Strong Normalization and Equi-(co)inductive Types

Andreas Abel*

Department of Computer Science, University of Munich
Oettingenstr.67, D-80538 München, Germany
abel@tcs.ifi.lmu.de

Abstract. A type system for the lambda-calculus enriched with recursive and corecursive functions over equi-inductive and -coinductive types is presented in which all well-typed programs are strongly normalizing. The choice of equi-inductive types, instead of the more common iso-inductive types, influences both reduction rules and the strong normalization proof. By embedding iso- into equi-types, the latter ones are recognized as more fundamental. A model based on orthogonality is constructed where a semantical type corresponds to a set of observations, and soundness of the type system is proven.

1 Introduction

Theorem provers based on the Curry-Howard-Isomorphism, such as Agda, Coq, Epigram, or LEGO are built on dependent types and use inductive and coinductive types to formalize data structures, object languages, logics, judgments, derivations, etc. Proofs by induction or coinduction are represented as recursive or corecursive programs, where only total programs represent valid proofs. As a consequence, only total programs, which are defined on all well-typed inputs, are accepted, and totality is checked by some static analysis (in case of Coq), or ensured by construction (in case of Epigram), or simply assumed (in case of LEGO and the current version of Agda).

Hughes, Pareto, and Sabry [16] have put forth a totality check based on sized types, such that each well-typed program is already total. Designed originally for embedded systems it has become attractive for theorem provers because of several advantages: First of all, its soundness can be proven by an interpretation of types as sets of total programs, as noted by Giménez [13]. Since soundness proofs for dependent types are delicate, the clarity that sized types offer should not be underestimated. Secondly, checking termination through types integrates the features of advanced type systems, most notably higher-order functions, higher-order types, polymorphism, and subtyping, into the termination check

* Research supported by the coordination action *TYPES* (510996) and thematic network *Applied Semantics II* (IST-2001-38957) of the European Union and the project *Cover* of the Swedish Foundation of Strategic Research (SSF).

without extra effort. Some advanced examples of what one can do with type-based termination, but not with syntactical, "term-based" termination checks, are given in other works of the author [4, 3, 2]. And last, type-based termination is just (a) sized inductive and coinductive types with subtyping induced by the sizes plus (b) typing rules for recursive and corecursive functions which ensure well-foundedness by checking that sizes are decreased in recursive instances. Due to this conceptual simplicity it is planned to replace the current term-based termination check in Coq by a type-based one; in recent works, sized types have been integrated with dependent types [9, 10].

Dependently typed languages, such as the languages of the theorem provers listed above, need to compare terms for equality during type-checking. In the presence of recursion, this equality test is necessarily incomplete.[1] A common heuristics is to normalize the terms and compare them for syntactic equality. In general, these terms are open, i. e., have free variables, and normalization takes place in all term positions, also under binders. This complicates matters considerably: Although a function is total in a semantical sense and terminates on all closed terms under lazy evaluation, it will probably diverge on open terms under full evaluation.[2] Hence, unfolding recursion has to be sensibly restricted during normalization. In related work [13, 8], inductive types are given by constructors, and a recursive function is only unfolded if its "recursive" argument, i. e., the argument that gets smaller in recursive calls, is a constructor.

We take a more foundational approach and consider a language, $\mathsf{F}_\omega^\wedge$, without constructors. Programs of $\mathsf{F}_\omega^\wedge$ are just λ-terms over constants fix_n^μ and fix_n^ν which introduce recursive and corecursive functions with n leading "parametric", i. e., non-recursive arguments. A recursive function $\mathsf{fix}_n^\mu\, s\, t_1\, \ldots\, t_n\, v$ with body s, parametric arguments t_i and recursive argument v is unfolded if v is a value, i. e., a λ-abstraction or an under-applied, meaning not fully applied, (co)recursive function. A corecursive function $\mathsf{fix}_n^\nu\, s\, t_1 \ldots t_n$ is unfolded if it is in evaluation position, e. g., applied to some term. In this article, we prove that this strategy is strongly normalizing for programs which are accepted by the sized type system.

For now, $\mathsf{F}_\omega^\wedge$ does not feature dependent types—they are not essential to studying the operational semantics, but cause considerable complications in the normalization proof. However, $\mathsf{F}_\omega^\wedge$ has arbitrary-rank polymorphism, thus, elementary data types like unit type, product type and disjoint sum can be defined by the usual Church-encodings. Inductive types are not given by constructors; instead we have least fixed-point types $\mu^a F$ which denote the ath iteration of type constructor F. Semantically $\mu^0 F$ is empty, $\mu^{a+1} F = F\,(\mu^a F)$, and for limit ordinals λ, $\mu^\lambda F$ denotes the union of all $\mu^a F$ for $a < \lambda$. It may help to think of the size index a as an strict upper bound for the height of the inhabitants of $\mu^a F$, viewed as trees. Dually, we have sized coinductive types $\nu^a F$, and a denotes

[1] And one would not expect that this test succeeds for the equation $f\,(\mathsf{fix}\,(g \circ f)) = \mathsf{fix}\,(f \circ g)$ given arbitrary (well-typed programs) f and g.

[2] Consider the recursive zero-function $\mathsf{zero}\, x = \mathsf{match}\ x\ \mathsf{with}\ 0 \mapsto 0 \mid y+1 \mapsto \mathsf{zero}\, y$. If applying zero to a variable triggers unfolding of recursion, normalization of zero will diverge.

the minimum number of elementary destructions one can safely perform on an element of $\nu^a F$, which is, in case of streams, the minimum number of elements one can read off this stream.

With sum and product types, common inductive types can be expressed as $\mu^a F$ for a suitable F; their constructors are then simply λ-terms. However, there is a design choice: *equi-inductive* types have $\mu^{a+1} F = F\,(\mu^a F)$ as a type equation in the system; *iso-inductive* types stipulate that $\mu^{a+1} F$ and $F\,(\mu^a F)$ are only isomorphic, witnessed by a folding operation $\mathsf{in} : F\,(\mu^a F) \to \mu^{a+1} F$ and an unfolding operation $\mathsf{out} : \mu^{a+1} F \to F\,(\mu^a F)$. The iso-approach has been taken in previous work by the author [3] and seems to be more common [12, 6, 19, 7], since it has a simpler theory. We go the foundational path and choose the "equi" flavor, which has consequences for the operational semantics and the normalization proof: since there are less syntactical "clutches" to hold on, more structure has to be built in the semantics.

Overview. In Section 2 we present System $\mathsf{F}^{\widehat{\omega}}$ with typing rules which only accept strongly normalizing functions. In Section 3 we motivate the reduction rules of $\mathsf{F}^{\widehat{\omega}}$ which are affected by *equi*-(co)inductive types. By embedding iso- into equi-inductive types in Section 4, we justify that equi-types are more fundamental than iso-types. We then proceed to develop a semantical notion of type, based on strong normalization and orthogonality (Section 5). Finally, we sketch the soundness proof for $\mathsf{F}^{\widehat{\omega}}$ in Section 6 and discuss some related work in Section 7.

2 System $\mathsf{F}^{\widehat{\omega}}$

Like in System F^{ω}, expressions of $\mathsf{F}^{\widehat{\omega}}$ are separated into three levels: kinds, type constructors, and terms (objects). Figure 1 summarizes the first two levels. In contrast to the standard presentation, we have a second base kind, ord, whose inhabitants are syntactic ordinals. Canonical ordinals are either $\mathsf{s}^n \imath = \mathsf{s}\,(\mathsf{s} \ldots (\mathsf{s}\,\imath))$ (notation: $\imath + n$), the nth successor of an ordinal variable $\imath$, or ∞, the closure ordinal of all inductive and coinductive types of $\mathsf{F}^{\widehat{\omega}}$. In spite of the economic syntax, expressions of kind ord, which we will refer to as *size expressions*, semantically denote ordinals up to the ωth uncountable (see Sect. 5.3). We use the metavariable a to range over size expressions and the metavariable $\imath$ to range over size variables. The metavariable X ranges over type constructor variables, which includes size variables.

Another feature of $\mathsf{F}^{\widehat{\omega}}$ are polarized kinds [27, 5, 1]. Function kinds are labeled with a polarity p that classifies the inhabiting type constructors as covariant ($p = +$), contravariant ($p = -$), or non-variant ($p = \circ$), the last meaning mixed or unknown variance. Inductive types are introduced using the type constructor constant

$$\mu_\kappa : \mathsf{ord} \xrightarrow{+} (\kappa \xrightarrow{+} \kappa) \xrightarrow{+} \kappa.$$

We write $\mu_\kappa\, a\, F$ usually as $\mu^a_\kappa F$ and drop index kind κ if clear from the context. The underlying type constructor $F : \kappa \xrightarrow{+} \kappa$ must be covariant—otherwise

Syntactic categories.

p	$::= + \mid - \mid \circ$	polarity
κ	$::= * \mid \mathsf{ord} \mid p\kappa \to \kappa'$	kind
κ_*	$::= * \mid p\kappa_* \to \kappa'_*$	pure kind
a, b, A, B, F, G	$::= C \mid X \mid \lambda X{:}\kappa.\, F \mid F\, G$	(type) constructor
C	$::= \to \mid \forall_\kappa \mid \mu_{\kappa_*} \mid \nu_{\kappa_*} \mid \mathsf{s} \mid \infty$	constructor constant
Δ	$::= \diamond \mid \Delta, X{:}p\kappa$	polarized context

The signature Σ assigns kinds to constants ($\kappa \xrightarrow{p} \kappa'$ means $p\kappa \to \kappa'$).

$\to$	$: * \xrightarrow{-} * \xrightarrow{+} *$	function space
$\forall_\kappa$	$: (\kappa \xrightarrow{\circ} *) \xrightarrow{+} *$	quantification
μ_{κ_*}	$: \mathsf{ord} \xrightarrow{+} (\kappa_* \xrightarrow{+} \kappa_*) \xrightarrow{+} \kappa_*$	inductive constructors
ν_{κ_*}	$: \mathsf{ord} \xrightarrow{-} (\kappa_* \xrightarrow{+} \kappa_*) \xrightarrow{+} \kappa_*$	coinductive constructors
s	$: \mathsf{ord} \xrightarrow{+} \mathsf{ord}$	successor of ordinal
∞	$: \mathsf{ord}$	infinity ordinal

Notation.

∇	for	μ or ν
∇^a	for	∇a
$\forall X{:}\kappa.\, A$	for	$\forall_\kappa(\lambda X{:}\kappa.\, A)$
$\forall X A$	for	$\forall X{:}\kappa.\, A$
$\lambda X F$	for	$\lambda X{:}\kappa.\, F$

Ordering and composition of polarities.

$$p \le p \qquad \circ \le p \qquad +p = p \qquad --= + \qquad \circ p = \circ \qquad pp' = p'p$$

Inverse application of a polarity to a context.

$$\begin{array}{ll} p^{-1}\diamond = \diamond & \circ^{-1}(\Delta, X{:}\circ\kappa) = (\circ^{-1}\Delta), X{:}\circ\kappa \\ +^{-1}\Delta = \Delta & \circ^{-1}(\Delta, X{:}+\kappa) = \circ^{-1}\Delta \\ -^{-1}(\Delta, X{:}p\kappa) = (-^{-1}\Delta), X{:}(-p)\kappa & \circ^{-1}(\Delta, X{:}-\kappa) = \circ^{-1}\Delta \end{array}$$

Kinding $\Delta \vdash F : \kappa$.

$$\textsc{kind-c}\ \frac{C{:}\kappa \in \Sigma}{\Delta \vdash C : \kappa} \qquad \textsc{kind-var}\ \frac{X{:}p\kappa \in \Delta \qquad p \le +}{\Delta \vdash X : \kappa}$$

$$\textsc{kind-abs}\ \frac{\Delta, X{:}p\kappa \vdash F : \kappa'}{\Delta \vdash \lambda X{:}\kappa.\, F : p\kappa \to \kappa'} \qquad \textsc{kind-app}\ \frac{\Delta \vdash F : p\kappa \to \kappa' \qquad p^{-1}\Delta \vdash G : \kappa}{\Delta \vdash F\, G : \kappa'}$$

Fig. 1. $\mathsf{F}^{\widehat{}}_\omega$: Kinds and constructors.

divergence is possible even without recursion (Mendler [21])—and κ must be a *pure* kind, i. e., not mention ord. The last condition seems necessary to define a uniform closure ordinal for all inductive types, meaning an ordinal ∞ such that $\mu^\infty F = F\,(\mu^\infty F)$. Inductive types are covariant in their size argument; the subtyping chain $\mu^a F \leq \mu^{a+1} F \leq \mu^{a+2} F \leq \cdots \leq \mu^\infty F$ holds. Dually, coinductive types, which are introduced by the constant

$$\nu_\kappa : \mathsf{ord} \xrightarrow{-} (\kappa \xrightarrow{+} \kappa) \xrightarrow{+} \kappa,$$

are contravariant, and we have the chain $\nu^\infty F \leq \cdots \leq \nu^{a+2} F \leq \nu^{a+1} F \leq \nu^a F$. Type constructors are identified modulo $\beta\eta$ and the laws $\mathsf{s}\,\infty = \infty$ and $\nabla^{a+1} F = F\,(\nabla^a F)$, where ∇ is a placeholder for μ or ν, here and in the following. Type constructor equality is kinded and given by the judgement $\Delta \vdash F = F' : \kappa$ for a kinding context Δ. Exept β and η, we have the axioms $\Delta \vdash \mathsf{s}\,\infty = \infty : \mathsf{ord}$ and $\Delta \vdash \nabla^{\mathsf{s}a}_\kappa F = F\,(\nabla^a_\kappa F) : \kappa$ for wellkinded F and $a : \mathsf{ord}$. Similarly, we have kinded higher-order subtyping $\Delta \vdash F \leq F' : \kappa$ induced by $\Delta \vdash a \leq \mathsf{s}\,a : \mathsf{ord}$ and $\Delta \vdash a \leq \infty : \mathsf{ord}$. Due to space restrictions, the rules have to be omitted, please find them in the author's thesis [2, Sect. 2.2].

Figure 2 displays terms and typing rules of $\mathsf{F}^{\widehat{}}_\omega$. Besides λ-terms, there are constants fix^μ_n to introduce functions that are recursive in the $n+1$st argument, and constants fix^ν_n to introduce corecursive functions with n arguments. The type $A(\imath)$ of a recursive function introduced by fix^μ_n must be of the shape

$$\forall \boldsymbol{X}.\, A_1 \to \cdots \to A_n \to \mu^\imath F^0\, \boldsymbol{G}^0 \to \cdots \to \mu^\imath F^m\, \boldsymbol{G}^m \to C,$$

where the A_i are contravariant in the size index $\imath$, C is covariant, and the F^j and $\boldsymbol{G}^j$ do not mention $\imath$. This criterion is written as A fix^μ_n-adm. (More precisely, a function of this type is simultaneously recursive in the arguments $n+1$ to $n+m+1$, but we are only interested in the first recursive argument.) Note that if the variance conditions were ignored, non-terminating functions would be accepted [3]. The type of a corecursive function $\mathsf{fix}^\nu_n\, s$ with n arguments has to be of the form

$$\forall \boldsymbol{X}.\, A_1 \to \cdots \to A_n \to \nu^\imath F\, \boldsymbol{G}$$

where the A_i are again contravariant in $\imath$ and F and $\boldsymbol{G}$ do not mention $\imath$ (criterion A fix^ν_n-adm).

Basic data types like unit, product, and sum can be added to the system, but we define them impredicatively (see Figure 2) since minimality of the system is a stronger concern in this work than efficiency. Some examples for sized types are:

$\mathsf{Nat} : \mathsf{ord} \xrightarrow{+} *$	$\mathsf{Tree} : \mathsf{ord} \xrightarrow{+} * \xrightarrow{-} * \xrightarrow{+} *$
$\mathsf{Nat} := \lambda\imath.\ \mu^\imath \lambda X.\, 1 + X$	$\mathsf{Tree} := \lambda\imath\lambda B\lambda A.\, \mu^\imath \lambda X.\, 1 + A \times (B \to X)$
$\mathsf{List} : \mathsf{ord} \xrightarrow{+} * \xrightarrow{+} *$	$\mathsf{Stream} : \mathsf{ord} \xrightarrow{-} * \xrightarrow{+} *$
$\mathsf{List} := \lambda\imath\lambda A.\ \mu^\imath \lambda X.\, 1 + A \times X$	$\mathsf{Stream} := \lambda\imath\lambda A.\, \nu^\imath \lambda X.\, A \times X$

A rich collection of examples is provided in the author's thesis [2, Sect. 3.2].

Syntactic categories.

Var	$\ni x$	variable
Tm	$\ni r, s, t ::= x \mid \lambda x t \mid r\, s \mid \mathsf{fix}_n^\mu \mid \mathsf{fix}_n^\nu$	term ($n \in \mathbb{N}$)
Val	$\ni v ::= \lambda x t \mid \mathsf{fix}_n^\nabla \mid \mathsf{fix}_n^\nabla s\, \boldsymbol{t}$ (where $\lvert \boldsymbol{t} \rvert \leq n$)	value ($\nabla \in \{\mu, \nu\}$)
Eframe	$\ni e(\_) ::= \_\ s \mid \mathsf{fix}_n^\mu\ s\ t_{1..n}\ \_$	evaluation frame
Ecxt	$\ni E ::= \mathsf{Id} \mid E \circ e \quad [\mathsf{Id}(r) = r, (E \circ e)(r) = E(e(r))]$	evaluation context
Cxt	$\ni \Gamma ::= \diamond \mid \Gamma, x : A \mid \Gamma, X : p\kappa$	typing context

Well-formed typing contexts.

$$\textsc{cxt-empty}\ \frac{}{\diamond\ \mathsf{cxt}} \qquad \textsc{cxt-tyvar}\ \frac{\Gamma\ \mathsf{cxt}}{\Gamma, X : \circ\kappa\ \mathsf{cxt}} \qquad \textsc{cxt-var}\ \frac{\Gamma\ \mathsf{cxt} \qquad \Gamma \vdash A : *}{\Gamma, x : A\ \mathsf{cxt}}$$

Typing $\Gamma \vdash t : A$.

$$\textsc{ty-var}\ \frac{(x : A) \in \Gamma \qquad \Gamma\ \mathsf{cxt}}{\Gamma \vdash x : A} \qquad \textsc{ty-abs}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x t : A \to B}$$

$$\textsc{ty-app}\ \frac{\Gamma \vdash r : A \to B \qquad \Gamma \vdash s : A}{\Gamma \vdash r\, s : B} \qquad \textsc{ty-sub}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A \leq B : *}{\Gamma \vdash t : B}$$

$$\textsc{ty-gen}\ \frac{\Gamma, X : \circ\kappa \vdash t : F\, X}{\Gamma \vdash t : \forall_\kappa F} \qquad \textsc{ty-inst}\ \frac{\Gamma \vdash t\ :\ \forall_\kappa F \qquad \Gamma \vdash G : \kappa}{\Gamma \vdash t : F\, G}$$

$$\textsc{ty-rec}\ \frac{\Gamma \vdash A : \mathsf{ord} \to * \qquad A\ \mathsf{fix}_n^\nabla\text{-}\mathsf{adm} \qquad \Gamma \vdash a : \mathsf{ord}}{\Gamma \vdash \mathsf{fix}_n^\nabla : (\forall \imath : \mathsf{ord}.\, A\, \imath \to A\, (\imath + 1)) \to A\, a}\ \nabla \in \{\mu, \nu\}$$

Impredicative definition of unit, product, and sum type.

$$\begin{aligned} 1 &:= \forall C.\, C \to C && : * \\ \times &:= \lambda A \lambda B \forall C.\, (A \to B \to C) \to C && : * \xrightarrow{+} * \xrightarrow{+} * \\ + &:= \lambda A \lambda B \forall C.\, (A \to C) \to (B \to C) \to C && : * \xrightarrow{+} * \xrightarrow{+} * \end{aligned}$$

Reduction $t \longrightarrow t'$: Closure of the following axioms under all term constructors:

$$\begin{aligned} (\lambda x t)\, s &\longrightarrow [s/x]t \\ \mathsf{fix}_n^\mu s\ t_{1..n}\, v &\longrightarrow s\, (\mathsf{fix}_n^\mu s)\, t_{1..n}\, v && \text{if } v \neq \mathsf{fix}_{n'}^\nu s'\, t_{1..n'} \\ e\, (\mathsf{fix}_n^\nu s\, t_{1..n}) &\longrightarrow e\, (s\, (\mathsf{fix}_n^\nu s)\, t_{1..n}) && \text{if } e \neq \mathsf{fix}_{n'}^\mu s'\, t_{1..n'}\ \_ \end{aligned}$$

Fig. 2. $\mathsf{F}_\omega^{\widehat{}}$: Terms and type assignment.

3 Operational semantics

In this section, the reduction rules for recursive and corecursive functions are developed. It is clear that unrestricted unfolding of fixed points $\mathsf{fix}\, s \longrightarrow s\,(\mathsf{fix}\, s)$ leads immediately to divergence. In the literature on type-based termination with iso-inductive types one finds the sound reduction rule $\mathsf{fix}^{\mu}_{n} s\, t_{1..n}\,(\mathsf{in}\, r) \longrightarrow s\,(\mathsf{fix}^{\mu}_{n} s)\, t_{1..n}\,(\mathsf{in}\, r)$, which requires the recursive argument to be a canonical inhabitant of the inductive type. Since the canonical inhabitants for equi-inductive types can be of any shape, we liberalize this rule to

$$\mathsf{fix}^{\mu}_{n} s\, t_{1..n}\, v \longrightarrow s\,(\mathsf{fix}^{\mu}_{n} s)\, t_{1..n}\, v, \tag{1}$$

where v is a value; in our case a λ-abstraction, or an under-applied (co)recursive function.

Elements of a coinductive type should be *delayed* by default, they should only be evaluated when they are *observed*, or *forced*, i. e., when they are surrounded by a non-empty evaluation context e. A candidate for a reduction rule is

$$e\,(\mathsf{fix}^{\nu}_{n} s\, t_{1..n}) \longrightarrow e\,(s\,(\mathsf{fix}^{\nu}_{n} s)\, t_{1..n}). \tag{2}$$

It is easy to find well-typed diverging terms if *less* than n arguments $t_{1..n}$ are required before the fixed-point can be unfolded.

Evaluation contexts $e(\_)$ are either applications $\_\; s$ or recursive functions $\mathsf{fix}^{\mu}_{n}\, s\, t_{1..n}\, \_$. The second form is necessary because, before reduction (1) can be performed, the recursive argument has to be evaluated, hence, must be in evaluation position. However, we run into problems if a corecursive value is in a recursive evaluation context, e. g., $\mathsf{fix}^{\mu}_{0}\,(\lambda x x)\,(\mathsf{fix}^{\nu}_{0}\,(\lambda z y))$. Such a term can be well-typed[3] if we use types like $\mu\lambda X.\, \nu\lambda Y.\, A$. Depending on which fixed-point we unfold we get completely different behavior: the recursion fix^{μ}_{0} can be unfolded ad infinitum, the term diverges. If we unfold the corecursion fix^{ν}_{0}, we arrive at $\mathsf{fix}^{\mu}_{0}\,(\lambda x x)\, y$, which is blocked. Another bad example is $\mathsf{fix}^{\mu}_{0} s\,(\mathsf{fix}^{\nu}_{0} s)$ with $s = \lambda z \lambda x x$. If we unfold recursion, we arrive at the normal form $\mathsf{fix}^{\nu}_{0} s$. Otherwise, if we first unfold corecursion, we obtain $\mathsf{fix}^{\mu}_{0} s\,(\lambda x x)$ which has normal form $\lambda x x$; the calculus is not locally confluent.

In this article, we restore acceptable behavior in the following way: A corecursive value inside a recursive evaluation context should block reduction, terms like $\mathsf{fix}^{\mu}_{0}\, s\,(\mathsf{fix}^{\nu}_{0}\, s')$ should be considered neutral, like variables. The drawback of this decision is that types like $\nu^{i}\lambda X.\, \mathsf{List}^{j}\, X$ (non-wellfounded, but finitely branching trees) are not well-supported by the system: Applying the List-length function to such a tree, like $\mathsf{fix}^{\nu}_{0}\lambda x.\mathsf{singletonList}(x)$, will not reduce. This seems to be a high price to pay for equi-(co)inductive types; in the iso-version, such problems do not arise. However, as we will see in the next section, even with these blocked terms, the equi-version is able to *completely simulate* reduction of the iso-version, so we have not lost anything in comparison with the iso-version, but we can gain something by improving the current reduction strategy in the equi-version.

[3] Note that $\mathsf{fix}^{\mu}_{0}\lambda x x : (\mu^{a}\lambda X X) \to C$ and $\mathsf{fix}^{\nu}_{0}\lambda x x : \nu^{a}\lambda X X$.

4 Embedding Iso- into Equi-(co)inductive Types

Why are we so interested in equi-inductive types, if they cause us trouble? Because they are the more primitive notion. Strong normalization for iso-inductive types can be directly obtained from the result for equi-inductive types, since there exists a trivial type and reduction preserving embedding. Let Delay_κ be defined by recursion on the pure kind κ as follows:

$$\begin{array}{ll} \mathsf{Delay}_* & (A) := 1 \to A \\ \mathsf{Delay}_{p\kappa\to\kappa'} & (F) := \lambda X{:}\kappa.\, \mathsf{Delay}_{\kappa'}(FX) \end{array}$$

Then we can define iso-inductive $\overline{\mu}_\kappa$ and iso-coinductive $\overline{\nu}_\kappa$ types in $\mathsf{F}^{\widehat{}}_\omega$ as follows:

$$\begin{array}{lll} \overline{\nabla}_\kappa & := \lambda\iota\lambda F{:}\kappa.\, \nabla^\iota_\kappa\, \mathsf{Delay}_\kappa(F) & \\ \mathsf{in}^\nabla(t) := \lambda z t & \text{where } z \notin \mathsf{FV}(t) & \mathsf{out}^\nabla(r) := r\,() \end{array}$$

Now $\mathsf{in}^\mu(t)$ is a non-corecursive *value* for each term t, and $\mathsf{out}^\nu(\_)$ is an applicative *evaluation context*, so we obtain in $\mathsf{F}^{\widehat{}}_\omega$ the reductions typical for iso-types:

$$\begin{array}{ll} \mathsf{fix}^\mu_n s\, t_{1..n}\,(\mathsf{in}^\mu(r)) & \longrightarrow s\,(\mathsf{fix}^\mu_n s)\, t_{1..n}\,(\mathsf{in}^\mu(r)) \\ \mathsf{out}^\nu(\mathsf{fix}^\nu_n s\, t_{1..n}) & \longrightarrow \mathsf{out}^\nu(s\,(\mathsf{fix}^\nu_n s)\, t_{1..n}). \end{array}$$

The reverse embedding, however, is *not* trivial. Since in the equi-system, folding and unfolding of inductive types can happen deep inside a type, equi-programs are not typable in the iso-system without major modifications. Only *typing derivations* of the equi-system can be translated into typing derivations of the iso-system. Thus, we consider equi-systems as more fundamental than iso-systems.

5 Semantical Types

A *strongly normalizing* term $t \in \mathsf{SN}$ is a term for which each reduction sequence ends in a value or a neutral term. A *neutral* term has either a variable in head position, or, in our case, a blocking fix^μ-fix^ν combination. We define SN inductively, extending previous works [15, 28, 17] by rules for (co)recursive terms (see Figure 3). Rule SN-ROLL is sound, but not strictly necessary; however, it simplifies the proof of extensionality (see lemma).

Safe reduction $t \rhd t'$ is a variant of weak head reduction which preserves strong normalization in both directions. In particular, SN is closed under safe expansion (rule SN-EXP). This works because we require $s \in \mathsf{SN}$ in rule SHR-β.

Lemma 1 (Properties of SN).

1. *Extensionality: If* $r\,x \in \mathsf{SN}$ *then* $r \in \mathsf{SN}$.
2. *Closure: If* $r \in \mathsf{SN}$ *and* $r \rhd r'$ *or* $r \lhd r'$ *then* $r' \in \mathsf{SN}$.
3. *Strong normalization: If* $r \in \mathsf{SN}$ *then there are no infinite reduction sequences* $r \longrightarrow r_1 \longrightarrow r_2 \longrightarrow \ldots$.
4. *Weak head normalization: If* $r \in \mathsf{SN}$ *then* $r \rhd r'$ *and* $r' \in \mathsf{SNe} \cup \mathsf{Val}$.

Strongly normalizing evaluation contexts $E \in \mathsf{Scxt}$.

$$\textsc{sc-id}\ \frac{}{\mathsf{Id} \in \mathsf{Scxt}} \qquad \textsc{sc-app}\ \frac{E \in \mathsf{Scxt} \qquad s \in \mathsf{SN}}{E \circ (\_\ s) \in \mathsf{Scxt}} \qquad \textsc{sc-rec}\ \frac{E \in \mathsf{Scxt} \qquad s, t_{1..n} \in \mathsf{SN}}{E \circ (\mathsf{fix}^{\mu}_{n} s\, t_{1..n}\ \_) \in \mathsf{Scxt}}$$

Strongly normalizing neutral terms $r \in \mathsf{SNe}$.

$$\textsc{sne-var}\ \frac{E \in \mathsf{Scxt}}{E(x) \in \mathsf{SNe}} \qquad \textsc{sne-fix}^{\mu}\textsc{fix}^{\nu}\ \frac{E \in \mathsf{Scxt} \qquad s, \boldsymbol{t}, s', \boldsymbol{t}' \in \mathsf{SN}}{E(\mathsf{fix}^{\mu}_{n} s\, t_{1..n}\, (\mathsf{fix}^{\nu}_{n'} s'\, t'_{1..n'})) \in \mathsf{SNe}}$$

Strongly normalizing terms $t \in \mathsf{SN}$.

$$\textsc{sn-sne}\ \frac{r \in \mathsf{SNe}}{r \in \mathsf{SN}} \qquad \textsc{sn-abs}\ \frac{t \in \mathsf{SN}}{\lambda x t \in \mathsf{SN}} \qquad \textsc{sn-fix}\ \frac{\boldsymbol{t} \in \mathsf{SN}}{\mathsf{fix}^{\nabla}_{n} \boldsymbol{t} \in \mathsf{SN}}\ |\boldsymbol{t}| \leq n+1$$

$$\textsc{sn-exp}\ \frac{r \rhd r' \qquad r' \in \mathsf{SN}}{r \in \mathsf{SN}} \qquad \textsc{sn-roll}\ \frac{s\,(\mathsf{fix}^{\nu}_{n} s)\, \boldsymbol{t} \in \mathsf{SN}}{\mathsf{fix}^{\nu}_{n} s\, \boldsymbol{t} \in \mathsf{SN}}\ |\boldsymbol{t}| \leq n$$

Safe reduction $t \rhd t'$ (plus reflexivity and transitivity).

$$\textsc{shr-}\beta\ \frac{s \in \mathsf{SN}}{E((\lambda x t)\, s) \rhd E([s/x]t)} \qquad \textsc{shr-rec}\ \frac{v \neq \mathsf{fix}^{\nu}_{n'} s'\, t'_{1..n'}}{E(\mathsf{fix}^{\mu}_{n} s\, t_{1..n}\, v) \rhd E(s\,(\mathsf{fix}^{\mu}_{n} s)\, t_{1..n}\, v)}$$

$$\textsc{shr-corec}\ \frac{e \neq \mathsf{fix}^{\mu}_{n'} s'\, t'_{1..n'}\ \_}{E(e(\mathsf{fix}^{\nu}_{n} s\, t_{1..n})) \rhd E(e(s\,(\mathsf{fix}^{\nu}_{n} s)\, t_{1..n}))}$$

Fig. 3. Strongly normalizing terms.

Alternatively, one can take 3. as the defining property of SN and from this prove 1., 2., and the SN- and SNE-rules in Figure 3. Property 4. holds then also, but only because there are no "junk terms" like $0\,(\lambda x x)$ in our language which block reduction but are neither neutral nor values.

In the remainder of this section, we prepare for the model construction for $\mathsf{F}^{\widehat{}}_{\omega}$ that will verify strong normalization. As usual, we interpret types as sets $\mathcal{A}$ of strongly normalizing terms, where $\mathcal{A}$ is closed under safe expansion. In the iso-case, we could interpret a coinductive type $\mathcal{C} := [\![\nu^{a+1} F]\!]$ as $\{r \mid \mathsf{out}\, r \in [\![F\,(\nu^a F)]\!]\}$, or in words, as these terms r whose *canonical observation* $\mathsf{out}\, r$ is already well-behaved. A corecursive object, say $\mathsf{fix}^{\nu}_{0} s$ can enter $\mathcal{C}$ by the safe expansion $\mathsf{out}\,(\mathsf{fix}^{\nu}_{0} s) \rhd \mathsf{out}\,(s\,(\mathsf{fix}^{\nu}_{0} s))$ provided that $s\,(\mathsf{fix}^{\nu}_{0} s) \in \mathcal{C}$ already. In the equi-case, however, a canonical observation is not available, we have no choice than to set the interpretation of $\mathcal{C}$ to the semantical type $[\![F\,(\nu^a F)]\!]$. How can now $\mathsf{fix}^{\nu}_{0} s$ enter $\mathcal{C}$? The solution is that each semantical type $\mathcal{A}$ is characterized by a set of evaluation contexts, $\mathcal{E}$, such that $t \in \mathcal{A}$ iff $E(t) \in \mathsf{SN}$ for all $E \in \mathcal{E}$. This characterization automatically ensures that $\mathcal{A}$ is closed under safe reduction and expansion. Now $\mathsf{fix}^{\nu}_{0}\, s$ enters $\mathcal{C}$ through the safe expansion $E(\mathsf{fix}^{\nu}_{0} s) \rhd E(s\,(\mathsf{fix}^{\nu}_{0} s))$. Formally, this will be proven in Lemma 5. In the following, we give constructions

and properties of semantical types. Due to lack of space, the presentation is rather dense, more details can be found in the author's thesis [2].

5.1 Orthogonality

We say that term t is *orthogonal* to evaluation context E,

$$t \perp E \;:\Longleftrightarrow\; E(t) \in \mathsf{SN}.$$

We could also say t *behaves well* in E. A *semantical type* $\mathcal{A}$ is the set of terms which behave well in all $E \in \mathcal{E}$, where $\mathcal{E}$ is some set of strongly normalizing evaluation contexts. The space of semantical types is called SAT.

$\mathcal{E}^\perp$	$:= \{t \mid t \perp E \text{ for all } E \in \mathcal{E}\}$		
$\mathcal{A}^\perp$	$:= \{E \mid t \perp E \text{ for all } t \in \mathcal{A}\}$		
SAT	$:= \{\mathcal{E}^\perp \mid \{\mathsf{Id}\} \subseteq \mathcal{E} \subseteq \mathsf{Scxt}\}$		saturated sets
$\mathcal{N}$	$:= \mathsf{Scxt}^\perp \supset \mathsf{SNe}$		neutral terms
$\mathcal{S}$	$:= \{\mathsf{Id}\}^\perp$		s.n. terms
$\overline{\mathcal{A}}$	$:= \mathcal{A}^{\perp\perp}$		closure
$\mathcal{A} \boxed{\to} \mathcal{E}^\perp$	$:= \{\mathsf{Id}, E \circ (\_\ s) \mid E \in \mathcal{E}, s \in \mathcal{A}\}^\perp$		function space
$\boxed{\bigcap}\mathfrak{A}$	$:= \bigcap \mathfrak{A}$	for $\mathfrak{A} \subseteq \mathsf{SAT}$	infimum
$\boxed{\bigcup}\mathfrak{A}$	$:= \overline{\bigcup \mathfrak{A}}$	for $\mathfrak{A} \subseteq \mathsf{SAT}$	supremum

The greatest semantical type is $\mathcal{S} = \mathsf{SN}$; the least semantical type $\mathcal{N}$ contains all terms which behave well in all good contexts, including the variables and even more, all safe expansions of strongly normalizing neutral terms. But due to rule SNE-FIX$^\mu$FIX$^\nu$, also some corecursive values inhabit $\mathcal{N}$, e. g., $\mathsf{fix}^\nu_0 \lambda zy$.

Lemma 2 (Properties of saturated sets).

1. *Galois connection:* $\mathcal{A}^\perp \supseteq \mathcal{E} \iff \mathcal{A} \subseteq \mathcal{E}^\perp$. *This implies* $\mathcal{A} \subseteq \mathcal{A}^{\perp\perp}$, $A \subseteq \mathcal{B} \Longrightarrow \mathcal{A}^\perp \supseteq \mathcal{B}^\perp$, *and* $\mathcal{A}^{\perp\perp\perp} = \mathcal{A}^\perp$, *and the same laws for* $\mathcal{E}$*s.*
2. *Biorthogonal closure: If* $\mathcal{A} \subseteq \mathcal{S}$ *then* $\{\mathsf{Id}\} \subseteq \mathcal{A}^\perp \subseteq \mathsf{Scxt}$ *and* $\mathcal{A}^{\perp\perp} \in \mathsf{SAT}$.
3. *De Morgan 1:* $\bigcap_{i\in I} \mathcal{E}_i^\perp = (\bigcup_{i\in I} \mathcal{E}_i)^\perp$.
4. *De Morgan 2:* $\bigcup_{i\in I} \mathcal{E}_i^\perp \subseteq (\bigcap_{i\in I} \mathcal{E}_i)^\perp$.
5. *Reduction/expansion closure: If* $t \in \mathcal{E}^\perp$ *and* $t \rhd t'$ *or* $t \lhd t'$ *then* $t' \in \mathcal{E}^\perp$.
6. *Normalization: If* $t \in \mathcal{A} \in \mathsf{SAT}$ *then either* $t \in \mathcal{N}$ *or* $t \rhd v$.
7. *Function space: If* $\mathcal{A} \subseteq \mathcal{S}$ *and* $\mathcal{B} \in \mathsf{SAT}$ *then* $\mathcal{A} \boxed{\to} \mathcal{B} \in \mathsf{SAT}$.
8. *Infimum and supremum: If* $\mathfrak{A} \subseteq \mathsf{SAT}$ *then* $\boxed{\bigcap}\mathfrak{A} \in \mathsf{SAT}$ *and* $\boxed{\bigcup}\mathfrak{A} \in \mathsf{SAT}$.

In general, the inclusion in law De Morgan 2 is strict; thus, taking the orthogonal seems to be an intuitionistic rather than a classical negation.

Lemma 3 (Abstraction and application). *Let* $\mathcal{B} \in \mathsf{SAT}$.

1. *If* $\mathsf{Var} \subseteq \mathcal{A}$ *and* $r\,s, [s/x]t \in \mathcal{B}$ *for all* $s \in \mathcal{A}$, *then* $r, \lambda xt \in \mathcal{A} \boxed{\to} \mathcal{B}$.
2. *If* $r \in \mathcal{A} \boxed{\to} \mathcal{B}$ *and* $s \in \mathcal{A}$ *then* $r\,s \in \mathcal{B}$.

The proof of 1. uses extensionality (Lemma 1) to show $r \perp \mathsf{Id}$ from $r\,x \in \mathcal{B}$.

5.2 Recursion and corecursion, semantically

In this section, we characterize admissible types for recursion and corecursion in our semantics and prove semantical soundness of type-based termination. Let O denote some initial segment of the set-theoretic ordinals.

The semantic type family $\mathcal{A} \in \mathsf{O} \to \mathsf{SAT}$ is *admissible for recursion on the* $n+1$*st argument* if

ADM-μ-SHAPE	there is an index set K and there are $\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{I}, \mathcal{C} \in K \times \mathsf{O} \to \mathsf{SAT}$ such that for all $\alpha \in \mathsf{O}$, $\mathcal{A}(\alpha) = \bigcap_{k \in K} (\mathcal{B}_{1..n}(k,\alpha) \boxrightarrow \mathcal{I}(k,\alpha) \boxrightarrow \mathcal{C}(k,\alpha))$,
ADM-μ-START	$\mathcal{I}(k,0) \subseteq \mathcal{N}$ for all $k \in K$, and
ADM-μ-LIMIT	$\bigcap_{\alpha<\lambda} \mathcal{A}(\alpha) \subseteq \mathcal{A}(\lambda)$ for all limits $\lambda \in \mathsf{O} \setminus \{0\}$.

In ADM-μ-SHAPE, the intersection $\bigcap$ stands for a quantification over types, the $\mathcal{B}_i$ for non-recursive arguments, the $\mathcal{I}$ for the recursive argument of inductive type, and $\mathcal{C}$ for the result type.

Lemma 4 (Recursion is a function). *Let $\mathcal{A} \in \mathsf{O} \to \mathsf{SAT}$ be admissible for recursion on the $n+1$st argument. If $s \in \mathcal{A}(\alpha) \boxrightarrow \mathcal{A}(\alpha+1)$ for all $\alpha+1 \in \mathsf{O}$, then $\mathsf{fix}^{\mu}_{n} s \in \mathcal{A}(\beta)$ for all $\beta \in \mathsf{O}$.*

Proof. By transfinite induction on $\beta \in \mathsf{O}$ [2, Lemma 3.32].

The soundness of corecursion makes crucial use of our definition of a semantical type by a set of evaluation contexts. It also requires that coinductive types denote the whole term universe $\mathcal{S}$ in the 0th iteration (ADM-ν-START).

The semantic type family $\mathcal{A} \in \mathsf{O} \to \mathsf{SAT}$ is *admissible for corecursion with n arguments* if

ADM-ν-SHAPE	for some index set K and $\mathcal{B}_{1..n}, \mathcal{C} \in K \times \mathsf{O} \to \mathsf{SAT}$, $\mathcal{A}(\alpha) = \bigcap_{k \in K} (\mathcal{B}_{1..n}(k,\alpha) \boxrightarrow \mathcal{C}(k,\alpha))$ for all $\alpha \in \mathsf{O}$,
ADM-ν-START	$\mathcal{S} \subseteq \mathcal{C}(k,0)$ for all $k \in K$, and
ADM-ν-LIMIT	$\bigcap_{\alpha<\lambda} \mathcal{A}(\alpha) \subseteq \mathcal{A}(\lambda)$ for all limits $\lambda \in \mathsf{O} \setminus \{0\}$.

Lemma 5 (Corecursion is a function). *Let $\mathcal{A} \in \mathsf{O} \to \mathsf{SAT}$ be admissible for corecursion with n arguments. If $s \in \mathcal{A}(\alpha) \boxrightarrow \mathcal{A}(\alpha+1)$ for all $\alpha+1 \in \mathsf{O}$, then $\mathsf{fix}^{\nu}_{n} s \in \mathcal{A}(\beta)$ for all $\beta \in \mathsf{O}$.*

Proof. By transfinite induction on $\beta \in \mathsf{O}$ [2, Lemma 3.37].

5.3 Lattices and Iteration

The saturated sets form a complete lattice $[\![*]\!] = \mathsf{SAT}$ with least element $\bot^* := \mathcal{N}$ and greatest element $\top^* := \mathcal{S}$. It is ordered by inclusion $\sqsubseteq^* := \subseteq$ and has arbitrary infima $\inf^* := \boxed{\bigcap}$ and suprema $\sup^* := \boxed{\bigcup}$. Let $[\![\mathsf{ord}]\!] := [0; \top^{\mathsf{ord}}]$ be an initial segment of the set-theoretic ordinals which is closed under suprema, such that all (co)inductive types reach their fixpoint at ordinal $\top^{\mathsf{ord}}$. An upper

bound for the $\top^{\mathsf{ord}}$ is the ωth uncountable [3], although the true closure ordinal is probably much smaller, and it would be interesting to find out more about it. With the usual ordering on ordinals, $[\![\mathsf{ord}]\!]$ constitutes a complete lattice as well. Function kinds $[\![\circ\kappa \to \kappa']\!] := [\![\kappa]\!] \to [\![\kappa']\!]$ are interpreted as set-theoretic function spaces; a covariant function kind denotes just the monotonic functions and a contravariant kind the antitonic ones. For all function kinds, ordering is defined pointwise: $\mathcal{F} \sqsubseteq^{p\kappa\to\kappa'} \mathcal{F}' :\iff \mathcal{F}(\mathcal{G}) \sqsubseteq^{\kappa'} \mathcal{F}'(\mathcal{G})$ for all $\mathcal{G} \in [\![\kappa]\!]$. Similarly, $\bot^{p\kappa\to\kappa'}(\mathcal{G}) := \bot^{\kappa'}$ is defined pointwise, and so are $\top^{p\kappa\to\kappa'}$, $\inf^{p\kappa\to\kappa'}$, and $\sup^{p\kappa\to\kappa'}$.

For monotone $\mathcal{F} \in [\![\kappa]\!] \xrightarrow{+} [\![\kappa]\!]$ we define iteration from below and above as usual:

$$\begin{array}{llll} \boldsymbol{\mu}^0\mathcal{F} & = \bot^\kappa & \boldsymbol{\nu}^0\mathcal{F} & = \top^\kappa \\ \boldsymbol{\mu}^{\alpha+1}\mathcal{F} & = \mathcal{F}(\boldsymbol{\mu}^\alpha\mathcal{F}) & \boldsymbol{\nu}^{\alpha+1}\mathcal{F} & = \mathcal{F}(\boldsymbol{\nu}^\alpha\mathcal{F}) \\ \boldsymbol{\mu}^\lambda\mathcal{F} & = \sup^\kappa_{\alpha<\lambda} \boldsymbol{\mu}^\alpha\mathcal{F} & \boldsymbol{\nu}^\lambda\mathcal{F} & = \inf^\kappa_{\alpha<\lambda} \boldsymbol{\nu}^\alpha\mathcal{F} \end{array}$$

For fixed $\mathcal{F}$, $\boldsymbol{\mu}^\alpha\mathcal{F}$ is monotonic in α and $\boldsymbol{\nu}^\alpha\mathcal{F}$ is antitonic in α.

6 Soundness

For a constructor constant $C : \kappa$, the semantics $[\![C]\!] \in [\![\kappa]\!]$ is defined as follows:

$$\begin{array}{llll} [\![\to]\!](\mathcal{A},\mathcal{B} \in [\![*]\!]) & := \mathcal{A} \boxrightarrow \mathcal{B} & [\![\infty]\!] & := \top^{\mathsf{ord}} \\ [\![\mu_\kappa]\!](\alpha)(\mathcal{F} \in [\![\kappa]\!] \xrightarrow{+} [\![\kappa]\!]) & := \boldsymbol{\mu}^\alpha\mathcal{F} & [\![\mathsf{s}]\!](\top^{\mathsf{ord}}) & := \top^{\mathsf{ord}} \\ [\![\nu_\kappa]\!](\alpha)(\mathcal{F} \in [\![\kappa]\!] \xrightarrow{+} [\![\kappa]\!]) & := \boldsymbol{\nu}^\alpha\mathcal{F} & [\![\mathsf{s}]\!](\alpha < \top^{\mathsf{ord}}) & := \alpha + 1 \\ [\![\forall_\kappa]\!](\mathcal{F} \in [\![\kappa]\!] \to [\![*]\!]) & := \bigcap_{\mathcal{G}\in[\![\kappa]\!]} \mathcal{F}(\mathcal{G}) & & \end{array}$$

We extend this semantics to constructors F in the usual way.

Let θ be a partial mapping from constructor variables to sets. We say $\theta \in [\![\Delta]\!]$ if $\theta(X) \in [\![\kappa]\!]$ for all $(X : p\kappa) \in \Delta$. A partial order on valuations is defined by $\theta \sqsubseteq \theta' \in [\![\Delta]\!] :\iff \theta(X) \sqsubseteq^p \theta'(X) \in [\![\kappa]\!]$ for all $(X : p\kappa) \in \Delta$. Herein, we have used $\sqsubseteq^-$ for $\sqsupseteq$, and $\sqsubseteq^\circ$ for $=$, and $\sqsubseteq^+$ as synonym for $\sqsubseteq$.

Theorem 1 (Soundness of type-related judgements). *Let $\theta, \theta' \in [\![\Delta]\!]$.*

1. *If $\Delta \vdash F : \kappa$ then $[\![F]\!]_\theta \in [\![\kappa]\!]$.*
2. *If $\Delta \vdash F = F' : \kappa$ and $\theta \sqsubseteq \theta' \in [\![\Delta]\!]$, then $[\![F]\!]_\theta \sqsubseteq [\![F']\!]_{\theta'} \in [\![\kappa]\!]$.*
3. *If $\Delta \vdash F \leq F' : \kappa$ and $\theta \sqsubseteq \theta' \in [\![\Delta]\!]$, then $[\![F]\!]_\theta \sqsubseteq [\![F']\!]_{\theta'} \in [\![\kappa]\!]$.*
4. *If $\Delta \vdash A$ fix^μ_n-adm, then $[\![A]\!]_\theta$ is admissible for recursion on the $n+1$st arg.*
5. *If $\Delta \vdash A$ fix^ν_n-adm, then $[\![A]\!]_\theta$ is admissible for corecursion with n arguments.*

We extend valuations θ to term variables and say $\theta \in [\![\Gamma]\!]$ if $\theta(X) \in [\![\kappa]\!]$ for all $(X : p\kappa) \in \Gamma$ and $\theta(x) \in [\![A]\!]_\theta$ for all $(x : A) \in \Gamma$. Let $(\!|t|\!)_\theta$ denote the capture-avoiding substitution of $\theta(x)$ for x in t, simultaneously for all $x \in \mathsf{FV}(t)$.

Theorem 2 (Soundness of $\mathsf{F}\widehat{\omega}$). *If $\Gamma \vdash t : A$ and $\theta \in [\![\Gamma]\!]$ then $(\!|t|\!)_\theta \in [\![A]\!]_\theta$.*

The theorem is proved by induction on the typing derivation [2, Thm. 3.49]. As a consequence, taking $\theta(x) = x$ for all $(x : A) \in \Gamma$ and $\theta(X) = \top^\kappa$ for all $(X : p\kappa) \in \Gamma$, we get $t = (\!|t|\!)_\theta \in [\![A]\!]_\theta \subseteq \mathsf{SN}$.

7 Conclusions

We have presented a type system for termination of recursive functions over equi-inductive and -coinductive types and shown its soundness by a model based on orthogonality. All reductions of the corresponding iso-system are simulated, hence, termination of the iso-system follows as a special case.

Parigot [24] already introduces equi-inductive types to model efficient recursion schemes in system AF_2, second order functional arithmetic. Raffalli [26] considers also equi-coinductive types. However, recursion is limited to Mendler-style (co)iteration [22], preventing a direct implementation of primitive recursive programs such as factorial. Iteration is but a special case of the recursion scheme of the present work, which generalizes course-of-value recursion.

Orthogonality has been introduced by Girard for the semantics of linear logic; it pops up again in Ludics [14]. Parigot has implicitly used orthogonality to prove strong normalization of second-order classical natural deduction [25]. His work has been extended by Matthes to positive fixed-point types [20]. Lindley and Stark [18] use orthogonality to show strong normalization of the monadic lambda-calculus and give credit to Pitts. Vouillon and Melliès [30] model recursive types with orthogonality, Vouillon [29] bases subtyping rules for union types on orthogonality.

Related works on type-based termination include: Hughes, Pareto, and Sabry [16], who treat first-order inductive and coinductive types that close at iteration ω. Their system is also *equi* in spirit [23, Ch. 3.10], however, they do not give reduction rules but construct a denotational model. Barthe et al. [8] prove strong normalization for recursive functions over sized inductive types of kind $*$. Although there are no explicit (un)folding operations in and out, the only way to generate inductive data is via constructors for labeled sums, which is crucially in the reduction rule for recursion. Thus, the system is *iso* in disguise, in is merged into the constructors, and out into case distinction. Blanqui [10] considers type-based termination for his Calculus of Algebraic Constructions—iso-inductive in spirit—which subsumes the Calculus of Inductive Constructions (CIC). Barthe, Gregoire, and Pastawski [9] have extended type-based termination to the CIC. Xi [31] bases termination on dependent types, albeit only dependencies on integer expressions, which gives him a great flexibility in termination measures. Since in his system a typing context can become unsatisfiable, he only shows call-by-value normalization of closed programs. Blanqui and Riba [11] manage to avoid unsatisfiable contexts, and thus, recover strong normalization.

In our treatment of equi-(co)inductive types, it is a bit unsatisfactory that terms like $\mathsf{fix}_0^\mu s\,(\mathsf{fix}_0^\nu\, s')$ are blocked. One could think of allowing both unfoldings, arriving at a non-confluent calculus. The techniques described in this paper are then no longer sufficient to prove strong normalization, but maybe methods used for normalization of classical logic could be employed.

Acknowledgments. Thanks to my supervisor, Martin Hofmann, and to Ralph Matthes for supporting my thesis work. Thanks to Lennart Beringer for proof reading and to the anonymous referees for their insightful and helpful comments.

Jérôme Vouillon's excellent article on semantics of union types [29] inspired me to build a model based on orthogonality. Thanks to Frédéric Blanqui and Colin Riba for their invitation to LORIA in February 2007 and for discussions which deepened my understanding of orthogonality, strong normalization, and equi-inductive types.

References

1. Abel, A.: Polarized subtyping for sized types. In: Grigoriev, D., Harrison, J., Hirsch, E. A., eds., Proc. of the 1st Int. Computer Science Symposium in Russia, CSR 2006, volume 3967 of Lect. Notes in Comput. Sci. Springer-Verlag (2006), 381–392
2. Abel, A.: A Polymorphic Lambda-Calculus with Sized Higher-Order Types. Ph.D. thesis, Ludwig-Maximilians-Universität München (2006)
3. Abel, A.: Semi-continuous sized types and termination. In: Ésik, Z., ed., Computer Science Logic, 20th Int. Workshop, CSL 2006, 15th Annual Conf. of the EACSL, volume 4207 of Lect. Notes in Comput. Sci. Springer-Verlag (2006), 72–88
4. Abel, A.: Towards generic programming with sized types. In: Uustalu, T., ed., Proc. of the 8th Int. Conf. on Mathematics of Program Construction, MPC '06, volume 4014 of Lect. Notes in Comput. Sci. Springer-Verlag (2006), 10–28
5. Abel, A., Matthes, R.: Fixed points of type constructors and primitive recursion. In: Marcinkowski, J., Tarlecki, A., eds., Computer Science Logic, 18th Int. Workshop, CSL 2004, 13th Annual Conf. of the EACSL, volume 3210 of Lect. Notes in Comput. Sci. Springer-Verlag (2004), 190–204
6. Altenkirch, T.: Constructions, Inductive Types and Strong Normalization. Ph.D. thesis, University of Edinburgh (1993)
7. Altenkirch, T.: Logical relations and inductive/coinductive types. In: Gottlob, G., Grandjean, E., Seyr, K., eds., Computer Science Logic, 12th Int. Workshop, CSL '99, 7th Annual Conf. of the EACSL. Lect. Notes in Comput. Sci., Springer-Verlag (1999), 343–354
8. Barthe, G., Frade, M. J., Giménez, E., Pinto, L., Uustalu, T.: Type-based termination of recursive definitions. Math. Struct. in Comput. Sci. **14** (2004) 1–45
9. Barthe, G., Grégoire, B., Pastawski, F.: CIC$\hat{}$: Type-based termination of recursive definitions in the Calculus of Inductive Constructions. In: Hermann, M., Voronkov, A., eds., Proc. of the 13th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2006, volume 4246 of Lect. Notes in Comput. Sci. Springer-Verlag (2006), 257–271
10. Blanqui, F.: A type-based termination criterion for dependently-typed higher-order rewrite systems. In: van Oostrom, V., ed., Rewriting Techniques and Applications (RTA 2004), Aachen, Germany, volume 3091 of Lect. Notes in Comput. Sci. Springer-Verlag (2004), 24–39
11. Blanqui, F., Riba, C.: Combining typing and size constraints for checking the termination of higher-order conditional rewrite systems. In: Hermann, M., Voronkov, A., eds., Proc. of the 13th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2006, volume 4246 of Lect. Notes in Comput. Sci. Springer-Verlag (2006), 105–119
12. Geuvers, H.: Inductive and coinductive types with iteration and recursion. In: Nordström, B., Pettersson, K., Plotkin, G., eds., Types for Proofs and Programs (TYPES'92), Båstad, Sweden (1992), 193–217

13. Giménez, E.: Structural recursive definitions in type theory. In: Larsen, K. G., Skyum, S., Winskel, G., eds., Int. Colloquium on Automata, Languages and Programming (ICALP'98), Aalborg, Denmark, volume 1443 of Lect. Notes in Comput. Sci. Springer-Verlag (1998), 397–408
14. Girard, J.-Y.: Locus solum: From the rules of logic to the logic of rules. Math. Struct. in Comput. Sci. **11** (2001) 301–506
15. Goguen, H.: Typed operational semantics. In: Deziani-Ciancaglini, M., Plotkin, G. D., eds., Proc. of the 2nd Int. Conf. on Typed Lambda Calculi and Applications, TLCA '95, volume 902 of Lect. Notes in Comput. Sci. Springer-Verlag (1995), 186–200
16. Hughes, J., Pareto, L., Sabry, A.: Proving the correctness of reactive systems using sized types. In: Proc. of the 23rd ACM Symp. on Principles of Programming Languages, POPL'96 (1996), 410–423
17. Joachimski, F., Matthes, R.: Short proofs of normalization. Archive of Mathematical Logic **42** (2003) 59–87
18. Lindley, S., Stark, I.: Reducibility and $\top\top$-lifting for computation types. In: Urzyczyn, P., ed., Proc. of the 7th Int. Conf. on Typed Lambda Calculi and Applications, TLCA 2005, volume 3461 of Lect. Notes in Comput. Sci. Springer-Verlag (2005)
19. Matthes, R.: Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types. Ph.D. thesis, Ludwig-Maximilians-University (1998)
20. Matthes, R.: Non-strictly positive fixed-points for classical natural deduction. Ann. Pure Appl. Logic **133** (2005) 205–230
21. Mendler, N. P.: Recursive types and type constraints in second-order lambda calculus. In: Proc. of the 2nd IEEE Symp. on Logic in Computer Science (LICS'87). IEEE Computer Soc. Press (1987), 30–36
22. Mendler, N. P.: Inductive types and type constraints in the second-order lambda calculus. Annals of Pure and Applied Logic **51** (1991) 159–172
23. Pareto, L.: Types for Crash Prevention. Ph.D. thesis, Chalmers University of Technology (2000)
24. Parigot, M.: Recursive programming with proofs. Theor. Comput. Sci. **94** (1992) 335–356
25. Parigot, M.: Proofs of strong normalization for second order classical natural deduction. The Journal of Symbolic Logic **62** (1997) 1461–1479
26. Raffalli, C.: Data types, infinity and equality in system af_2. In: Börger, E., Gurevich, Y., Meinke, K., eds., Proc. of the 7th Wksh. on Computer Science Logic, CSL '93, volume 832 of Lect. Notes in Comput. Sci. Springer-Verlag (1994), 280–294
27. Steffen, M.: Polarized Higher-Order Subtyping. Ph.D. thesis, Technische Fakultät, Universität Erlangen (1998)
28. van Raamsdonk, F., Severi, P., Sørensen, M. H., Xi, H.: Perpetual reductions in lambda calculus. Inf. Comput. **149** (1999) 173–225
29. Vouillon, J.: Subtyping union types. In: Marcinkowski, J., Tarlecki, A., eds., Computer Science Logic, 18th Int. Workshop, CSL 2004, 13th Annual Conf. of the EACSL, volume 3210 of Lect. Notes in Comput. Sci. Springer-Verlag (2004), 415–429
30. Vouillon, J., Melliès, P.-A.: Semantic types: A fresh look at the ideal model for types. In: Jones, N. D., Leroy, X., eds., Proc. of the 31st ACM Symp. on Principles of Programming Languages, POPL 2004. ACM Press (2004), 52–63
31. Xi, H.: Dependent types for program termination verification. J. Higher-Order and Symb. Comput. **15** (2002) 91–131