Andrews, "Resolution in type theory," *Journal of Symbolic Logic 36* (1971) 414-432.

[Bar81] Henk Barendregt, *The Lambda Calculus*, North Holland, Amsterdam, 1981, xiv+615pp.

[BCD83] Henk Barendregt, Mario Coppo and Mariangiola Dezani-Ciancaglini, "A filter lambda-model and the completeness of type-assignment," *Journal of Symbolic Logic 48* (1983) 931-940.

[BM84] Kim B. Bruce and Albert Meyer, "The denotational semantics of second-order polymorphic lambda calculus"; manuscript, 1984 (to be presented at the International Symposium on Semantics of Data Types, Antibes, France, June 1984).

[CD80] Mario Coppo and Mariangiola Dezani-Ciancaglini, "An extension of basic functionality theory for λ-calculus," *Notre-Dame Journal of Formal Logic 21* (1980) 685-693.

[CDV81] Mario Coppo, Mariangiola Dezani-Ciancaglini and B. Veneri, "Functional character of solvable terms," *Zeitschr. f. math. Logik und Grundlagen d. Math 27* (1981) 45-58.

[CF58] Haskell B. Curry and Robert Feys, *Combinatory Logic*, North-Holland, Amsterdam-New York-Oxford, 1958.

[Chu40] Alonzo Church, "A formulation of the simple theory of types," *Journal of Symbolic Logic 5* (1940), pp 56-68.

[Cop80] Mario Coppo, "An extended polymorphic type system for applicative languages"; in P. Dembinski (ed.), *Mathematical Foundations of Computer Science*, Springer (LNCS #88), Berlin (1980) 194-204.

[deB70] N.G. de Bruijn, "The mathematical language AUTOMATH, its usage and some of its extensions"; in *Symposium on Automatic Demonstration*, Springer (LNM #125), Berlin (1970) 29-61.

[Fen71] Jens Eric Fenstad (ed.), *Proceedings of the Second Scandinavian Logic Symposium*, North-Holland, Amsterdam (1971) vii+405 pp.

[FLO83] Steven Fortune, Daniel Leivant, and Michael O'Donnell, "The expressiveness of simple and second order type structures," *Journal of the ACM 30* (1983), pp 151-185.

[Fri73] Harvey Friedman, "Some applications of Kleene's method for intuitionistic systems"; in H. Rogers and A.R.D. Mathias (eds.), *Cambridge Summer School in Mathematical Logic*, Springer (LNM #337), Berlin (1973) 113-170.

[Gir71] Jean-Yves Girard, "Une extension de l'interprétation de Godel a l'analyse, et son application à l'élimination des coupures dans l'analyse et dans la théorie des types"; in [Fen71] 63-92.

[Gir72] Jean-Yves Girard, *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre superieur*, Thése de Doctorat d'Etat, 1972, Paris.

[How80] William A. Howard, "The formulae-as-types notion of construction"; in [SH80] 479-490.

[Lei81] Daniel Leivant, "The complexity of parameter passing in polymorphic procedures," *Proceedings of the Thirteenth Annual Symposium on Theory of computing (1981) 38-45.*

[Lei83a] Daniel Leivant, "Polymorphic type inference"; *Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages* (1983) 88-98.

[Lei83b] Daniel Leivant, "Reasoning about functional programs and complexity classes associated with type disciplines"; *Twenty-fourth Annual Symposium on Foundations of Computer Science* (1983) 460-469.

[Mar71] Per Martin-Lof, "Hauptsatz for the theory of species"; in [Fen71] 217-234.

[Mar82] Per Martin-Lof, "Constructive mathematics and computer programming"; in L.J. Cohen et als. (editors), *Logic, Methodology and Philosophy of Science VI*, North-Holland, Amsterdam (1982).

[Mor68] James Morris, *Lambda Calculus Models of Programming Languages*, PhD thesis, MIT, Cambridge (Massachusetts), 1968.

[MPS84] David B. MacQueen, Gordon Plotkin and Ravi Sethi, "An ideal model for recursive polymorphic types," *Conference Record of the Eleventh Annual ACM Symposium on Principles of Programming Languages* (1984) 165-174.

[MS82] David B. MacQueen and Ravi Sethi, "A semantic model of types for applicative languages"; *ACM Symposium on LISP and Functional Programming*, 1982, 243-252.

[Pot80] Garrel Pottinger, "A type assignment to the strongly normalizable terms"; in [SH80] 561-578.

[Pra67] Dag Prawitz, "Completeness and Hauptsatz for second order logic," *Theoria 33* (1967) 246-258.

[Pra68] Dag Prawitz, "Hauptsatz for higher order logic," *Journal of Symbolic Logic 33* (1968) 452-457.

[Pra71] Dag Prawitz, "Ideas and results in proof theory"; in [Fen71] 235-308.

[Rey74] John C. Reynolds, "Towards a theory of type structures," in *Programming Symposium (Colloque sur la Programmation, Paris)*, Springer (LNCS #19), Berlin (1974) 408-425.

[San67] L.E. Sanchis, "Functionals defined by recursion," *Notre-Dame Journal of Formal Logic 8* (1967) 161-174.

[SH80] J.P. Seldin and J.R. Hindley (editors), *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, London, (1980) 606pp.

[Ste72] Soren Stenlund, *Combinators, λ-Terms and Proof Theory*, Reidel, Dordrecht (1972) 184pp.

[Tai66] W.W. Tait, "A non-constructive proof of Gentzen's Hauptsatz for second order predicate logic," *Bulletin of the American Mathematical Society 72* (1966) 980-983.

[Tai67] W.W. Tait, "Intensional interpretation of functionals of finite type," *Journal of Symbolic Logic 32* (1967) 198-212.

[Tai75] W.W. Tait, "A realizability interpretation of the theory of species"; in R. Parikh (ed.), *Logic Colloquium*, Springer (LNM #453), Berlin (1975) 240-251.

[Tak67] Moto-O Takahashi, "A proof of cut-elimination theorem in simple type theory," *Journal of the Mathematical Society of Japan 19* (1967) 399-410.