Annals of Pure and Applied Logic 51 (1991) 159–172
North-Holland

Inductive types and type constraints in the second-order lambda calculus*

Nax Paul Mendler

Department of Computer Science, Manchester University, Manchester, UK M13 9PL

Communicated by A. Nerode

Abstract

Mendler, N.P., Inductive types and type constraints in the second-order lambda calculus, Annals of Pure and Applied Logic 51 (1991) 159–172.

We add to the second-order lambda calculus the type constructors μ and ν, which give the least and greatest solutions to positively defined type expressions. Strong normalizability of typed terms is shown using Girard's *candidat de réductibilité* method. Using the same structure built for that proof, we prove a necessary and sufficient condition for determining when a collection of equational type constraints admit the typing of only strongly normalizable terms.

1. Introduction

In the first half of this paper, we give an extension to the second-order lambda calculus [3, 5, 6, 9], which permits the definition of least and greatest solutions to positively defined type expressions using the type constructors μ and ν, respectively. With μ one can define inductive types such as the natural numbers, constructive ordinals, lists and trees, and there are inductive combinators available for each type; with ν, 'lazy' types such as streams and potentially infinite trees can be defined. The focus here is not model theory as in [1, 2, 4], but on the normalization property: the central result is a proof of strong normalizability (i.e., that every reduction sequence of a term is finite) using Girard's *candidat de réductibilité* method [6, 7], which we extend to exploit the fact that the collection of ground forms a complete lattice, and that inductive types can be viewed as the least and greatest fixed points of monotonic (but not necessarily continuous) operations on it.

Using the structures built in the first half of this paper, in the second half we consider typing terms in the presence of equational type constraints [2], which allow the typing of more terms. The problem statement:

* This research was partly supported by the National Science Foundation under grant MCS-81-04018.

Given a collection of equations constraints $\tau_i = T_i$ for i in a fixed index set I, where τ_i is a type constant and T_i is a type expression in which τ_i or other type constants may occur, and which are closed under reflexivity, symmetry, transitivity and substitution, and the type rule:

$$\frac{a:A \quad A=B}{a:B}.$$

Are the resulting typed terms strongly normalizable?

We give a decidable (for finite I) test for this. Thus we see, for example, that constraints:

$$\tau_0 = \tau_1 \to \tau_0, \qquad \tau_1 = \tau_0 \to \tau_1$$

will yield only strongly normalizable typed terms, while constraints:

$$\tau_0 = \tau_1 \to \tau_0, \qquad \tau_1 = \tau_2 \to \tau_1, \qquad \tau_2 = \tau_0 \to \tau_2$$

will allow a diverging term to be typed. Our test will be the absence of derivable type equalities $t_i = A$ where τ_i occurs negatively in A.

2. Inductive types

In this section we will prove the strong normalizability of terms in a second-order lambda calculus equipped with inductive types. First, the type expressions, terms and reduction are defined. Then the rest of the section consists of the proof of strong normalization, which we now outline.

(1) Let T be the set of strongly normalizable untyped terms. Ξ, a complete lattice of subsets of T, is defined. Types will be modeled by elements of Ξ.

(2) In Proposition 3, operations on Ξ that will model the type constructors $\to$, Δ, μ and ν are defined.

(3) These operations are used to extend an environment ρ, a function from type variables to Ξ, to $[\![\cdot]\!]\rho$, a function from type expressions to Ξ.

(4) We define what it means for an untyped term t to be a substitution instance of a typed term a, with respect to a given environment. The untyped, 'stripped' term $|a|$ will be an instance of a.

(5) Truth for typing judgments is defined by:

$$\models a:A \;\equiv\; \forall\rho\ \forall \text{ instances } t \text{ of } a,\ t \in [\![A]\!]\rho.$$

In Proposition 7, by an induction on the derivation of typing judgments, all judgments are shown to be true.

(6) In Lemma 8, we show that typed term a is strongly normalizable if $|a|$ is strongly normalizable. Now our conclusion, Theorem 9, follows: by (5), if $\vdash a:A$, then $\models a:A$; and by (4), this implies $|a| \in [\![A]\!]\rho$. By (1) and (3) this implies that $|a|$ must be strongly normalizable. Thus by Lemma 8, a is strongly normalizable.

2.1. Type expressions, terms and reduction

Assume a denumerably infinite supply of type variables $V_1, V_2, V_3, \ldots$, and let X, Y and Z range over them. Define the type expressions as follows.

- X is a type expression.
- If A and B are type expressions, then so are $A \to B$ and $\Delta X.A$.
- If X occurs positively in type expression A, then $\mu X.A$ and $\nu X.A$ are type expressions.

In type expressions $\Delta X.A$, $\mu X.A$ and $\nu X.A$, the X before the dot becomes bound, as well as any free occurrence of X in A. ('$\Delta X.A$' is our notation for type abstraction. It is sometimes written $\Pi X.A$.) 'X occurs positively in A', Pos(A, X), iff each free occurrence of X in A is on the left-hand side of an even number of $\to$'s; similarly, Neg(A, X) iff each free occurrence is on the left-hand side of an odd number of $\to$'s. Let A, B and C range over type expressions. Let FV(A) denote the set of type variables occurring free in A. As one might expect, in the standard encodings [8]:

$$A \times B \equiv \Delta Z.(A \to B \to Z) \to Z,$$
$$A + B \equiv \Delta Z.(A \to Z) \to (A \to Z) \to Z.$$

X occurs positively in them iff X occurs positively in A and B. ('$\to$' associates to the right: $A \to B \to C \equiv A \to (B \to C)$.) The positivity requirement will ensure the monotonicity of certain complete lattice operations, and hence the existence of least and greatest fixed points for them, which will be the meaning we will give to the inductive types.

Typed terms are defined as follows.

Definition (*typed terms*). For each type $\mu \equiv \mu X.A$, there are constants

$$\text{in}^{\mu}: A[\mu/X] \to \mu,$$
$$R^{\mu}: \Delta Y.(\Delta X.(X \to Y) \to A \to Y) \to \mu \to Y,$$

and for each type $\nu \equiv \nu X.A$, there are constants

$$\text{out}^{\nu}: \nu \to A[\nu/X],$$
$$S^{\nu}: \Delta Y.(\Delta X.(Y \to X) \to Y \to A) \to Y \to \nu,$$

where Y does not occur free in A and is distinct from X. There are also the typing rules

$$\frac{}{x^A : A}, \qquad \frac{b : B}{\lambda x^A.b : A \to B},$$

$$\frac{b : B}{\Lambda X.b : \Delta X.B}, \quad X \text{ does not occur free in } A \text{ for any } x^A \text{ occurring free in } b,$$

$$\frac{c : A \to B \quad a : A}{ca : B}, \qquad \frac{b : \Delta X.B}{bA : B[A/X]}.$$

We will write $\vdash a : A$ when the typing judgment $a : A$ is derivable. For a type $\mu \equiv \mu X.A$, the constant in^{μ} lets us construct a term of type μ by constructing one

in the 'unrolling' of the μ type, $A[\mu/X]$. The constant R^{μ} is the induction combinator: for a given type Y, we can construct a function of type $\mu \to Y$ by supplying an 'induction hypothesis', a term of type $\Delta X.(X \to Y) \to A \to Y$ (remember free occurrences of X in A are being bound by the ΔX). Such a term can be thought of as extending the domain of a function of type $X \to Y$ to make it a function of type $A \to Y$. For a type $\nu \equiv \nu X.A$, we dualize this. The constant out^{ν} lets us 'unroll' a term of type ν into one of type $A[\nu/X]$. The constant S lets us construct terms of type $Y \to \nu$ according to a term supplied of type $\Delta X.(Y \to X) \to Y \to A$. Such a term uses a function of type $Y \to X$ to construct a function of type $Y \to A$.

The reductions for typed terms are as follows.

(1) $(\lambda x^{A}.a) \mapsto a[b/x^{A}]$,
(2) $\lambda x^{A}.(ax^{A}) \mapsto a$,
(3) $(\Lambda X.a)B \mapsto a[B/X]$,
(4) $\Lambda X.(aX) \mapsto a$,
(5) $R^{\mu}Ba(\text{in}(b)) \mapsto a\mu(R^{\mu}Ba)b$,
(6) $\text{out}(S^{\nu}Bab) \mapsto a\nu(S^{\nu}Ba)b$.

The first four reductions are the standard reductions of the second-order lambda calculus. The second reduction requires x^{A} not to occur free in a, and the fourth requires X not to occur free in a.

For $\mu \equiv \mu X.A$, in the reduction rule for R^{μ}, our inductively defined function of type $\mu \to B$ is $R^{\mu}Ba$, and so a must be of type $\Delta X.(X \to B) \to (A \to B)$. Letting X be μ, $a\mu$ will map a function of type $\mu \to B$, namely $R^{\mu}BA$, to one of type $A[\mu/X] \to B$, to which we can apply b. And this is how the reduction rules compute.

For $\nu \equiv \nu X.A$, in the reduction rule for S^{ν}, the roles of constructor and recursor are reversed. $S^{\nu}Ba$ is an inductively defined function of type $B \to \nu$, and so a must be of type $\Delta X.(B \to X) \to B \to A$. Letting X be ν, $a\nu$ will map a function of type $B \to \nu$, namely $S^{\nu}Ba$, to one of type $B \to A[\nu/X]$, to which we can apply b of type B. Again, this is how the reduction rules compute.

Define $a > b$ to mean a reduces to b by a single application of a reduction rule to a subterm of a, and define $a >^{*} b$ to mean a reduces to b by a finite number of such reductions.

2.2. An untyped term model

We will now construct a model of this lambda calculus, modeling types by almost arbitrary collections of untyped terms. We take as untyped terms the terms of the typed lambda calculus stripped of their type decorations: we write $|a|$ for the 'stripped' untyped term that corresponds to a typed term a. In other words we will have constants in, R, out and S in addition to untyped variables and the term forming operations of abstraction and application. Let t, u and v range over untyped terms. We also inherit stripped versions of the reduction rules of the typed terms:

(1) $(\lambda x.t)u \mapsto t[u/x]$,
(2) $\lambda x.(tx) \mapsto t$,
(3) $Rt(\text{in}(u)) \mapsto t(Rt)u$,
(4) $\text{out}(Stu) \mapsto t(St)u$.

As with the typed terms, we write $t > u$ to represent a single step reduction and $t >^* u$ to represent some finite number of reductions. Let T be the set of strongly normalizable untyped terms.

Say an untyped term is in *outermost normal form* (ONF) iff it has one of the following syntactic forms.

$$\lambda x.a, \quad R, \quad Rt, \quad \text{in}, \quad \text{in}\, t, \quad S, \quad St, \quad Stu, \quad \text{out}.$$

These are all terms for which reduction must apply only to proper subterms, if at all. Say a subset $\xi \subseteq \mathsf{T}$ is *closed under reduction* if $t \in \xi$ and $t >^* u$ implies $u \in \xi$. Say a subset $\xi \subseteq \mathsf{T}$ is *complete for outermost normal form* if $t \in \xi$, whenever $t \in \mathsf{T}$ and for any $u \in \text{ONF}$, $t >^* u$ implies $u \in \xi$. Now let Ξ be defined as the subsets $\xi \subseteq \mathsf{T}$ which are closed under reduction and complete for outermost normal form. We will model types by the elements of Ξ. Requiring terms to be strongly normalizable and have ξ closed under reduction may seem like reasonable requirements, but the last condition is a rather technical one, needed for certain lemmas like the following, which shows elements of Ξ are closed under the reverse of β reduction.

Lemma 1. *For $\xi \in \Xi$, $t[u/x] \in \xi$ and $u \in \mathsf{T}$ imply $(\lambda x.t)u \in \xi$.*

Proof. (1) $t \in \mathsf{T}$. Any reduction sequence of t,

$$t = t_0 > t_1 > t_2 > \cdots,$$

by substitution of u for x is a reduction sequence of $t[u/x]$:

$$t[u/x] = t_0[u/x] > t_1[u/x] > t_2[u/x] > \cdots,$$

and since $t[u/x] \in \mathsf{T}$, we must have $t \in \mathsf{T}$.

(2) $(\lambda x.t)u \in \mathsf{T}$. Since t and u are strongly normalizable, any infinite reduction must preform an outermost β or η reduction. In the β case, suppose there is a reduction sequence

$$(\lambda x.t)u >^* (\lambda x.t')u' > t'[u'/x] > \cdots,$$

where $t >^* t'$ and $u >^* u'$. Then it is easy to see that $t[u/x] >^* t'[u/x] >^* t'[u'/x]$, so by reduction closure $t'[u'/x] \in \xi$ and thus $t'[u'/x] \in \mathsf{T}$ and so this sequence is finite. In the η case, suppose there is a reduction sequence

$$(\lambda x.t)u >^* (\lambda x.t''x)u' > t''u' > \cdots,$$

where $t >^* t''x$ and $u >^* u'$. Let t' be $t''x$. Again, it is easy to see that $t[u/x] >^* t'[u/x] >^* t'[u'/x] = t''u'$, so by reduction closure $t'[u'/x] \in \xi$ and thus $t'[u'/x] \in \mathsf{T}$ and so this sequence is finite.

(3) Every term in ONF to which $(\lambda x.t)u$ reduces is in ξ. Suppose $(\lambda x.t)u$ reduces to a term $v \in \text{ONF}$. The reduction must have preformed an outermost β or η reduction, but as analyzed in the previous step, in either such reduction there is a term $t'[u'/x] \in \xi$ such that $t'[u'/x] >^* v$. By closure under reduction conclude $v \in \xi$.

(4) By (2) and (3) conclude that $(\lambda x\,.t)u \in \xi$. □

Proposition 2. *Ξ is a complete lattice under $\subseteq$, with the least element $\bot$ being the set of $u \in \mathsf{T}$ which cannot reduce to* ONF, *and the greatest lower bound of a nonempty subset of Ξ is its intersection.*

Note that variables are in $\bot$, and so in every $\xi \in \Xi$. Also note that the least upper bound of a non-empty chain of elements is its union.

Let $\Xi \xrightarrow{\text{mon}} \Xi$ be the collection of monotonic operations on Ξ. Give this function space the usual point-wise ordering. Now we can define operations on Ξ that correspond to the type constructors. For function space, let $\rightarrow \in \Xi \times \Xi \rightarrow \Xi$ be defined by

$$\xi_1 \rightarrow \xi_2 \equiv \{t \in \mathsf{T} \mid \forall u \ (u \in \xi_1 \Rightarrow tu \in \xi_2)\}.$$

We model type abstraction by intersection: let $\boldsymbol{\Delta} \in (\Xi \rightarrow \Xi) \rightarrow \Xi$ be defined by

$$\boldsymbol{\Delta}(f) \equiv \bigcap \{f(\xi) \mid \xi \in \Xi\}.$$

For the μ and ν types we, of course, will be taking least fixed points (lfp) and greatest fixed points (gfp). But recall $\mu \equiv \mu X.A$ is not equal to $A[\mu/X]$, but rather there is the isomorphism in^{μ} between them. And similarly for $\nu X.A$ and out^{ν}. This leads us to the definitions: let $\boldsymbol{\mu}, \boldsymbol{\nu} : (\Xi \xrightarrow{\text{mon}} \Xi) \rightarrow \Xi$ be defined as

$$\boldsymbol{\mu}(f) \equiv \text{lfp of } \lambda\xi.\{t \in \mathsf{T} \mid \forall u \ (t >^* \text{in}(u) \Rightarrow u \in f(\xi))\},$$
$$\boldsymbol{\nu}(f) \equiv \text{gfp of } \lambda\xi.\{t \in \mathsf{T} \mid \text{out}(t) \in f(\xi)\}.$$

(Note we are using 'bold lambda' as a notation for functions in the model.)

Proposition 3. *The definitions given above are well formed. Moreover, $\rightarrow$ is anti-monotonic in its first argument and monotonic in its second, and $\boldsymbol{\Delta}$, $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ are all monotonic.*

Define *environments* to be mappings ρ from type variables to Ξ. If $\xi \in \Xi$, then let $\rho[\xi/X]$ be the environment where

$$\rho[\xi/X](Y) \equiv \begin{cases} \xi, & \text{if } X = Y, \\ \rho(Y), & \text{otherwise.} \end{cases}$$

For a given environment ρ, we extend it to a mapping $[\![\cdot]\!]\rho$ from type expressions to Ξ by induction on type expressions:

$$[\![X]\!]\rho \equiv \rho(X), \qquad [\![A \rightarrow B]\!]\rho \equiv [\![A]\!]\rho \rightarrow [\![B]\!]\rho,$$
$$[\![\Delta X.A]\!]\rho \equiv \boldsymbol{\Delta}(\lambda\xi.[\![A]\!]\rho[\xi/X]), \qquad [\![\mu X.A]\!]\rho \equiv \boldsymbol{\mu}(\lambda\xi.[\![A]\!]\rho[\xi/X]),$$
$$[\![\nu X.A]\!]\rho \equiv \boldsymbol{\nu}(\lambda\xi.[\![A]\!]\rho[\xi/X]).$$

The following proposition lists the various assertions needed to show that this definition is sensible: that the meaning of a type expression is invariant under α-conversion of bound type variables, and only depends on the values the environment assigns to type variables that occur free in it; that the set of terms assigned to $[\![A]\!]\rho$ is an element of Ξ; and finally that the functions applied to the $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ operators are indeed monotonic. The proof is by a strightforward induction on A.

Proposition 4. *For all type expressions A and environments ρ and ρ',*
(1) *if* $Y \notin \mathrm{FV}(A)$, *then* $[\![A]\!]\rho = [\![A[Y/X]]\!]\rho[\rho(X)/Y]$;
(2) *if* $\forall X \in \mathrm{FV}(A).\rho(X) = \rho'(X)$, *then* $[\![A]\!]\rho = [\![A]\!]\rho'$;
(3) $[\![A]\!]\rho \in \Xi$;
(4) $\mathrm{Pos}(A, X) \Rightarrow \lambda\xi.[\![A]\!]\rho[\xi/X]$ *is a monotonic operation on* Ξ;
(5) $\mathrm{Neg}(A, X) \Rightarrow \lambda\xi.[\![A]\!]\rho[\xi/X]$ *is an anti-monotonic operation on* Ξ.

Note that $\lambda\xi.[\![A]\!]\rho[\xi/X]$ may be monotonic without being continuous: consider $\lambda\xi.[\![1 + X + (N \to X)]\!]\rho[\xi/X]$, where $N \equiv \mu Y.1 + Y$ and $1 \equiv \Delta Z.Z \to Z$.

2.3. Strong normalizability

Now we define a notion of truth for judgments. Fix environment ρ; an untyped term t is an *instance* of a typed term a with respect to ρ if a's free variables are among $x_1^{B_1}, \ldots, x_n^{B_n}$ and

$$t = |a[b_1, \ldots, b_n/x_1^{B_1}, \ldots, x_n^{B_n}]|,$$

where $b_i \in [\![B]\!]\rho$, for $1 \leq i \leq n$. In particular, since variables are always in each $\xi \in \Xi$, we can let $b_i = x_i$ and find $|a|$ is an instance of a. Define truth for judgments by

$$\models a : A \equiv \forall\rho\, \forall \text{ instances } t \text{ of } a,\ t \in [\![A]\!]\rho.$$

Here is the standard substitution lemma and a corollary we will need in Proposition 7.

Lemma 5. $[\![A[B/X]]\!]\rho = [\![A]\!]\rho[\xi/X]$, *where* $\xi = [\![B]\!]\rho$.

Corollary 6. *For* $\mu \equiv \mu X.A$ *and* $\nu \equiv \nu X.A$,

$$[\![\mu]\!]\rho = \{t \in \mathsf{T} \mid \forall u\ (t >^* \mathrm{in}(u) \Rightarrow u \in [\![A[\mu/X]]\!]\rho)\},$$
$$[\![\nu]\!]\rho = \{t \in \mathsf{T} \mid \mathrm{out}(t) \in [\![A[\nu/X]]\!]\rho\}.$$

We can now show soundness.

Proposition 7. $\vdash a : A$ *implies* $\models a : A$.

Proof. Proof by induction on the derivation of judgments. Each axiom and rule is considered in turn. Here are the justifications for the constants; the other rules are justified as in Girard's proof [6, 7].

$$\mathrm{in}^{\mu} : A[\mu/X] \to \mu.$$

Fix $\mu \equiv \mu X.A$, ρ and $t \in [\![A[\mu/X]]\!]\rho$. It suffices to show $\mathrm{in}(t) \in [\![\mu]\!]\rho$. Since $t \in \mathsf{T}$, $\mathrm{in}(t) \in \mathsf{T}$. If $\mathrm{in}(t) >^* \mathrm{in}(t')$, then $t >^* t'$ and $t' \in [\![A[\mu/X]]\!]\rho$ by closure under reduction, hence $\mathrm{in}(t) \in [\![\mu]\!]\rho$, by Corollary 6. Conclude $\models \mathrm{in}^{\mu} : A[\mu/X] \to \mu$.

$$R^{\mu} : \Delta Y.(\Delta X.(X \to Y) \to A \to Y) \to \mu \to Y.$$

Recall how the proof that a monotonic function on a complete lattice has a least fixed point proceeds. By ordinal induction an ascending chain is defined:

$$G_0 \equiv \bot, \qquad G_{\alpha+1} \equiv g(G_\alpha),$$
$$G_\lambda \equiv \bigsqcup \{G_\alpha \mid \alpha < \lambda\} \quad \text{for limit } \lambda.$$

Any fixed point is an upper bound for this chain. By a cardinality argument there must be a least ordinal α such that for some $\beta > \alpha$, $G_\alpha = G_\beta$. But by antisymmetry, $G_\alpha = G_{\alpha+1}$, and so G_α must be the least fixed point of g.

Fix $\mu \equiv \mu X.A$, ρ, $\xi \in \Xi$, and let $\rho_1 \equiv \rho[\xi/Y]$. The chain for least fixed point $[\![\mu]\!]\rho_1$ is;

$$\xi_0 \equiv \bot,$$
$$\xi_{\alpha+1} \equiv \{t \in \mathsf{T} \mid \forall u\ (t >^* \mathrm{in}(u) \Rightarrow u \in [\![A]\!]\rho_1[\xi_\alpha/X])\},$$
$$\xi_\lambda \equiv \bigcup_{\alpha<\lambda} \xi_\alpha \quad \text{for limit } \lambda.$$

Fix $t \in [\![\Delta X.(X \to Y) \to A \to Y]\!]\rho_1$. It suffices to prove:

$$\forall u \in [\![\mu]\!]\rho_1 \quad Rtu \in \xi.$$

which we will do by induction on the chain with limit $[\![\mu]\!]\rho_1$.

Base case: $u \in \xi_0$. Since t and u are in T, and since u cannot reduce to a term of the form $\mathrm{in}(u')$, $Rtu \in \bot$, so that $Rtu \in \xi$.

Inductive case: $u \in \xi_{\alpha+1}$. If u cannot reduce to a term of the form $\mathrm{in}(u')$, the argument is as in the base case. Otherwise, let $\rho_2 \equiv \rho_1[\xi_\alpha/X]$. Note $[\![\mu]\!]\rho = [\![\mu]\!]\rho_1 = [\![\mu]\!]\rho_2$ by Proposition 4. First we show $Rtu \in \mathsf{T}$: an infinite reduction of it must begin:

$$Rtu >^* Rt'u' > t'(Rt')u' > \cdots, \tag{1}$$

where $t >^* t'$ and $u >^* \mathrm{in}(u')$. Since $u \in \xi_{\alpha+1}$, $u' \in [\![A]\!]\rho_2$. But (i) t is in $[\![(X \to Y) \to A \to Y]\!]\rho_2$ by the definition of Δ, and so is t' by closure under reduction, (ii) Rt is in $[\![X \to Y]\!]\rho_2$ by the inductive assumption, and so is Rt' by closure under reduction; and by these two facts $t'(Rt')u' \in \xi$, so (1) is finite and therefore $Rtu \in \mathsf{T}$. Now suppose Rtu can reduce to some term $v \in \mathrm{ONF}$. The reduction sequence must look like (1), so there is a $t'(Rt')u' \in \xi$ that reduces to v. Thus $v \in \xi$ by closure under reduction, and as we have shown every term in ONF to which Rtu reduces is in ξ, conclude $Rtu \in \xi$. Therefore $\vdash R^\mu : \Delta Y.(\Delta X.(X \to Y) \to A \to Y) \to \mu \to Y$.

$$\mathrm{out}^\nu : \nu \to A[\nu/X].$$

Fix $\nu \equiv \nu X.A$, ρ and $t \in [\![\nu]\!]\rho$. It suffices to show $\mathrm{out}(t) \in [\![A[\nu/X]]\!]\rho$, but that is immediate from Corollary 6. Conclude $\vdash \mathrm{out}^\nu : \nu \to A[\nu/X]$.

$$S^\nu : \Delta Y.(\Delta X.(Y \to X) \to Y \to A) \to Y \to \nu.$$

Fix $\nu \equiv \nu X.A$, ρ, $\xi \in \Xi$, $t \in [\![\Delta X.(Y \to X) \to Y \to A]\!]\rho[\xi/Y]$ and $u \in \xi$. It suffices to show

$$Stu \in [\![\nu]\!]\rho[\xi/Y],$$

which can be shown by induction on the descending chain used to define $[\![v]\!]\rho[\xi/Y]$, in a similar argument to the proof for R. □

Lemma 8. *A typed term a is strongly normalizable if* $|a| \in \mathsf{T}$.

Proof. This follows from the observation that a 'type' β or η reduction reduces the number of Λ's in a term by exactly one. Suppose there is an infinite reduction sequence $a = a_0 > a_1 > a_2 > \cdots$. Stripping it yields a sequence $|a| = |a_0|, |a_1|, |a_2|, \ldots$ in which either $|a_i| > |a_{i+1}|$, when a_i reduces to a_{i+1} by a reduction other than a type β or η, or there is a 'stutter': $|a_i| = |a_{i+1}|$, when the reduction was a type reduction. As there can be only finite subsequences of stuttering, they can be removed to yield an infinite reduction of $|a|$. Contradiction. □

Now, this section's results follows.

Theorem 9. *All typed terms are strongly normalizable.*

Proof. For $\vdash a:A$, by Proposition 7 we have $\models a:A$. As an instance of a we take the stripped term $|a|$ and note $\models a:A$ implies $|a| \in [\![A]\!]\rho$ and so $|a| \in \mathsf{T}$. Finally, by Lemma 8 this implies that a is strongly normalizable. □

3. Equational type constraints

A *type constraint* is an equantion

$$\tau = T$$

between a type constant τ and a type expression T in which τ and other type constants may occur. These equations are used in the typing of terms. For example, the equation $\tau = \tau \to \tau$ can be used to 'type' all the terms of the untyped lambda calculus. The goal of this section is to give a condition **P** on sets of type constraints which will hold exactly when the resulting typed terms are strongly normalizable. For simplicity, we do this in the setting of the simply typed lambda calculus, although it should be clear how to add type abstraction.

The section is organized as follows. First, types, terms and reduction are defined. Second, we state condition **P** and show how its violation leads to the typing of diverging terms. Third, we give an equivalent formulation of **P** which identifies equivalence classes $[i]$ of the type constants τ_i and a total ordering $\prec$ of these classes. We wish to repeat the previous proof of strong normalizability, using $\prec$ to define a version of $[\![\cdot]\!]$, but two complications appear: $\prec$ may not be well founded, so there is no obvious well-founded order to type expressions; and in defining certain $[\![\tau_i]\!]$ simultaneously we must take fixed points of operations which are not all monotonic with respect to set inclusion. But these problems can be overcome, and we may outline the proof as follows.

(1) Let the definitions of T, Ξ, environments ρ, and $\to$ stand as in Section 2.

(2) Fix a derivation of $\vdash a^* : A^*$ and let I' be the union of classes $[i]$ for which τ_i appears in this derivation. Define a notion of level ordinals for type expressions with respect to I'.

(3) Environments ρ are extended to mappings $[\![\cdot]\!]\rho$ from type expressions to Ξ by level induction.

(4) Define the instances of a typed term, with respect to an environment, and define truth for the two forms of judgment: type membership and type equality.

(5) We cannot expect all the typing rules to be sound, because we have made no effort to ensure $\vDash \tau_i = T_i$ for $i \notin I'$. However, by Proposition 12, the axioms and rules actually used in the derivation of $\vdash a^* : A^*$ are sound, so we may conclude $\vDash a^* : A^*$.

(6) As in the first section, from this we can argue that a^* is strongly normalizable. Since a^* was arbitrary, when property **P** holds of the type constraints, all typed terms are strongly normalizable.

3.1. Type expressions and terms

We define type expressions to be those of the simply typed lambda calculus with atomic types τ_i for i in the fixed index set I. Given fixed type expressions T_i for $i \in I$, define judgments as follows.

Definition (typed terms with equational constraints).

$$\frac{}{\tau_i = T_i}\, i \in I, \qquad \frac{}{A = A},$$

$$\frac{A = B}{B = A}, \qquad \frac{A = B \quad B = C}{A = C},$$

$$\frac{A = B}{C = C'}, \quad C' \text{ is } C \text{ with some occurrences of } A \text{ replaced by } B,$$

$$\frac{a : A \quad A = B}{a : B}, \qquad \frac{}{x^A : A},$$

$$\frac{b : B}{\lambda x^A . b : A \to B}, \qquad \frac{c : A \to B \quad a : A}{ca : B}.$$

3.2. A condition on the constraints

In the previous section we used positivity to guarantee the monotonicity of certain semantic operations, and so were able to model inductive types. Here we have a similar positivity requirement: the required condition **P** is that τ_i must occur only positively in all types C judged equal to τ_i, that is

$$\mathbf{P}: \quad \forall C, \tau_i \, ((\vdash \tau_i = C) \Rightarrow \mathrm{Pos}(C, \tau_i)).$$

It is an easy exercise to give a polynomial algorithm for **P** when I is finite. If **P** fails to hold, C can be used in typing a diverging term. We construct such a term now. So suppose $\vdash \tau = C$, where there is a negative occurrence of τ in C. If this negative

occurrence of τ is on the left-hand side of $2k+1$ arrows, then C is C_{2k+1}, which is in the following form.

$$\begin{aligned}
&C_{2k+1} \equiv A_{2k+1}^1 \to \cdots \to A_{2k+1}^{n_{2k+1}} \to (C_{2k}) \to B_{2k+1},\\
&\vdots\\
&C_1 \equiv A_1^1 \to \cdots \to A_1^{n_1} \to (C_0) \to B_1,\\
&C_0 \equiv A_0^1 \to \cdots \to A_0^{n_0} \to \tau.
\end{aligned}$$

Some abbreviations will be useful:

$$\lambda \boldsymbol{x}_i.a \equiv \lambda x_i^1, \ldots, x_i^{n_i}.a, \qquad a\boldsymbol{z}_i \equiv a z_i^1 \cdots z_i^{n_i}.$$

Assume in the following that z_i^j and x_i^j are variables of type A_i^j, y_i is a variable of type C_i and f_i^j is a variable of type $B_j \to B_i$. For the case $k=0$ we define the terms c_i of type C_i as follows.

$$c_1 \equiv \lambda \boldsymbol{x}_1.\lambda y_0.f_1^1(y_0 \boldsymbol{z}_0 z_1 y_0), \qquad c_0 \equiv \lambda \boldsymbol{x}_0.c_1.$$

Now we reduce $c_1 \boldsymbol{z}_1 c_0$:

$$c_1 \boldsymbol{z}_1 c_0 >^* f_1^1(c_0 \boldsymbol{z}_0 \boldsymbol{z}_1 c_0) >^* f_1^1(c_1 \boldsymbol{z}_1 c_0).$$

And we conclude this term is not normalizable. For the case $k>0$ we define terms c_i of type C_i as follows.

$$\begin{aligned}
&c_1 \equiv \lambda \boldsymbol{x}_1.\lambda y_0.f_1^{2k+1}(y_0 \boldsymbol{z}_0 \boldsymbol{z}_{2k+1} y_{2k}),\\
&c_3 \equiv \lambda \boldsymbol{x}_3.\lambda y_2.f_3^2(y_2 \boldsymbol{z}_2 c_1),\\
&\vdots\\
&c_{2k+1} \equiv \lambda \boldsymbol{x}_{2k+1}.\lambda y_{2k}.f_{2k+1}^{2k}(y_{2k} \boldsymbol{z}_{2k} c_{2k-1}),\\
&c_0 \equiv \lambda \boldsymbol{x}_0.c_{2k+1},\\
&c_2 \equiv \lambda \boldsymbol{x}_2.\lambda y_1.f_2^1(y_1 \boldsymbol{z}_1 c_0),\\
&\vdots\\
&c_{2k} \equiv \lambda \boldsymbol{x}_{2k}.\lambda y_{2k-1}.f_{2k}^{2k-1}(y_{2k-1} \boldsymbol{z}_{2k-1} c_{2k-2}).
\end{aligned}$$

Now we reduce $c_{2k+1} \boldsymbol{z}_{2k+1} c_{2k}$:

$$\begin{aligned}
&>^* f_{2k+1}^{2k}(c_{2k} \boldsymbol{z}_{2k} c_{2k-1}[c_{2k}/y_{2k}]) >^* \cdots (c_{2k-1}[c_{2k}/y_{2k}] \boldsymbol{z}_{2k-1} c_{2k-2}) \cdots\\
&\quad\vdots\\
&>^* \cdots (c_2 \boldsymbol{z}_2 c_1[c_{2k}/y_{2k}]) \cdots >^* \cdots (c_1[c_{2k}/y_{2k}] \boldsymbol{z}_1 c_0) \cdots\\
&>^* \cdots (c_0 \boldsymbol{z}_0 \boldsymbol{z}_{2k+1} c_{2k}) \cdots >^* \cdots (c_{2k+1} \boldsymbol{z}_{2k+1} c_{2k}) \cdots.
\end{aligned}$$

And we conclude this term is not normalizable.

As an example, consider the second example given in the introduction:

$$\tau_0 = \tau_1 \to \tau_0, \qquad \tau_1 = \tau_2 \to \tau_1, \qquad \tau_2 = \tau_0 \to \tau_2.$$

From it we can derive $\vdash \tau_0 = ((\underline{\tau_0} \to \tau_2) \to \tau_1) \to \tau_0$, which contains the underlined negative occurrence of τ_0. The diverging term the previous prescription would construct is

$$(c_3)\lambda y^{\tau_0 \to \tau_2}.w^{\tau_2 \to \tau_1}.(yc_3), \quad \text{where } c_3 \equiv \lambda z^{(\tau_0 \to \tau_2) \to \tau_1}.v^{\tau_1 \to \tau_0}(z \lambda x^{\tau_0}.u^{\tau_0 \to \tau_2}(xz)).$$

The following states a more useful formulation of **P** which will be used in the next subsection.

Proposition 10. *Condition* **P** *is equivalent to the existence of an equivalence relation on I, whose classes* $[i]$ *are ordered by a relation* $\prec$ *such that if* τ_i *occurs in* T_j, *then* $[i]=[j]$ *or* $[i]\prec[j]$. *Furthermore, each class* $[i]$ *is partitioned into two disjoint halves,* $[i]^+$ *and* $[i]^-$, *with* $j\in[i]^+$ *implying* $\mathrm{Pos}(T_i, \tau_j)$ *and* $j\in[i]^-$ *implying* $\mathrm{Neg}(T_i, \tau_j)$.

3.3. Strong normalization

In this subsection we assume **P** and then argue that all typed terms are strongly normalizable. It is convenient to have type variables in order to solve the type equations, so extend the definition of type expressions by adding type variables V_i indexed by the index set I. Clearly, if this extended lambda calculus is strongly normalizable, so is the original one.

The notion now of an untyped term being in outermost normal form is being in the form $\lambda x.t$. Redefine Ξ with respect to this and let the definitions of environments and the semantic operation $\to$ stand as before. We would like to extend the environment ρ to a mapping $[\![\cdot]\!]\rho$ from type expressions to Ξ, but we cannot simply do structural induction on type expressions, because of atomic types τ_i. Order $\prec$ should be involved but it is not necessarily well founded: suppose I consists of the natural numbers and T_i is $\tau_{i+1}\to\tau_{i+1}$. The solution is to fix a derivation of $\vdash a^*:A^*$ and only be concerned with the τ_i that occur in it. Once we have concluded a^* is strongly normalizable, we can generalize to prove the final result.

So fix a derivation of $\vdash a^*:A^*$ and let I' be the union of the classes $[i]$ for τ_i appearing in this derivation. For $i\in I'$, let its rank $\#i$ be 1 plus the finite number of distinct classes $[j]\subseteq I'$ for which $[j]\prec[i]$. Now we can define a level ordinal for each type expression:

$$\begin{aligned}\mathscr{L}(X)&\equiv 0, \qquad \mathscr{L}(\tau_i)\equiv 0, \quad \text{if } i\notin I',\\ \mathscr{L}(B\to C)&\equiv \sup(\mathscr{L}(B), \mathscr{L}(C))+1,\\ \mathscr{L}(\tau_i)&\equiv \omega*(\#i), \quad \text{if } i\in I'.\end{aligned}$$

We wish to extend an environment ρ by this level induction to a mapping $[\![\cdot]\!]\rho$ but a second complication emerges: at level $\omega*(\#i)$ we will want to find a fixed point of operation $f\in\Xi^{[i]}\to\Xi^{[i]}$, where

$$f(\langle\xi_j\rangle_{j\in[i]})\equiv\langle[\![T_k[V_j/\tau_j]_{j\in[i]}]\!]\rho[\xi_j/V_j]_{j\in[i]}\rangle_{k\in[i]}. \tag{2}$$

This function may not be monotonic if we onrder $\Xi^{[i]}$ by the usual product ordering;

$$\langle\xi_i\rangle_{j\in[i]}\sqsubseteq\langle\xi'\rangle_{j\in[i]}\equiv\forall j\in[i]\ \xi_j\subseteq\xi'_j.$$

But f will be monotonic if we choose the ordering

$$\langle\xi_i\rangle_{j\in[i]}\sqsubseteq\langle\xi'\rangle_{j\in[i]}\equiv\forall j\in[i]^+\ \xi_j\subseteq\xi'_j\wedge\forall j\in[i]^-\ \xi'_j\subseteq\xi_j.$$

An example of such a situation is the first example in the introduction, $\tau_0 = \tau_1 \to \tau_0$ and $\tau_1 = \tau_0 \to \tau_1$. This ordering means that taking the least fixed point of f will yield the least solutions ξ_j for $j \in [i]^+$ and the greatest solutions for $j \in [i]^-$, but as we shall see, *any* fixed point will do.

Finally, we can define $[\![\cdot]\!]\rho$ by level induction:

$$[\![\tau_i]\!]\rho \equiv \bot, \quad \text{for } i \notin I', \qquad [\![X]\!]\rho \equiv \rho(X), \qquad [\![A \to B]\!]\rho \equiv [\![A]\!]\rho \to [\![B]\!]\rho.$$

and at level $\omega * (\#i)$ define $\langle [\![\tau_j]\!]\rho \rangle_{j \in [i]}$ to be the least fixed point of f, as defined in (2). Reinterpreting Proposition 4 and Lemma 5 in the context of this section, we see they all hold by the same arguments. We can draw another corollary from Lemma 5.

Corollary 11. *If $i \in I$, then $\forall \rho\, [\![\tau_i]\!]\rho = [\![T_i]\!]\rho$.*

Define *instance* as before and define truth by:

$$\vDash a : A \;\equiv\; \forall \rho\, \forall \text{ instances } t \text{ of } a,\ t \in [\![A]\!]\rho,$$

$$\vDash A = B \;\equiv\; \forall \rho\, [\![A]\!]\rho = [\![B]\!]\rho.$$

As noted earlier, *all* the rules may not be sound, but we can show enough to conclude $\vDash a^* : A^*$:

Proposition 12. *Except for $\tau_i = T_i$ when $i \notin I'$, all the axioms and rules of types terms with equational constants of the definition (in Section 3.1) are sound.*

Proof. Consider each rule or axiom in turn. The arguments for the rules and axioms carried over from the definition of typed terms (in Section 2.1) are as before. For $i \in I'$, $\vDash \tau_i = T_i$ holds by Corollary 11. The results for reflexivity, symmetry and transitivity of type equality are trivial to verify, the soundness of the substitution rules follows from Lemma 5, and the soundness of

$$\frac{a : B \quad A = B}{a : B}$$

is immediate from the definition of truth. □

Proposition 12 implies $\vDash a^* : A^*$, and as before, this implies a^* is strongly normalizable. Since a^* was an arbitrary typed term, we have shown the following theorem.

Theorem 13. *Property* **P** *implies all typed terms are strongly normalizable.*

4. Conclusions

We have shown how one may add general inductive type constructors to the second-order lambda calculus while preserving the strong normalizability property of terms. Second, we have given a syntactic condition on equational type

constraints which holds exactly when the resulting terms are strongly normalizable. In both cases a similar method of proof, based on Girard's method, was employed.

Acknowledgements

The author is grateful to Robert Constable, John Mitchell and Prakesh Panangaden for helpful discussions, Albert Meyer for suggesting the question of normalization with type constraints, and Val Beazu-Tannen, Furio Honsell and Dexter Kozen for further suggestions on the text.

References

[1] R. Amadio, K.B. Bruce and G. Longo, The finitary projection model for second order lambda calculus and solutions to higher order domain equations, Proc. 1st Ann. Symposium on Logic in Computer Science (IEEE Press, New York, 1986) 122–130.

[2] V. Beazu-Tannen and A.R. Meyer, Lambda calculus with constrained types, R. Parikh, ed., Logics of Programs, Lecture Notes in Comput. Sci. 193 (Springer, New York, 1985) 23–40.

[3] K.B. Bruce and A.R. Meyer, The semantics of second order polymorphic lambda calculus, in: Kahn, D. MacQueen and G. Plotkin, eds., Symposium on semantics of data types, Lecture Notes in Comput. Sci. 173 (Springer, New York, 1984) 131–144.

[4] M. Coppo and M. Zacchi, Type inference and logical relations, Proc. 1st Ann. Symposium on Logic in Computer Science (IEEE Press, New York, 1986) 218–226.

[5] S. Fortune, D. Leivant and M. O'Donnell, The expressiveness of simple and second-order type structures, J. Assoc. Comput. Mach. 30 (1) (1983) 151–185.

[6] 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, J.E. Fenstad, ed., Proc. Second Scandinavian Logic Symposium (North-Holland, Amsterdam, 1971) 63–92.

[7] J.-Y. Girard, Interprétation fonctinelle et elimination des coupures dans l'arithmetique d'order superieur, Ph.D. Thesis, Univ. Paris, (1972).

[8] D. Prawitz, Natural Deduction (Almqvist and Wiksell, Stockholm, 1965).

[9] J. Reynolds, Towards a theory of type structures, in: B. Robinet, ed., Colloque sur la Programmation, Lecture Notes in Comput. Sci. 19 (Springer, New York, 1974) 403–425.