Theoretical Computer Science 44 (1986) 51-68
North-Holland

TYPING AND COMPUTATIONAL PROPERTIES OF LAMBDA EXPRESSIONS*

Daniel LEIVANT
Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213, U.S.A.

Communicated by M. Nivat
Received December 1984
Revised April 1985

Abstract. We use a perception of second-order typing in the λ-Calculus, as conveying semantic properties of expressions in models over λ-expressions, to exhibit natural and uniform proofs of theorems of Girard (1971/72) and of Coppo, Dezani and Veneri (1981) about the relations between typing properties and computational properties of λ-expressions (solvability, normalizability, strong normalizability), and of some generalizations of these theorems.

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 structured programming, in particular, to guarantee that the composition of procedures is meaningful. A reasonable expectation is that typing excludes improper computational behavior such as infinite regression of nonrecursive procedure calls.

The relation between typing and computational properties of nonrecursive procedure calls can be encapsulated in the typed lambda calculus that results from considering the procedure mechanism and the typing discipline of the programming language in hand. The question is, then, what convergence properties of λ-expressions are guaranteed by particular type disciplines. We consider two target forms for convergence: normal and head-normal λ-epxressions. A λ-expression is *normal* if it cannot be further reduced. It is *head-normal* if it is of the form $\lambda\bar{x}.z\bar{E}$ (i.e., $\lambda x_1 \ldots \lambda x_k.zE_1 \ldots E_m$). There arise three convergence properties of interest. The strongest of these properties is strong normalization: an expression E is *strongly normalizable* if every reduction sequence starting with E terminates (with a normal expression). The expression E is *normalizable* if there merely exists some reduction

* Research partially supported by ONR Grant N00014-84-K-0415 and by DARPA Grant F33615-81-K-1539.

sequence that converts E to a normal expression. Finally, E is *solvable* if there is a reduction sequence that converts E to a head-normal expression[1] (the equivalence of this definition to the original definition of solvability is due to Wadsworth [37, 3]). Clearly, every strongly-normalizable expression is normalizable, and every normalizable expression is solvable. The rationale for considering solvable expressions is that they have natural computational and semantical characterizations which make them, in suitable senses, the only 'meaningful' expressions [3].

Given a property P of λ-expressions, e.g., 'normalizable', we shall say that a typed λ-calculus *is* P if every expression thereof is P.

Programming languages such as PASCAL and ALGOL68 are based on relatively simple type disciplines, and their procedure call mechanism is represented by the simply typed lambda calculus or by fragments thereof [22, 12 (Section 4)]. It is not hard to see that the simply typed lambda calculus is strongly normalizable [33, 22, 30, 1, 12 (Subsection 6.1)].

A new dimension in typing was introduced by the invention of second-order and higher-order type disciplines [14, 15, 29]. From the programming language viewpoint, generic (parametric) types are motivated by the wish to avoid repeated definition of one procedure for arguments of different types. Once generic procedures are used, conceptual consistency dictates that they be also 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

There is no 'simple' proof that the Second-Order Lambda Calculus is normalizable: no such proof can be given in Second-Order Arithmetic, since the normalizability of the Second-Order Lambda Calculus implies the consistency of Second-Order Arithmetic [18]. A combinatorially isomorphic problem, that of 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 [33, 36, 26, 27] whereby every provable formula of Second-Order Logic has a normal (natural deduction) proof. The Strong Normalization Theorem for Second-Order Logic was finally proved by Girard [14, 15] by injecting second-order reasoning into the combinatorial method of Tait [34]. (Independently, Friedman injected second-order reasoning into a different syntactic method, Kleene's slash, with related results [13].) Girard's method was then applied and refined, in different guises, to various formalisms [28, 21, 32, 35, 12].

[1] We do not consider λ-expressions that convert to a head-normal form by any reduction sequence (a natural analog to strong normalizability), since these expressions do not have natural computational or semantical characterizations.

Two perceptions of typing

The observation that typed expressions convert to normal form goes back, for Combinatory Logic, to Curry and Feys [7]. However, they were thinking of a type not as an inherent property of objects, but as a semantic notion, so that one expression can have different types. This *semantical* perception of typing has been pursued and developed, for example, in [9, 25, 6, 24, 3, 23, 4]. 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 typed calculus, each object carries its type, a perception of types we dub *ontological.* To say that "*E is of type* $\tau \rightarrow \sigma$" implies, under the ontological interpretation, that the domain of E consists precisely of all objects of type τ. Under the semantical interpretation, 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 relation to Proof Theory. In Church's Theory of Types [8], 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 [7, 17, 10].

In spite of the fundamental differences between the ontological and the semantical viewpoints of typing, the computational aspects of the two disciplines are closely related (see Section 6 of this paper and [19]). A typing calculus for the ontological discipline is in fact isomorphically embeddable in a typing calculus for the semantical discipline.

Characterization of computational properties by typing conditions

Within the semantical discipline it makes sense to consider implications that are inverse to the Normalization Theorems. One sets forth a broader notion of typing, and considers the question of whether a certain computational property of λ-expressions, such as solvability or normalizability, implies typability. Questions of this kind were studied by Coppo, Dezani and Veneri [6]. They showed that for a suitable notion of type, 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 paper 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 [32, 34], Girard [14, 15], and Coppo, Dezani and Pottinger [9, 5, 25, 6]. We discuss the relations between typing and convergence within the semantical discipline. The novelty here is in considering types as semantical properties of λ-expressions in second-order models over λ-expressions. The techniques of Tait and Girard then factorize into natural steps: semantic validity of type inference for the models considered, construction of particular models, and closure properties of models.

Working in the semantic discipline permits a richer notion of type and therefore, more general results. This choice is fecund, however, even if one is interested only in ontological typing, since the combinatorial aspects of the proofs are more transparent and uniform when carried out for the semantic discipline, while the results can be effortlessly transported to the ontological discipline (see Section 6).

The connections between typing and convergence properties are summarized in Theorems 5.5, 5.9, and 5.14 below. Some new results are analogues and generalizations of the results of [6]. For example, 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. This suggests that strong normalizability and solvability are properties which are more 'stable' than normalizability.

1. Polymorphic typing

Polymorphic (i.e. second-order) types are defined inductively as follows (compare [29, 15, 24, 12]):

$R \in \mathbf{TV}$ —type variables (a countable set),

$\tau \in \mathbf{T}$ —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. An occurrence of a type expression σ in τ is *positive* (*negative*) if it is in the negative (i.e. left-side) scope of an even (odd, respectively) number of instances of $\to$ in τ. The occurrence is *strictly positive* 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). We denote the set of proper types by $\mathbf{T}_{\mathrm{pr}}$ and the set of first-order types by $\mathbf{T}_{\mathrm{fo}}$. Also, $\mathbf{T}_{\mathrm{prfo}} \equiv \mathbf{T}_{\mathrm{pr}} \cap \mathbf{T}_{\mathrm{fo}}$.

A *type statement* is an expression $E : \tau$, where E is a λ-expression, and τ is a type expression. Following De Bruijn [10], 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. Thus, $C[x_i] = \tau_i$. A *typing* is an expression $C \vdash E : \tau$, where C is a context, and $E : \tau$ is a type statement.

A typing $C \vdash E : \tau$ 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.

We can now define a type-inference formalism (compare [12, 24]). A 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 $\to I$ and $\to E$ (I for 'introduction', E for 'elimination'):

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

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

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

$$\frac{C \vdash E:\sigma}{C \vdash E:\forall R\sigma} \quad (R \text{ not free in } C),$$

$$\frac{C \vdash E:\forall R\sigma}{C \vdash E:\sigma[\rho/R]} \quad (\rho \text{ free for } R \text{ in } \sigma).$$

In the rule $\forall E$ the substituted type expression ρ is the *eigentype* of the inference. A derivation is *in* $\mathbf{T}_0 \subseteq \mathbf{T}$ if all types therein are in $\mathbf{T}_0$. A derivation is *essentially in* $\mathbf{T}_0$ if all eigentypes therein are in $\mathbf{T}_0$.

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:

$\mathbf{\Lambda} \equiv$ the set of all λ-expressions,

$\mathbf{\Lambda}_I \equiv$ 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 consider a monadic second-order language L, with the following nonlogical constants:

- for each λ-expression E and each (possibly empty) list $\xi = x_1 \ldots x_k$ of distinct λ-variables a k-ary function letter $\underline{E}_\xi$;
- a binary function letter $*$, which we use in infix.

We use $X, \ldots$ as individual logical variables, and $R, \ldots$ as unary predicate variables. In the kind of models we shall consider momentarily, the former will range over λ-expressions, and the latter over certain sets of λ-expressions.

We define a family of L-formulas φ^τ, $\tau \in \mathbf{T}$, by induction on τ. $\varphi^\tau[E]$ renders that the λ-expression E behaves functionally as a mapping of type τ.

$$\varphi^R[X] \equiv R(X), \qquad \varphi^\omega[X] \equiv \mathbf{true},$$

$$\varphi^{\sigma \to \rho}[X] \equiv \forall Y(\varphi^\sigma[Y] \to \varphi^\rho[X * Y]),$$

$$\varphi^{\sigma \wedge \rho}[X] \equiv \varphi^\sigma[X] \wedge \varphi^\rho[X], \qquad \varphi^{\forall R.\sigma}[X] \equiv \forall R \varphi^\sigma[X],$$

We write $\varphi[\psi/R]$ for the result of simultaneously substituting the formula ψ for all occurrences of the predicate letter R in φ. For our purpose it suffices to consider monadic predicate letters R, and formulas ψ with exactly one free individual variable, for which the formal definition of this substitution is trivial.

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

Proof. The proof is straightforward by induction on σ. $\square$

Remark. The mapping $\sigma \Rightarrow \varphi^\sigma$ is right inverse to the mapping $\varphi \Rightarrow \tau(\varphi)$ of [20], i.e., $\tau(\varphi^\sigma) \equiv \sigma$. We have here a duality between the (modified) notion of 'formula as type' [17, 20], and of the notion above of 'type as formula'.

We shall need to refer to a closure property of sets $\subseteq \mathbf{\Lambda}$. We say that a set $\mathbf{Q}$ is *extensional in* $\mathbf{U}$ if it is closed under $=_\beta$ with respect to $\mathbf{U}$, i.e.,

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

We say that $\mathbf{Q}$ is simply *extensional* if $\mathbf{Q}$ is extensional in itself. For example, $\mathbf{\Lambda}$ is extensional in any $\mathbf{Q} \subseteq \mathbf{\Lambda}$; $\mathbf{N}$ and $\mathbf{L}$ are extensional in $\mathbf{\Lambda}$; $\mathbf{S}$ is extensional, but not extensional in $\mathbf{\Lambda}$; $\mathbf{\Lambda}_I$ is not extensional in $\mathbf{\Lambda}$; $\mathbf{N}_I$ is extensional in $\mathbf{\Lambda}_I$, but not in $\mathbf{\Lambda}$.

In the rest of this paper we shall use the expression *model* for monadic Henkin models over $\mathbf{\Lambda}$ [16], with the following properties:

(1) the universe of individuals is $\mathbf{\Lambda}$;

(2) the universe of (monadic) predicates (i.e. sets) is a nonempty collection $\mathbf{C}$ of subsets of $\mathbf{\Lambda}$, each extensional in $\mathbf{U} \equiv \bigcup \mathbf{C}$;

(3) for each λ-expression E and each list $\xi \equiv x_1 \ldots x_k$ of λ-variables, $\underline{E}_\xi$ is interpreted as the k-ary function over Λ defined by $\underline{E}_\xi F_1 \ldots F_k \equiv E[F_1/x_1 \ldots F_k/x_k]$ (assuming renaming of bound variables in E as needed to make the substitution legal); $*$ is interpreted as application-formation, i.e. $E * F$ is interpreted as the λ-expression EF;

Thus, a model is completely determined by a collection $\mathbf{C}$ of subsets of Λ. We use $\mathbf{C}$ to denote the model determined by $\mathbf{C}$.

Suppose $\mathbf{C}$ is a model, φ an L-formula. We give the standard meaning to the semantic satisfaction relation $\mathbf{C}, \eta \models \varphi$, where η is a valuation of individual (logical) variables (occurring free in φ) as λ-expressions, and of predicate variables as elements of $\mathbf{C}$.

We write $[\tau]_{\mathbf{C}\eta}$ for $\{E \in \Lambda \mid \mathbf{C}, \eta \models \varphi^\tau[\underline{E}]\}$. When $\mathbf{C}$ is clear from the context, we use the abbreviations $\eta \models \varphi$ and $[\tau]_\eta$.

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

Proof. The proof is straightforward by induction on τ. □

Let $\mathbf{T}_0$ be a set of types. A model $\mathbf{C}$ is *complete (for* $\mathbf{T}_0$) if $[\tau]_{\mathbf{C}\eta} \in \mathbf{C}$ for every τ (in $\mathbf{T}_0$) and every valuation η in $\mathbf{C}$.

Lemma 2.3 (Soundness of type inference). *Suppose the typing*

$$(*) \qquad \{x_j : \sigma_i\}_{i=1,\ldots,k} \vdash E\,;\,\tau$$

is derived in $\mathbf{T}_0$. *Then*

$$\mathbf{C}, \eta \models \textstyle\bigwedge_i \varphi^{\sigma_i}(X_i) \to \varphi^\tau(\underline{E}_\xi \vec{X})$$

(*where* $\xi \equiv x_1 \ldots x_k$) *for every model* $\mathbf{C}$ *which is complete for* $\mathbf{T}_0$, *and every valuation* η *therein.*

Proof. The lemma is proved by induction on the derivation Δ of $(*)$.

If Δ consists of an initial typing then either $E \equiv x_i$ or $\tau \equiv \omega$, and the lemma is trivial.

Suppose that the last inference of Δ is $\to I$,

$$\frac{C, y : \sigma \vdash E : \rho}{C \vdash \lambda y.E : \sigma \to \rho}.$$

By the induction assumption,

$$\eta \models \textstyle\bigwedge_i \varphi^{\sigma_i}(X_i) \to \forall Y(\varphi^\sigma(Y) \to \varphi^\rho(\underline{E}_{\xi,y}\vec{X}Y)).$$

By Lemma 2.2, $[\rho]_\eta$ is extensional in $\bigcup \mathbf{C}$. If $\eta \models \varphi^\sigma(Y)$, then $G \in \bigcup \mathbf{C}$, where

$G \equiv \eta Y$, and so,

$$E[\bar{F}/\bar{x}, G/y] \in [\rho]_\eta \text{ implies } (\lambda y E[\bar{F}])G \in [\rho]_\eta.$$

I.e., $\eta \models \varphi^\rho(\underline{E}_{\xi,y}\bar{X}Y) \to \varphi^\rho(\underline{\lambda y.E}_\xi\bar{X} * Y)$. So, by the definition of $\varphi^{\sigma\to\rho}$,

$$\eta \models \bigwedge_i \varphi^{\sigma_i}(X_i) \to \varphi^{\sigma\to\rho}(\underline{\lambda y.E}_\xi\bar{X}).$$

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

$$\eta \models \bigwedge_i \varphi^{\sigma_i}(X_i) \to \varphi^{\sigma\to\rho}(\underline{E}_\xi\bar{X}) \quad\text{and}\quad \eta \models \bigwedge_i \varphi^{\sigma_i}(X_i) \to \varphi^{\sigma}(\underline{F}_\xi\bar{X}).$$

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

$$\eta \models \bigwedge_i \varphi^{\sigma_i}(X_i) \to \varphi^{\rho}(\underline{E}_\xi\bar{X} * \underline{F}_\xi\bar{X}).$$

But $\underline{E}_\xi\bar{X} * \underline{F}_\xi\bar{X} \equiv \underline{EF}_\xi\bar{X}$, so

$$\eta \models \bigwedge_i \varphi^{\sigma_i}(X_i) \to \varphi^{\rho}(\underline{EF}_\xi\bar{X}).$$

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

$$\eta \models \bigwedge_i \varphi^{\sigma_i}(X_i) \to \varphi^{\forall R\sigma}(\underline{E}_\xi\bar{X}).$$

By the definition of $\varphi^{\forall R\sigma}$ this implies

$$\eta[\mathbf{Q}/R] \models \bigwedge_i \varphi^{\sigma_i}(X_i) \to \varphi^{\sigma}(\underline{E}_\xi\bar{X})$$

for every $\mathbf{Q} \in \mathbf{C}$. But $[\rho]_\eta \in \mathbf{C}$, since $\mathbf{C}$ is complete, hence,

$$\eta \models \forall R \varphi^\sigma(\underline{E}_\xi\bar{X}) \to \varphi^\sigma(\underline{E}_\xi\bar{X})[\varphi^\rho/R].$$

Moreover, by Lemma 2.1, $\varphi^\sigma[\varphi^\rho/R] \equiv \varphi^{\sigma[\rho/R]}$. Altogether we get

$$\eta \models \bigwedge_i \varphi^{\sigma_i}(X_i) \to \varphi^{\sigma[\rho/R]}(\underline{E}_\xi\bar{X}).$$

The cases for the rules $\forall I$, $\wedge I$, and $\wedge E$ are straightforward. □

In fact, the proof establishes a slightly stronger result, which is formulated in the following lemma.

Lemma 2.4. *Suppose that Δ is a derivation of the typing $\{x_i : \sigma_i\}_{i=1,\dots,k} \vdash E : \tau$. If $\mathbf{C}$ is a model complete for the set of eigentypes in Δ, then $\bigwedge_i \varphi^{\sigma_i}(X_i) \to \varphi^\tau(\underline{E}_\xi\bar{X})$ (where $\xi \equiv x_1 \dots x_k$) is valid in $\mathbf{C}$.*

3. Model constructions

We describe two model constructions. The first construction is trivial, and will be used in the proof of Theorem 4.7 below.

Lemma 3.1. *Suppose that $\mathbf{U} \subseteq \mathbf{\Lambda}$ and $\mathbf{V} \subseteq \mathbf{U}$ are extensional in $\mathbf{U}$. Then $\{\mathbf{V}, \mathbf{U}\}$ is a model.*

For example, $\{\mathbf{N}, \mathbf{\Lambda}\}$ and $\{\mathbf{L}, \mathbf{\Lambda}\}$ are models.

The second construction yields complete models, which we use below in the proofs of Theorems 4.1 and 4.3. Let $\mathbf{U} \subseteq \mathbf{\Lambda}$, z a λ-variable. Let $z\mathbf{U}^* \equiv \{zE_1 \dots E_k \mid E_i \in \mathbf{U}, k \geq 0\}$. Define

$$\mathbf{C}_\mathbf{U} \equiv \{\mathbf{Q} \mid z\mathbf{U}^* \subseteq \mathbf{Q} \subseteq \mathbf{U}, \mathbf{Q} \text{ is extensional in } \mathbf{U}\}.$$

Lemma 3.2. *Suppose* $\mathbf{U}$ *is extensional. Assume that, for some* λ*-variable* z, $z\mathbf{U}^* \subseteq \mathbf{U}$ *and that* $E \in \mathbf{U}$ *whenever* $Ez \in \mathbf{U}$. *Then* $\mathbf{C}_\mathbf{U}$ *is a model complete for* $\mathbf{T}_{\text{pr}}$. *Moreover, if* $\mathbf{U} = \mathbf{\Lambda}$, *then* $\mathbf{C}_\mathbf{U}$ *is complete (for* $\mathbf{T}$*).*

Proof. The lemma's conditions guarantee that $\mathbf{U} \in \mathbf{C}_\mathbf{U}$, hence, $\mathbf{C}_\mathbf{U} \neq \emptyset$, so $\mathbf{C}_\mathbf{U}$ is a model. By Lemma 2.2, each $[\tau]_{\mathbf{C}_\mathbf{U}\eta}$ is extensional in $\mathbf{C}_\mathbf{U}$. It remains to show that $z\mathbf{U}^* \subseteq [\tau]_{\mathbf{C}_\mathbf{U}\eta} \subseteq \mathbf{U}$ for every valuation η in $\mathbf{C}_\mathbf{U}$ and every $\tau \in \mathbf{T}_{\text{pr}}$. We proceed by induction on τ. Let η be a valuation in $\mathbf{C}_\mathbf{U}$.

If τ is a type variable R, then $[\tau]_\eta = \eta(R) \in \mathbf{C}_\mathbf{U}$. The last clause of the lemma is trivial if $\tau \equiv \omega$.

Suppose $\tau \equiv \sigma \to \rho$. To show $z\mathbf{U}^* \subseteq [\tau]_\eta$, consider an arbitrary $z\bar{E} \in z\mathbf{U}^*$. Suppose $F \in [\sigma]_\eta$. By the induction assumption, $[\sigma]_\eta \subseteq \mathbf{U}$, and so, $z\bar{E}F \in z\mathbf{U}^*$. By the induction assumption $z\mathbf{U}^* \subseteq [\rho]_\eta$, so, $z\bar{E}F \in [\rho]_\eta$. We showed that

$$\eta \models \forall X(\varphi^\sigma(X) \to \varphi^\rho(\underline{z\bar{E}} * X)).$$

Thus, $z\bar{E} \in [\tau]_\eta$.

To show $[\tau]_\eta \subseteq \mathbf{U}$, assume $E \in [\tau]_\eta$. By the induction assumption, $z \in [\sigma]_\eta$, so $Ez \in [\rho]_\eta$. But by the induction assumption $[\rho]_\eta \subseteq \mathbf{U}$, hence $Ez \in \mathbf{U}$, which implies that $E \in \mathbf{U}$ by the lemma's assumptions.

If $\tau \equiv \forall R\sigma$ then we have

$$[\forall R\sigma]_\eta = \bigcap \{[\sigma]_\theta \mid \theta = \eta[\mathbf{Q}/R], \mathbf{Q} \in \mathbf{C}_\mathbf{U}\}.$$

By the induction assumptions, $z\mathbf{U}^* \subseteq [\sigma]_\theta \subseteq \mathbf{U}$ for every valuation θ. So, $z\mathbf{U}^* \subseteq [\forall R\sigma]_\eta \subseteq \mathbf{U}$.

The case for τ conjunctive is similar. □

Example. $\mathbf{\Lambda}$ and $\mathbf{S}$ satisfy the conditions for $\mathbf{U}$ in Lemma 3.2. Hence, $\mathbf{C}_\mathbf{\Lambda}$ is complete, and $\mathbf{C}_\mathbf{S}$ is complete for $\mathbf{T}_{\text{pr}}$.

4. Computational properties implied by typing

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

- every expression with a typing derived in $\mathbf{T}_{\text{pr}}$ is strongly normalizable;
- every expression with an sp-proper typing is solvable; and
- every expression with a p-proper typing derived in $\mathbf{T}_{\text{fo}}$ is normalizable.

4.1. *Strongly normalizable expressions*

The following theorem is essentially Girard's Theorem [14, 15], formulated for the semantical discipline (and with type conjunction as an additional type construct).

Theorem 4.1. *If E has a typing derived in* $\mathbf{T}_{\mathrm{pr}}$, *then E is strongly normalizable.*

Proof. Assume $C \vdash E : \tau$ is derived in $\mathbf{T}_{\mathrm{pr}}$. It suffices to consider closed expressions E, because $E \in \mathbf{S}$ iff $\lambda \vec{x}.E \in \mathbf{S}$. Assume the theorem's premise (with $C \equiv \emptyset$). Then, by Lemma 2.3, $E \in [\tau]_{\mathbf{C}\eta}$ for any complete model $\mathbf{C}$ and any valuation η therein. Thus, $E \in [\tau]_{\mathbf{C_S}\eta}$, by Lemma 3.2. By the definition of $\mathbf{C_S}$ and by Lemma 3.2, $[\tau]_{\mathbf{C_S}\eta} \subseteq \mathbf{S}$, where η is arbitrary, e.g., the constant valuation $\mathbf{S}$. It follows that $E \in \mathbf{S}$. □

Remark. The theorem fails when only the derived typing is required to be proper, but not the entire derivation, as can be seen from the following example.

Example. There is a derived typing $\vdash E : \tau$, where τ is proper but E is not even normalizable. Let Ω be a nonnormalizable expression. It is easy to see that

$$\vdash \lambda z.z\Omega : (\forall R.R) \to Q$$

(Q a type variable) is derivable, using $\Omega : \omega$ as an initial typing.

4.2. *Solvable expressions*

The model we use here is $\mathbf{C}_\Lambda$. Note that $\mathbf{L} \in \mathbf{C}_\Lambda$, since $\mathbf{L}$ is extensional in Λ, $z\Lambda^* \subset \mathbf{L}$.

Lemma 4.2. *Let ζ be the constant valuation* $\mathbf{L}$. *Suppose that τ is an sp-proper type. Then* $[\tau]_{\mathbf{C}_\Lambda \zeta} \subseteq \mathbf{L}$.

Proof. The lemma is proved by induction on sp-proper types. If τ is a type variable R, then $[\tau]_\zeta \equiv \zeta(R) \equiv \mathbf{L}$.

Suppose $\tau \equiv \sigma \to \rho$. To show $[\tau]_\zeta \subseteq \mathbf{L}$, assume $E \in [\tau]_\zeta$. Since, by Lemma 3.2, $\mathbf{C}_\Lambda$ is complete, $[\sigma]_\zeta \in \mathbf{C}_\Lambda$; hence, $z\Lambda^* \subseteq [\sigma]_\zeta$, and so $z \in [\sigma]_\zeta$ (z a λ-variable). Thus, $Ez \in [\rho]_\zeta$. Since τ is sp-proper, so must be ρ, and by the induction assumption we have $[\rho]_\zeta \subseteq \mathbf{L}$. Thus, $Ez \in \mathbf{L}$, and so, $E \in \mathbf{L}$.

Suppose $\tau \equiv \forall R.\sigma$. Then, $[\tau]_\zeta \equiv \bigcap \{[\sigma]_\theta \mid \theta \equiv \zeta[\mathbf{Q}/R],\ \mathbf{Q} \in \mathbf{C}_\Lambda\}$, which is $\subseteq [\sigma]_\zeta$ since ζ is one of the θ's. By the induction assumption $[\sigma]_\zeta \subseteq \mathbf{L}$, so $[\tau]_\zeta \subseteq \mathbf{L}$.

The case where τ is conjunctive is similar. □

The following theorem is proved in [6] for typings derived in $\mathbf{T}_{\mathrm{fo}}$.

Theorem 4.3. *If E has a derived 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 2.3, this implies that $E \in [\tau]_{\mathbf{C}\eta}$ for any complete model $\mathbf{C}$ and η therein. By Lemma 3.2 $\mathbf{C}_\Lambda$ is complete, and by Lemma 4.2 $[\tau]_{\mathbf{C}_\Lambda \zeta} \subseteq \mathbf{L}$, where ζ is the constant valuation $\mathbf{L}$. Hence $E \in \mathbf{L}$. □

4.3. Normalizable expressions

We show that if E has a p-proper typing (essentially) derived in $\mathbf{T}_{\mathrm{fo}}$, then E is normalizable. The model we use here is $\{\mathbf{N}, \Lambda\}$, which we denote by $\mathbf{M}$ for the rest of this section.

Lemma 4.4. *Let* $\tau \in \mathbf{T}_{\mathrm{prfo}}$. *Let* ζ *be the constant valuation* $\mathbf{N}$. *Then* $[\tau]_{\mathbf{M}\zeta} \subseteq \mathbf{N}$.

Proof. We prove, by induction on proper types, that $z\mathbf{N}^* \subseteq [\tau]_\zeta \subseteq \mathbf{N}$ (z a λ-variable).

If τ is a type variable R, then $[\tau]_\zeta \equiv \zeta(R) \equiv \mathbf{N}$.

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

To show $z\mathbf{N}^* \subseteq [\tau]_\zeta$, let $z\bar{E} \in z\mathbf{N}^*$. By the induction assumption, $[\sigma]_\zeta \subseteq \mathbf{N}$, so $z\bar{E}F \in z\mathbf{N}^*$ for every $F \in [\sigma]_\zeta$. By the induction assumption, $z\mathbf{N}^* \subseteq [\rho]_\zeta$, so $z\bar{E}F \in [\rho]_\zeta$. Thus, $z\bar{E} \in [\tau]_\zeta$.

The case where τ is conjunctive is trivial. □

Remark. Unlike in the proof of Lemma 4.2, we need the lower bound condition $z\mathbf{N}^* \subseteq [\tau]_\zeta$ in the proof of Lemma 4.4. This condition breaks down if $\tau \equiv \forall R.\sigma$, because ζ is held fixed (unlike in the proof of Lemma 3.2).

Lemma 4.5. *Let* ζ *and* η *be valuations in* $\mathbf{M}$, *which are identical 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. The lemma is proved 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. *Suppose* $\tau \in \mathbf{T}_{\mathrm{fo}}$ *is p-proper,* ζ *a valuation in* $\mathbf{M}$. *There is a type* $\theta \in \mathbf{T}_{\mathrm{prfo}}$ *such that* $[\tau]_\zeta \subseteq [\theta]_\zeta$.

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 Lemma 4.5. □

Theorem 4.7 (Coppo, Dezanni and Venneri [6]). *If* E *has a p-proper typing* (*essentially*) *derived in* $\mathbf{T}_{\mathrm{fo}}$, *then* E *is normalizable.*

Proof. It suffices to consider closed E. Suppose that $\vdash E : \tau$ has a derivation in $\mathbf{T}_{\mathrm{fo}}$, where τ is p-proper. By Lemmas 2.4 and 3.1 $E \in [\tau]_\eta$ for every valuation η in $\mathbf{M}$. In particular, $E \in [\tau]_\zeta$, where ζ is the constant valuation $\mathbf{N}$. By Lemma 4.6 $[\tau]_\zeta \subseteq [\theta]_\zeta$ for some $\theta \in \mathbf{T}_{\mathrm{prfo}}$, and by Lemma 4.4 $[\theta]_\zeta \subseteq \mathbf{N}$. So $E \in \mathbf{N}$. □

By the example above, the restriction in Theorem 4.7 to essentially first-order derivations is necessary. 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 (subtypes not being counted), then no initial typing $C \vdash E : \omega$ can occur in the derivation. All occurrences of ω can then be replaced by some fresh type variable, and all types become proper. This brings us back to the case of Theorem 4.1.

5. Computational properties implying typing; characterization theorems

All expressions trivially have the type ω. We show here that expressions satisfying certain computational properties can be assigned more interesting types [6]. Namely,

- every solvable expression E has an sp-proper typing derived in $\mathbf{T}_{\text{fo}}$;
- every normalizable expression has a p-proper typing derived in $\mathbf{T}_{\text{fo}}$;
- every strongly normalizable expression has a typing derived in $\mathbf{T}_{\text{prfo}}$.

5.1. Solvable expressions

Lemma 5.1. *If E is in head-normal form, then E has a sp-proper typing derived in* $\mathbf{T}_{\text{fo}}$.

Proof. Clearly, $z : \omega \to \omega \to \ldots \to \omega \to R \vdash zF_1 \ldots F_k : R$ is derived, where there are k occurrences of ω and R is a type variable. □

Lemma 5.2. *If* $C \vdash E[F/x] : \tau$ *is derived in* $\mathbf{T}_{\text{fo}}$, *then so is* $C \vdash (\lambda x.E)F : \tau$.

Proof. Suppose that Δ derives $C \vdash E[F/x] : \tau$. This implies, by induction on Δ, that $C, x : \omega \vdash E : \tau$ is derived in $\mathbf{T}_{\text{fo}}$. The lemma follows by $\to I$ and $\to E$. □

Lemma 5.3. *Suppose that E β-converts to F (in one step). If* $C \vdash F : \tau$ *is derived in* $\mathbf{T}_{\text{fo}}$, *then so is* $C \vdash E : \tau$.

Proof. The lemma is proved by induction on the depth of the conversion in E, using Lemma 5.2 for the base case. □

Theorem 5.4 ([6]). *If a λ-expression E is solvable, then it has an sp-proper typing derived in* $\mathbf{T}_{\text{fo}}$.

Proof. The theorem is proved 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 5.3. □

Corollary 5.5. *The following conditions are equivalent.*

(1) *E is solvable.*

(2) *E has a derived sp-proper typing.*

(3) *E has an sp-proper typing derived in* $\mathbf{T}_{\text{fo}}$.

Proof. By Theorem 5.4 (1) implies (3), (3) trivially implies (2), and (2) implies (1) by Theorem 4.3. □

5.2. Normalizable expressions

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 different.

Lemma 5.6. *Let* $\mathbf{T}_0 \subseteq \mathbf{T}$ *be closed under type conjunction. If* $C \vdash E : \tau$ *is derived in* $\mathbf{T}_0$, *and* C' *is a context in* $\mathbf{T}_0$, *then* $C \wedge C' \vdash E : \tau$ *is derived in* $\mathbf{T}_0$.

Proof. The lemma can be proved by induction on Δ. □

Lemma 5.7. *If E is normal then it has a typing derived in* $\mathbf{T}_{\text{prfo}}$.

Proof. The lemma is proved by induction on E. If E is a λ-variable then the lemma is trivial.

Suppose $E \equiv \lambda x.F$. By the induction assumption there is a derivation Δ in $\mathbf{T}_{\text{prfo}}$ for a 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 $E \equiv zF_1 \ldots F_k$, z a λ-variable. By the induction assumption there are derivations Δ_i in $\mathbf{T}_{\text{prfo}}$ deriving typings $C_i \vdash F_i : \sigma_i$, $i = 1, \ldots, k$. Define a context C by $C[x] = C_1[x] \wedge \cdots \wedge C_k[x]$ for any λ-variable 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 5.6 there are derivations in $\mathbf{T}_{\text{prfo}}$ for $C \vdash F_i : \sigma_i$, $i = 1, \ldots, k$. Applying k times $\wedge E$ and k times $\rightarrow E$ yields a derivation in $\mathbf{T}_{\text{prfo}}$ for $C \vdash E : R$. □

Theorem 5.8 ([6]). *If a λ-expression E is normalizable, then it has a p-proper typing derived in* $\mathbf{T}_{\text{fo}}$.

Proof. The theorem is proved by induction on the length of the shortest β-conversion leading from E to an expression in normal form. The induction basis is Lemma 5.7. The induction step is Lemma 5.3. □

Corollary 5.9. *The following conditions are equivalent.*

(1) *E is normalizable.*

(2) *E has a p-proper typing derived essentially in* $\mathbf{T}_{\text{fo}}$.

(3) *E has a p-proper typing derived in* $\mathbf{T}_{\text{fo}}$.

Proof. By Theorem 5.8 (1) implies (3); (3) trivially implies (2), and (2) implies (1) by Theorem 4.7. □

5.3. Strongly normalizable expressions

In this subsection let $\mathbf{T}_0 \subseteq \mathbf{T}$ be a set of types closed under type conjunction. The case we shall use is $\mathbf{T}_0 = \mathbf{T}_{\mathrm{prfo}}$.

Lemma 5.10. *Suppose that F is free for x in E. If $C \vdash E[F/x] : \tau$ is derived in $\mathbf{T}_0$, then either there is a σ such that both $C, x : \sigma \vdash E : \tau$ and $C \vdash F : \sigma$ are derived in $\mathbf{T}_0$, or x is not free in E and $C \vdash E : \tau$ is derived in $\mathbf{T}_0$.*

Proof. The lemma is proved by induction on the derivation Δ of $C \vdash E[F/x]$. The induction basis is trivial.

If the last inference of Δ is $\rightarrow E$ then $E \equiv GH$, and there are typings $C \vdash G : \rho \rightarrow \tau$ and $C \vdash H : \rho$ derived by the immediate subderivations of Δ. Suppose x is free in neither G nor H. Then, by the induction assumption, there are σ_1 and σ_2 for which the typings $C, x : \sigma_1 \vdash G : \rho \rightarrow \tau$, $C \vdash F : \sigma_1$, $C, x : \sigma_2 \vdash H : \rho$, and $C \vdash F : \sigma_2$ are all derived in $\mathbf{T}_0$. Setting $\sigma \equiv \sigma_1 \wedge \sigma_2$ we have $C, x : \sigma \vdash GH : \tau$ by Lemma 5.6 and $\rightarrow E$. And we have $C \vdash F : \sigma$ by $\wedge I$.

Other cases for x and other cases for Δ are similar. □

Lemma 5.11. *If $E[F/x]$ and F have typings derived in $\mathbf{T}_0$, then so does $(\lambda x E)F$.*

Proof. Suppose that $C_1 \vdash E[F/x] : \tau$ and $C_2 \vdash F : \rho$ are derived in $\mathbf{T}_0$. Let $C \equiv C_1 \wedge C_2$. If the first case of Lemma 5.10 holds (for some σ), then $C \vdash \lambda x E : \sigma \rightarrow \tau$ by Lemma 5.6, and $C \vdash (\lambda x E)F : \tau$ follows by $\rightarrow E$. If the second case of Lemma 5.10 holds, then $C \vdash \lambda x E : \rho \rightarrow \tau$, and $C \vdash (\lambda x E)F : \tau$ follows by $\rightarrow E$ and the lemma's assumptions. □

Lemma 5.12. *Suppose that E converts to F by replacing a subexpression $(\lambda x.G)H$ by $G[H/x]$. If both F and H have typings derived in $\mathbf{T}_0$, then so does E.*

Proof. The lemma is proved by induction on the depth of the replaced expression in E. The basis is Lemma 5.11. The induction step is trivial. □

Theorem 5.13. *If a λ-expression E is strongly normalizable, then it has a typing derived in $\mathbf{T}_{\mathrm{prfo}}$.*

Proof. The theorem is proved by induction on the conversion tree of E. If E is normal then the lemma holds by Lemma 5.7. If E converts to F by replacing $(\lambda x.G)H$ by $G[H/x]$, then both F and H have typings derived in $\mathbf{T}_{\mathrm{prfo}}$, by the induction assumption, so E has such a typing by Lemma 5.12. □

Corollary 5.14. *The following conditions are equivalent.*

(1) *E is strongly normalizable.*

(2) *E has a typing derived in* $\mathbf{T}_{\mathrm{pr}}$.

(3) *E has a typing derived in* $\mathbf{T}_{\mathrm{prfo}}$.

Proof. By Theorem 5.13 (1) implies (3), (3) trivially implies (2), and (2) implies (1) by Theorem 4.1. □

6. Strong normalization in the ontological discipline

We now consider the ontological Second-Order Lambda Calculus, and prove that it is strongly normalizable, as a corollary of the strong normalization theorem (Theorem 4.1) for the semantical discipline. The idea is simply this. Every typed expression E can be viewed as an untyped expression E^{u} decorated with its own type inference (Propostion 6.1). A sequence S of β-conversions on a typed expression E is isomorphic to a sequence of β-conversions on E^{u}, possibly with intermittent stretches of type-β-conversions. The first sequence is finite by Theorem 4.1, while each sequence of type conversions is finite because a type-β-conversion reduces the number of type applications in a typed λ-expression. Thus, S must be finite.

The Second-Order Lambda Calculus is defined in [29, 15, 12]. In this calculus, like in other typed versions of the Lambda Calculus, all expressions (and subexpressions) carry their type. A type-derivation for an expression is therefore merely a reconstruction of the typing information from the structure of the expression. This is illustrated by the modified inference rules $\to I$ and $\to E$ for λ-abstraction and application:

$$\frac{C \vdash E : \sigma}{C \vdash \Lambda R.E : \Delta R\sigma} \quad (R \text{ not free in } C),$$

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

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

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

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

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

Proposition 6.1. *Let E be an expression of the Second-Order Lambda Calculus. The untyped λ-expression E^{u} has a typing derived in proper and $\wedge$-free types.*

In the Second-Order Lambda Calculus one has two kinds of β-conversion: for abstraction over individual variables: $(\lambda x^{\sigma}.E^{\tau})F^{\sigma}$ converts to $E[F/x]$; and for abstraction over types: $(\Lambda R.E^{\tau})\sigma$ converts to $E[\sigma/R]$.

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

Proof. The proof of the lemma is trivial from the definition of the mapping $E \Rightarrow E^{\mathrm{u}}$. □

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. The lemma is proved by induction on the depth in E of the converted redex, using Lemma 6.2 for the induction basis. □

For an expression E of the Second-Order Lambda Calculus let αE be the number of type-applications in E.

Lemma 6.4. *Let E be an expression of the Second-Order Lambda Calculus. If E converts to F by a type-β-conversion, then*
(a) $F^{\mathrm{u}} \equiv E^{\mathrm{u}}$,
(b) $\alpha F = \alpha E - 1$.

Proof. The lemma is proved by induction on the depth of the converted redex in E. The induction basis is proved by induction on expressions. □

Theorem 6.5 ([14, 15]). *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 Lemma 6.1 and Theorem 4.1, E^{u} is strongly normalizable. We show that every F to which E converts is strongly normalizable by (main) induction on the conversion tree of E^{u} and secondary induction on αE.

Suppose that E converts to F. If this is an object β-conversion, then E^{u} β-converts to F^{u} by Lemma 6.3, so by the induction assumption F is strongly normalizable. If the conversion is a type-β-conversion, then $F^{\mathrm{u}} \equiv E^{\mathrm{u}}$ and $\alpha F < \alpha E$, by Lemma 6.4. So F is strongly normalizable by the secondary induction assumption.

Every expression to which E converts being strongly normalizable, it follows that E must be strongly normalizable. $\square$

References

[1] P.B. Andrews, Resolution in type theory, *J. Symbolic Logic* **36** (1971) 414–432.

[2] H. Barendregt, *The Lambda Calculus* (North-Holland, Amsterdam, 1981).

[3] H. Barendregt, M. Coppo and M. Dezani-Ciancaglini, A filter lambda-model and the completeness of type-assignment, *J. Symbolic Logic* **48** (1983) 931-940.

[4] K.B. Bruce and A.R. Meyer, The semantics of second-order polymorphic lambda calculus, in: G. Kahn, D.B. MacQueen and G. Plotkin, eds., *Semantics of Data Types*, Lecture Notes in Computer Science **173** (Springer, Berlin, 1984) 131-144.

[5] M. Coppo and M. Dezani-Ciancaglini, An extension of basic functionality theory for λ-calculus, *Notre-Dame J. Formal Logic* **21** (1980) 685-693.

[6] M. Coppo, M. Dezani-Ciancaglini and B. Veneri, Functional character of solvable terms, *Z. Math. Logik Grundlag. Math.* **27** (1981) 45-58.

[7] H.B. Curry and R. Feys, *Combinatory Logic* (North-Holland, Amsterdam, 1958).

[8] A. Church, A formulation of the simple theory of types, *J. Symbolic Logic* **5** (1940) 56-68.

[9] M. Coppo, An extended polymorphic type system for applicative languages, in: P. Dembinski, ed., *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science **88** (Springer, Berlin, 1980) 194-204.

[10] N.G. de Bruijn, The mathematical language AUTOMATH, its usage and some of its extensions, *Symposium on Automatic Demonstration*, Lecture Notes in Mathematics **125** (Springer, Berlin, 1970) 29-61.

[11] J.E. Fenstad, ed., *Proceedings of the Second Scandinavian Logic Symposium* (North-Holland, Amsterdam, 1971).

[12] S. Fortune, D. Leivant and M. O'Donnell, The expressiveness of simple and second order type structures, *J. ACM* **30** (1983) 151-185.

[13] H. 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*, Lecture Notes in Mathematics **337** (Springer, Berlin, 1973) 113-170.

[14] J.-Y. Girard, Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types, in: J.E. Fenstad, ed., *Proceedings of the Second Scandinavian Logic Symposium* (North-Holland, Amsterdam, 1971) 63-92.

[15] J.-Y. Girard, Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur, Thèse de Doctorat d'Etat, 1972, Paris.

[16] L. Henkin, Completeness in the theory of types, *J. Symbolic Logic* **15** (1950) 81-91.

[17] W.A. Howard, The formulae-as-types notion of construction, in: J.P. Seldin and J.R. Hindley, eds., *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism* (Academic Press, London, 1980) 479-490.

[18] D. Leivant, The complexity of parameter passing in polymorphic procedures, *Proc. 13th Ann. Symp. on Theory of Computing* (1981) 38-45.

[19] D. Leivant, Polymorphic type inference, *Conf. Record 10th Ann. ACM Symp. on Principles of Programming Languages* (1983) 88-98.

[20] D. Leivant, Reasoning about functional programs and complexity classes associated with type disciplines, *24th Ann. Symp. on Foundations of Computer Science* (1983) 460-469.

[21] P. Martin-Löf, Hauptsatz for the theory of species, in: J.E. Fenstad, ed., *Proceedings of the Second Scandinavian Logic Symposium* (North-Holland, Amsterdam, 1971) 217-234.

[22] J. Morris, Lambda calculus models of programming languages, Ph.D. Thesis, M.I.T., Cambridge, MA, 1968.

[23] D.B. MacQueen, G. Plotkin and R. Sethi, An ideal model for recursive polymorphic types, *Conf. Record 11th Ann. ACM Symp. on Principles of Programming Languages* (1984) 165-174.

[24] D.B. MacQueen and R. Sethi, A semantic model of types for applicative languages, *ACM Symp. on LISP and Functional Programming* (1982) 243-252.

[25] G. Pottinger, A type assignment to the strongly normalizable terms, in: J.P. Seldin and J.R. Hindley, eds., *To H.B. Curry*; *Essays on Combinatory Logic, Lambda Calculus and Formalism* (Academic Press, London, 1980) 561-578.

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

[27] D. Prawitz, Hauptsatz for higher order logic, *J. Symbolic Logic* **33** (1968) 452-457.

[28] D. Prawitz, Ideas and results in proof theory, in: J.E. Fenstad, ed., *Proceedings of the Second Scandinavian Logic Symposium* (North-Holland, Amsterdam, 1971) 235-308.

[29] J.C. Reynolds, Towards a theory of type structures, *Programming Symposium* (*Colloque sur la Programmation, Paris*), Lecture Notes in Computer Science **19** (Springer, Berlin, 1974) 408-425.

[30] L.E. Sanchis, Functionals defined by recursion, *Notre-Dame J. Formal Logic* **8** (1967) 161-174.

[31] J.P. Seldin and J.R. Hindley, eds., *To H.B. Curry*: *Essays on Combinatory Logic, Lambda Calculus and Formalism* (Açademic Press, London, 1980).

[32] S. Stenlund, *Combinators, λ-Terms and Proof Theory* (Reidel, Dordrecht, 1972).

[33] W.W. Tait, A non-constructive proof of Gentzen's Hauptsatz for second order predicate logic, *Bull. Amer. Math. Soc.* **72** (1966) 980-983.

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

[35] W.W. Tait, A realizability interpretation of the theory of species, in: R. Parikh, ed., *Logic Colloquium*, Lecture Notes in Mathematics **453** (Springer, Berlin, 1975) 240-251.

[36] Moto-O Takahashi, A proof of cut-elimination theorem in simple type theory, *J. Math. Soc. Japan* **19** (1967) 399-410.

[37] C.P. Wadsworth, The relation between computational and denotational properties of Scott's D_∞-model of the lambda-calculus, *SIAM J. Comput.* **5** (1976) 488-521.