Mixed Inductive/Coinductive Types and Strong Normalization

Andreas Abel[⋆]

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

Abstract. We introduce the concept of *guarded* saturated sets, saturated sets of strongly normalizing terms closed under folding of corecursive functions. Using this tool, we can model equi-inductive and equi-coinductive types with terminating recursion and corecursion principles. Two type systems are presented: Mendler (co)iteration and sized types. As an application we show that we can directly represent the mixed inductive/coinductive type of stream processors with associated recursive operations.

1 Introduction

Symbolic evaluation, aka evaluation of terms with free variables, is used, amongst others, for optimization through partial evaluation in compilers and for checking term equivalence in languages based on dependent types—such as the theorem provers Agda, Coq, Epigram, and LEGO, founded on intensional type theory. In these applications, symbolic evaluation is required to terminate. My long term research goal is to develop expressive type systems that guarantee termination, and these type system shall include inductive and coinductive types.

Most research on inductive types has focused on the *iso*-style, i. e., there are explicit operations $\mathsf{in} : F\,(\mu F) \to \mu F$ and $\mathsf{out} : \mu F \to F\,(\mu F)$ for wrapping and unwrapping inductive types. In contrast, *equi*-inductive types come with the type equation $\mu F = F\,(\mu F)$, so wrapping and unwrapping is silent on the term level. Recently [4], I have put forth a type system for strongly normalizing terms with *equi*-(co)inductive types, but it behaves badly for so-called mixed inductive/coinductive types.

However, mixed inductive/coinductive types are important in the context of intensional type theory. Ghani, Hancock, and Pattinson [10] show how the type $\nu X.\, \mu Y.\, (B \times X) + (A \to Y)$ of stream processors is inhabited by codes of functions from streams over A to streams over B. They define *eating*, a function which takes a stream processor and an input stream and produces an output stream; *eating* executes the code of a stream processor. Swierstra [17] demonstrated how a small modification of stream processors could be used to model I/O in a dependently typed programming language.

[⋆] Research partially supported by the EU coordination action *TYPES* (510996).

Z. Shao (Ed.): APLAS 2007, LNCS 4807, pp. 286–301, 2007.

In this article, I present a concept which paves the way to a satisfactory treatment of mixed equi-(co)inductive types: *guarded type expressions*. The term *guardedness* has been used as a criterion whether corecursive programs denote well-defined functions. A corecursive call is guarded if it appears under a constructor of the coinductive type. In the same sense, a type expression is guarded if it is headed by a proper type constructor, like function space, cartesian product, disjoint sum, or a primitive type. Using the guardedness criterion, we can avoid coinductive types which contain no weak head values, and the remaining coinductive types have the pleasant property that they already contain a corecursive value if they contain its unfolding. This property gives rise to the new concept of *guarded saturated set*, on which we base our normalization proof.

Related Work. There is a rich body of work on type systems for termination of recursion, starting with Mendler [12], with contributions by Amadio and Coupet-Grimal [6], a group around Giménez and Barthe [7,8], and Blanqui and Riba [9]. All of these works are concerned with *iso*-(co)inductive types. Parigot [13] introduces equi-inductive and coinductive types in second-order functional arithmetic, an extension of System F. [15] provides Mendler iteration and coiteration schemes for these types and proves that all well-typed terms are hereditarily solvable, if the involved types satisfy a certain *strictness* condition. We require a condition only on coinductive types. Hughes, Pareto, and Sabry [11] present sized types in the equi-style, yet they consider only finitely branching data types and explicitly exclude a type of stream processors. In my previous attempt at equi-(co)inductive types [4] I constructed a semantics based on biorthogonals, which are due to Girard and have been successfully applied at interpreting languages based on classical logic (see, e. g., Parigot [14]). However, I had to consider a recursive function applied to a corecursive value blocked, preventing the use of mixed inductive/coinductive types. In this article, this flaw is overcome by a semantics based on saturated sets.

Overview. In Sec. 2, we will see a λ-calculus with recursion and corecursion and a saturated-set semantics of strongly normalizing terms. On this semantics, we base first a type system with Mendler (co)iteration (Sec. 3), and then a more flexible one with sized types (Sec. 4).

2 Untyped Language and Semantics

As an idealized purely functional programming language, we consider the λ-calculus with pairs and projections, injections and case analysis, and recursion and corecursion. In this section, we define semantical types as sets of strongly normalizing terms and prove formation, introduction and elimination rules for these semantical types. Especially interesting will be the principles for terminating recursion and corecursion which will be derived from the construction of inductive and coinductive types by ordinal iteration.

In all expressions throughout this article a dot "." denotes an opening parenthesis closing as far to the right as syntactically meaningful. $[M/x]N$ denotes the capture avoiding substitution of M for x in N. Let x range over a countably infinite set Var of variables. We define our language as the lambda calculus equipped with constants c. The values v are λ-abstractions, pairs, injections, and not fully applied constants (including recursive functions and corecursive values).

$$\begin{array}{lll}
c & ::= () \mid \mathsf{pair} \mid \mathsf{fst} \mid \mathsf{snd} \mid \mathsf{inl} \mid \mathsf{inr} \mid \mathsf{case} & \\
 & \quad \mid \mathsf{fix}^{\mu} \mid \mathsf{fix}^{\nu}_{n} \qquad (n \in \mathbb{N}) & \text{constants} \\
r, s, t & ::= c \mid x \mid \lambda x t \mid r\, s & \text{terms} \\
v, w & ::= c \mid \lambda x t \mid \mathsf{pair}\, r \mid \mathsf{pair}\, r\, s \mid \mathsf{inl}\, r \mid \mathsf{inr}\, r & \\
 & \quad \mid \mathsf{fix}^{\mu}\, s \mid \mathsf{fix}^{\nu}_{n} s\, \boldsymbol{t} \quad (|\boldsymbol{t}| \leq n) & \text{(weak-head) values} \\
e^{-}(\_) & ::= \_\mathsf{s} \mid \mathsf{fst}\, \_ \mid \mathsf{snd}\, \_ \mid \mathsf{case}\, \_\, s\, t & \text{non-recursive evaluation frames} \\
e(\_) & ::= e^{-}(\_) \mid \mathsf{fix}^{\mu}\, s\, \_ & \text{evaluation frames} \\
E(\_) & ::= \_ \mid E(e(\_)) & \text{evaluation contexts.}
\end{array}$$

We distinguish between possibly recursive $e(\_)$ and non-recursive $e^{-}(\_)$ evaluation frames. An evaluation context is $E(\_)$ is a stack of evaluation frames. Corecursive functions are only unfolded in a non-recursive evaluation frame

Reduction. Computation is modeled as small-step reduction relation. These are the axioms of β-contraction $e(v) \rightarrowtail t$.

$$\begin{array}{llll}
(\lambda x t)\, s & \rightarrowtail [s/x]t & \mathsf{fst}\,(\mathsf{pair}\, r\, s) & \rightarrowtail r \\
\mathsf{fix}^{\mu}\, s\, v & \rightarrowtail s\,(\mathsf{fix}^{\mu}\, s)\, v & \mathsf{snd}\,(\mathsf{pair}\, r\, s) & \rightarrowtail s \\
e^{-}(\mathsf{fix}^{\nu}_{n}\, s\, t_{1..n}) & \rightarrowtail e^{-}(s\,(\mathsf{fix}^{\nu}_{n}\, s)\, t_{1..n}) & \mathsf{case}\,(\mathsf{inl}\, r)\, s\, t & \rightarrowtail s\, r \\
 & & \mathsf{case}\,(\mathsf{inr}\, r)\, s\, t & \rightarrowtail t\, r
\end{array}$$

One-step reduction $\longrightarrow$ is the closure of $\rightarrowtail$ under all term constructors, multi-step reduction $\longrightarrow^{+}$ its transitive closure and $\longrightarrow^{*}$ its reflexive-transitive closure. Weak head reduction is defined by $E(t) \longrightarrow_{\mathsf{w}} E(t') \iff t \rightarrowtail t'$.

By only unfolding corecursive values in non-recursive evaluation frames, we avoid critical pairs. This does not lead to stuck terms, since in such a case the recursive function constituting the frame can be unfolded instead. In previous work [4], we considered a corecursive value in a recursive frame as stuck, leading to an unsatisfactory treatment of mixed induction/coinduction. The present work overcomes this flaw.

Strong normalization and saturated sets. A term t is *strongly normalizing* (s.n.), written $t \in \mathsf{SN}$, if all reduction sequences starting with t are finite. Note that subterms and reducts of s.n. terms are also s.n. Terms $E(x) \in \mathsf{SN}$ are called s.n. and neutral and their collection is denoted by SNe.

A set of terms $\mathcal{A}$ is a *semantical type*, written $\mathcal{A} \in \mathsf{SAT}_{\mathsf{u}}$, if

1. $\mathsf{SNe} \subseteq \mathcal{A} \subseteq \mathsf{SN}$,
2. each term in $\mathcal{A}$ weak-head reduces either to a value or a neutral term,
3. $\mathcal{A}$ is closed under weak head expansion that does not introduce diverging terms.

The first condition ensures that each semantical type contains all variables, such that we can construct an open s.n. term model of our calculus. The second condition is used to justify recursive functions $\mathsf{fix}^{\mu}\, s$, which reduce under call-by-(weak-head)-value (see Lemma 3). The third condition ensures that a redex like $(\lambda x t)\, s$ inhabits a semantical type if its reduct (here $[s/x]t$) does so. This is needed, for instance, to establish that $\lambda x t$ is in the semantical function space, and similarly for $\mathsf{pair}\, r\, s$, case distinctions and recursive functions.

The third condition can be made precise by defining *safe weak head reduction*, $\rhd$, by the following rules:

$$\begin{array}{llllll}
(\lambda x t)\, s & \rhd [s/x]t & \text{if } s \in \mathsf{SN} & \mathsf{fst}\,(\mathsf{pair}\, r\, s) & \rhd\, r & \text{if } s \in \mathsf{SN} \\
\mathsf{fix}^{\mu}\, s\, v & \rhd\, s\,(\mathsf{fix}^{\mu}\, s)\, v & & \mathsf{snd}\,(\mathsf{pair}\, r\, s) & \rhd\, s & \text{if } r \in \mathsf{SN} \\
e^{-}(\mathsf{fix}^{\nu}_{n}\, s\, t_{1..n}) & \rhd\, e^{-}(s\,(\mathsf{fix}^{\nu}_{n}\, s)\, t_{1..n}) & & \mathsf{case}\,(\mathsf{inl}\, r)\, s\, t & \rhd\, s\, r & \text{if } t \in \mathsf{SN} \\
E(t) & \rhd\, E(t') & \text{if } t \rhd t' & \mathsf{case}\,(\mathsf{inr}\, r)\, s\, t & \rhd\, t\, r & \text{if } s \in \mathsf{SN}
\end{array}$$

We define $\rhd$ as the reflexive-transitive closure of the above rules. Now if $t \rhd t' \in \mathsf{SN}$, then $t \in \mathsf{SN}$. For a reduction relation R, let ${}^{R}\mathcal{A} := \{t \mid t\ R\ t' \in \mathcal{A}\}$ and $\mathcal{A}^{R} := \{t' \mid \mathcal{A} \ni t\ R\ t'\}$. Condition 3 of semantical types can then be written as ${}^{\rhd}\mathcal{A} \subseteq \mathcal{A}$.

The greatest semantical type is called $\mathcal{S}$, it contains all s.n. terms except those whose weak-head reduction gets stuck, like $\mathsf{fst}\,(\lambda x x)$. The least semantical type is $\mathcal{N} := {}^{\rhd}\mathsf{SNe}$, and it is closed under s.n. evaluation contexts: if $r \in \mathcal{N}$ and $E(x) \in \mathsf{SNe}$ then $E(r) \in \mathcal{N}$.

Guarded semantical types. A semantical type $\mathcal{A}$ is *guarded*, written $\mathcal{A} \in \mathsf{SAT_g}$, if $s\,(\mathsf{fix}^{\nu}_{n}\, s)\, t_{1..n} \in \mathcal{A}$ implies $\mathsf{fix}^{\nu}_{n}\, s\, t_{1..n} \in \mathcal{A}$. Let $\blacktriangleright\, \supseteq\, \rhd$ be the reflexive-transitive closure of safe weak head reduction plus the axiom

$$\mathsf{fix}^{\nu}_{n}\, s\, t_{1..n} \blacktriangleright s\,(\mathsf{fix}^{\nu}_{n}\, s)\, t_{1..n}.$$

Note that $r \blacktriangleright r'$ implies $e^{-}(r) \rhd e^{-}(r')$.

A semantical type $\mathcal{A}$ is guarded iff ${}^{\blacktriangleright}\mathcal{A} \subseteq \mathcal{A}$. The premier example of a non-guarded type is $\mathcal{N}$. Note that $\mathcal{S}$ is closed under $\blacktriangleright$-expansion, since $\mathsf{fix}^{\nu}_{n}\, s\, t_{1..n}$ is a strongly normalizing value if $s, t_{1..n} \in \mathsf{SN}$. Thus, $\mathcal{S}$ is guarded.

Constructions on semantical types. The following constructions produce guarded semantical types, even for unguarded $\mathcal{A}, \mathcal{B} \in \mathsf{SAT_u}$.

$$\begin{array}{ll}
\mathcal{A} \to \mathcal{B} & := \{r \mid r\, s \in \mathcal{B} \text{ for all } s \in \mathcal{A}\} \\
\mathcal{A} \times \mathcal{B} & := \{r \mid \mathsf{fst}\, r \in \mathcal{A} \text{ and } \mathsf{snd}\, r \in \mathcal{B}\} \\
\mathcal{A} + \mathcal{B} & := {}^{\blacktriangleright}(\mathsf{inl}(\mathcal{A}) \cup \mathsf{inl}(\mathcal{B}) \cup \mathsf{SNe}) \\
1 & := {}^{\blacktriangleright}(\{()\} \cup \mathsf{SNe})
\end{array}$$

Note that $\mathsf{SAT_g}$ and $\mathsf{SAT_u}$ are closed under arbitrary intersections and *unions*. The last property is the advantage of saturated-sets semantics, it does not always hold for candidates of reducibility or biorthogonals, and even when it holds the proof is non-trivial [16].

If $\mathcal{F}$ is a monotone operator on sets of terms, and α an ordinal, we define the term sets $\mu^\alpha\mathcal{F}$ and $\nu^\alpha\mathcal{F}$ by iteration on α as follows.

$$\begin{array}{llll} \mu^0 & \mathcal{F} := \mathcal{N} & \nu^0 & \mathcal{F} := \mathcal{S} \\ \mu^{\alpha+1} & \mathcal{F} := \mathcal{F}(\mu^\alpha\mathcal{F}) & \nu^{\alpha+1} & \mathcal{F} := \mathcal{F}(\nu^\alpha\mathcal{F}) \\ \mu^\lambda & \mathcal{F} := \bigcup_{\alpha<\lambda}\mu^\alpha\mathcal{F} & \nu^\lambda & \mathcal{F} := \bigcap_{\alpha<\lambda}\nu^\alpha\mathcal{F} \end{array}$$

Herein, λ denotes limit ordinals > 0. Let ∞ denote the ordinal at which, for any $\mathcal{F}$, iteration from below reaches the least fixed-point $\mu^\infty\mathcal{F} = \mathcal{F}(\mu^\infty\mathcal{F})$, and iteration from above reaches the greatest fixed-point $\nu^\infty\mathcal{F} = \mathcal{F}(\nu^\infty\mathcal{F})$. Since term sets are countable, ∞ is at most the first uncountable ordinal.

Now if $\mathcal{F}(\mathcal{A})$ is guarded for any $\mathcal{A} \in \mathsf{SAT_u}$, then $\mu^\alpha\mathcal{F}$ will be guarded for $\alpha \geq 1$. If $\mathcal{F}(\mathcal{A})$ is guarded for any *guarded* $\mathcal{A}$, then $\nu^\alpha\mathcal{F}$ is guarded for all α.

Lemma 1 (Semantical formation). *The following implications, written as rules, hold:*

$$\frac{\mathcal{A},\mathcal{B} \in \mathsf{SAT_u}}{\mathcal{A}\star\mathcal{B} \in \mathsf{SAT_g}}\ \star \in \{\to,\times,+\} \qquad \frac{}{1 \in \mathsf{SAT_g}} \qquad \frac{}{\mathcal{N} \in \mathsf{SAT_u}} \qquad \frac{}{\mathcal{S} \in \mathsf{SAT_g}}$$

$$\frac{\mathcal{F} \in \mathsf{SAT_u} \to \mathsf{SAT}_b}{\mu^\infty\mathcal{F} \in \mathsf{SAT}_b}\ b \in \{\mathsf{u},\mathsf{g}\} \qquad \frac{\mathcal{F} \in \mathsf{SAT_g} \to \mathsf{SAT}_b}{\nu^\infty\mathcal{F} \in \mathsf{SAT}_b}\ b \in \{\mathsf{u},\mathsf{g}\}$$

Proof. We show the first implication, $\mathcal{A} \to \mathcal{B} \in \mathsf{SAT_g}$. It is sufficient to assume $\{x\} \subseteq \mathcal{A} \subseteq \mathsf{SN}$ and $\mathcal{B} \in \mathsf{SAT_u}$. Let $r \in \mathcal{A} \to \mathcal{B}$. First, $r\,x \in \mathcal{B} \subseteq \mathsf{SN}$ by assumption, hence $r \in \mathsf{SN}$. Second, we know that $r\,x$ weak-head reduces to either a neutral term or a value. Hence, either r weak-head reduces to a neutral term, or to a λ-abstraction, which is a value. Third, let $r' \blacktriangleright r$. Then for any $s \in \mathcal{A}$ we have $r'\,s \rhd r\,s$ which, since $\mathcal{B} \in \mathsf{SAT_u}$, implies $r'\,s \in \mathcal{B}$. This entails $r' \in \mathcal{A}$.

Lemma 2 (Semantical typing). *The following implications hold:*

$$\frac{[s/x]t \in \mathcal{B} \textit{ for all } s \in \mathcal{A}}{\lambda xt \in \mathcal{A} \to \mathcal{B}} \qquad \frac{r \in \mathcal{A} \to \mathcal{B} \qquad s \in \mathcal{A}}{r\,s \in \mathcal{B}}$$

$$\frac{r \in \mathcal{A} \qquad s \in \mathcal{B}}{\mathsf{pair}\,r\,s \in \mathcal{A}\times\mathcal{B}} \qquad \frac{r \in \mathcal{A}\times\mathcal{B}}{\mathsf{fst}\,r \in \mathcal{A}} \qquad \frac{r \in \mathcal{A}\times\mathcal{B}}{\mathsf{snd}\,r \in \mathcal{B}} \qquad \frac{}{() \in 1}$$

$$\frac{t \in \mathcal{A}}{\mathsf{inl}\,t \in \mathcal{A}+\mathcal{B}} \qquad \frac{t \in \mathcal{B}}{\mathsf{inr}\,t \in \mathcal{A}+\mathcal{B}} \qquad \frac{r \in \mathcal{A}+\mathcal{B} \qquad s \in \mathcal{A}\to\mathcal{C} \qquad t \in \mathcal{B}\to\mathcal{C}}{\mathsf{case}\,r\,s\,t \in \mathcal{C}}$$

Proof. The rules for λ, pair, and case are proven by closure of saturated sets under safe weak head expansion. (The remaining rules hold already by definition.) We show the last implication. Assume $r \in \mathcal{A}+\mathcal{B}$, then $r \blacktriangleright r'$ where r' is either neutral or a left or right injection. We observe that $\mathsf{case}\,r\,s\,t \rhd \mathsf{case}\,r'\,s\,t$ and distinguish the three cases: In the first case $\mathsf{case}\,r'\,s\,t \in \mathsf{SNe}$, hence, $\mathsf{case}\,r\,s\,t \in \mathcal{N} \subseteq \mathcal{C}$. In the second case, $r' = \mathsf{inl}\,r''$ with $r'' \in \mathcal{A}$, thus, $\mathsf{case}\,r\,s\,t \rhd s\,r'' \in \mathcal{C}$. The third case is analogous to the second.

The following semantical typing for recursion is the foundation of type-based termination à la Mendler [12], Amadio et al. [6] and Barthe et al. [7]. In a typical application of the following lemma, $\mathcal{I}(\alpha)$ will be some inductive type $\mu^\alpha \mathcal{F}$; then $\mathcal{I}(0) = \mathcal{N}$.

Lemma 3 (Recursion). *For all ordinals $\alpha \leq \infty$ let $\mathcal{I}(\alpha), \mathcal{C}(\alpha) \in \mathsf{SAT}_\mathsf{u}$ with $\mathcal{I}(0) \subseteq \mathcal{N}$. Set $\mathcal{A}(\alpha) := \mathcal{I}(\alpha) \to \mathcal{C}(\alpha)$ and stipulate continuity: $\bigcap_{\alpha<\lambda} \mathcal{A}(\alpha) \subseteq \mathcal{A}(\lambda)$ for all limit ordinals $\lambda > 0$. Then the following implication holds for all $\beta \leq \infty$:*

$$\frac{s \in \bigcap_{\alpha<\infty} \mathcal{A}(\alpha) \to \mathcal{A}(\alpha+1)}{\mathsf{fix}^\mu\, s \in \mathcal{A}(\beta)}.$$

Proof. By transfinite induction on β. The limit case is handled by the continuity condition on $\mathcal{A}$. For the other cases, assume $r \in \mathcal{I}(\beta)$ and show $\mathsf{fix}^\mu\, s\, r \in \mathcal{C}(\beta)$. If $r \in \mathcal{N}$ then $\mathsf{fix}^\mu\, s\, r \in \mathcal{N} \subseteq \mathcal{C}(\beta)$; since $\mathcal{I}(0) \subseteq \mathcal{N}$, this handles the case $\beta = 0$. Otherwise $r \rhd v$ and $\beta = \alpha + 1$ for some α. It is sufficient to show that the weak head reduct $s\,(\mathsf{fix}^\mu\, s)\, v$ of $\mathsf{fix}^\mu\, s\, r$ is in $\mathcal{C}(\alpha+1)$, but this follows from the induction hypothesis $\mathsf{fix}^\mu s \in \mathcal{A}(\alpha)$ by the assumption $s \in \mathcal{A}(\alpha) \to \mathcal{A}(\alpha+1)$.

The proof for $\beta = 0$ needs $\mathcal{N}$ to be closed under evaluation contexts, $\mathsf{fix}^\mu s\, \_$ in our case. If $\mathcal{N}$ was also guarded, then $\mathsf{fix}^\nu_0 \lambda\_x \in \mathcal{N}$ and $\mathsf{fix}^\mu(\lambda f f)(\mathsf{fix}^\nu_0 \lambda\_x) \in \mathcal{N}$, a diverging term. Thus, the least type needs to be classified as unguarded.

Remark 1 (Continuity). Let $\mathcal{N}at^\alpha = \mu^\alpha(\mathcal{X} \mapsto 1 + \mathcal{X})$ be the semantical type corresponding to the set of natural numbers $< \alpha$. The function $\mathcal{A}(\alpha) = (\mathcal{N}at^\omega \to \mathcal{N}at^\alpha) \to 1$ violates the continuity condition: one can implement a test $p(f)$ in our calculus that halts whenever it has found numbers n, m with $f(n) = f(m)$. The test will halt for bounded functions $f \in \mathcal{N}at^\omega \to \mathcal{N}at^\alpha$ for $\alpha < \omega$, but diverges on, for example, any strictly monotone unbounded function $f \in \mathcal{N}at^\omega \to \mathcal{N}at^\omega$. This justifies the necessity of the continuity condition for the soundness of our semantics [2].

The following lemma dualizes Lemma 3; it is tailored for guarded $\mathcal{C}(\alpha) = \nu^\alpha \mathcal{F}$. To prove it, we have introduced the concept of guardedness in the first place.

Lemma 4 (Corecursion). *For $\alpha \leq \infty$ let $\mathcal{B}_1(\alpha), \ldots, \mathcal{B}_n(\alpha) \in \mathsf{SAT}_\mathsf{u}$ and $\mathcal{C}(\alpha) \in \mathsf{SAT}_\mathsf{g}$ such that $\mathcal{S} \subseteq \mathcal{C}(0)$. Set $\mathcal{A}(\alpha) := \mathcal{B}_1(\alpha) \to \cdots \to \mathcal{B}_n(\alpha) \to \mathcal{C}(\alpha)$ and stipulate $\bigcap_{\alpha<\lambda} \mathcal{A}(\alpha) \subseteq \mathcal{A}(\lambda)$ for limits λ. Then for all $\beta \leq \infty$,*

$$\frac{s \in \bigcap_{\alpha<\infty} \mathcal{A}(\alpha) \to \mathcal{A}(\alpha+1)}{\mathsf{fix}^\nu_n\, s \in \mathcal{A}(\beta)}.$$

Proof. By transfinite induction on β, limits again handled by continuity of $\mathcal{A}$. Assume $t_i \in \mathcal{B}_i(\beta)$ for $i = 1..n$ and show $r := \mathsf{fix}^\nu_n\, s\, t_{1..n} \in \mathcal{C}(\beta)$. In case $\beta = 0$ it is sufficient to show $r \in \mathcal{S}$, but this holds since r is a value and its direct subterms are all s.n. In case $\beta = \alpha+1$, observe that $r \blacktriangleright s\,(\mathsf{fix}^\nu_n\, s)\, t_{1..n} \in \mathcal{C}(\alpha+1)$ by induction hypothesis $\mathsf{fix}^\nu_n\, s \in \mathcal{A}(\alpha)$ and assumption $s \in \mathcal{A}(\alpha) \to \mathcal{A}(\alpha+1)$. Since $\mathcal{C}(\alpha+1)$ is guarded, we are done.

We have identified semantically sound principles for recursion and corecursion. In the next sections, we implement two type systems on this basis.

3 A Basic Type System: Mendler (Co)Iteration

In this section, we consider a type system for iteration over equi-inductive types and coiteration over equi-coinductive types in the style of Mendler [12]. Mendler-iteration, like conventional iteration coming from initial algebra semantics, is usually formulated for iso-inductive types, with an explicit constructor $\mathsf{in} : F(\mu F) \to \mu F$. Our developments in the last section paved the way for equi-style formulations.

Types are given by the following grammar

$$\begin{array}{ll} \star & ::= \to \mid \times \mid + \\ A, B, C & ::= X \mid 1 \mid A \star B \mid \forall X A \mid \mu X A \mid \nu X A. \end{array}$$

The type constructors $\forall$, μ, and ν bind variable X in A. The type $\mu X X$ is an empty, unguarded type; we especially need to avoid unguarded coinductive types like $\nu Y \mu X X$. To this end, we present a kinding judgement with two base kinds: $*_{\mathsf{g}}$, guarded types, and $*_{\mathsf{u}}$, unguarded types.

Let θ be a map from type variables to semantical types. We define the semantics $[\![A]\!]_\theta$ of type A by recursion on A as follows:

$$\begin{array}{llll} [\![X]\!]_\theta & = \theta(X) & [\![\forall X A]\!]_\theta & = \bigcap_{\mathcal{X} \in \mathsf{SAT}_{\mathsf{u}}} [\![A]\!]_{\theta[X \mapsto \mathcal{X}]} \\ [\![A \star B]\!]_\theta & = [\![A]\!]_\theta \star [\![B]\!]_\theta & [\![\mu X A]\!]_\theta & = \mu^\infty(\mathcal{X} \in \mathsf{SAT}_{\mathsf{u}} \mapsto [\![A]\!]_{\theta[X \mapsto \mathcal{X}]}) \\ [\![1]\!]_\theta & = 1 & [\![\nu X A]\!]_\theta & = \nu^\infty(\mathcal{X} \in \mathsf{SAT}_{\mathsf{g}} \mapsto [\![A]\!]_{\theta[X \mapsto \mathcal{X}]}) \end{array}$$

Kinding. Let Δ be a finite map from type variables to base kinds. We write $\Delta, X : \kappa$ for the updated map Δ' with $\Delta'(X) = \kappa$ and $\Delta'(Y) = \Delta(Y)$ in case $Y \neq X$. In the update operation, we presuppose $X \notin \mathsf{dom}(\Delta)$. The judgment $\Delta \vdash A : \kappa$ is inductively given by the following rules (where $b \in \{\mathsf{u}, \mathsf{g}\}$).

$$\frac{}{\Delta \vdash X : \Delta(X)} \qquad \frac{}{\Delta \vdash 1 : *_{\mathsf{g}}} \qquad \frac{\Delta \vdash A : *_{\mathsf{g}}}{\Delta \vdash A : *_{\mathsf{u}}} \qquad \frac{\Delta \vdash A : *_{\mathsf{u}} \qquad \Delta \vdash B : *_{\mathsf{u}}}{\Delta \vdash A \star B : *_{\mathsf{g}}}$$

$$\frac{\Delta, X : *_{\mathsf{u}} \vdash A : *_b}{\Delta \vdash \forall X A : *_b} \qquad \frac{\Delta, X : *_{\mathsf{u}} \vdash A : *_b}{\Delta \vdash \mu X A : *_b}\ \mathit{pos} \qquad \frac{\Delta, X : *_{\mathsf{g}} \vdash A : *_{\mathsf{g}}}{\Delta \vdash \nu X A : *_{\mathsf{g}}}\ \mathit{pos}$$

In the formation rules for (co)inductive types we require (*pos*) that X appears only positively in A (otherwise, the denoted fixed-points might not exist).

The soundness of kinding is immediate. Let $\theta \in [\![\Delta]\!]$ iff $\Delta(X) = *_b$ implies $\theta(X) \in \mathsf{SAT}_b$ for all X.

Theorem 1 (Soundness of kinding). *If $\Delta \vdash A : *_b$ and $\theta \in [\![\Delta]\!]$ then $[\![A]\!]_\theta \in \mathsf{SAT}_b$.*

Type equality. Let $\Delta \vdash A = A'$ be the least congruence over the two axioms

$$\frac{\Delta \vdash \mu X A : *_{\mathsf{u}}}{\Delta \vdash \mu X A = [\mu X A / X] A} \qquad \frac{\Delta \vdash \nu X A : *_{\mathsf{g}}}{\Delta \vdash \nu X A = [\nu X A / X] A}.$$

Lemma 5 (Soundness of type substitution and equality)

1. $\llbracket [B/X]A \rrbracket_\theta = \llbracket A \rrbracket_{\theta[X \mapsto \llbracket B \rrbracket_\theta]}$.
2. If $\Delta \vdash A = A'$ *and* $\theta \in \llbracket \Delta \rrbracket$ *then* $\llbracket A \rrbracket_\theta = \llbracket A' \rrbracket_\theta$.

Typing. Let Γ be a finite map from type variables to kinds and term variables to types, with additional update operation $\Gamma, x\!:\!A$. Each Γ can be viewed as a Δ, by ignoring the term variable bindings. The typing judgement $\Gamma \vdash t : A$ is inductively given by the following rules:

$$\frac{\Gamma \vdash \Gamma(x) : *_{\mathsf{u}}}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma, x\!:\!A \vdash t : B}{\Gamma \vdash \lambda x t : A \to B} \qquad \frac{\Gamma \vdash r : A \to B \qquad \Gamma \vdash s : A}{\Gamma \vdash r\, s : B}$$

$$\frac{\Gamma, X\!:\!*_{\mathsf{u}} \vdash t : A}{\Gamma \vdash t : \forall X A} \qquad \frac{\Gamma \vdash t : \forall X A \qquad \Gamma \vdash B : *_{\mathsf{u}}}{\Gamma \vdash t : [B/X]A} \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A = B}{\Gamma \vdash t : B}$$

$$\frac{}{\Gamma \vdash c : \Sigma(c)} \qquad \frac{\Gamma \vdash \mu X A : *_{\mathsf{u}} \qquad \Gamma \vdash C : *_{\mathsf{u}}}{\Gamma \vdash \mathsf{fix}^\mu : (\forall X.\, (X \to C) \to A \to C) \to \mu X A \to C}$$

$$\frac{\Gamma \vdash \nu X A : *_{\mathsf{g}} \qquad \Gamma \vdash B_i : *_{\mathsf{u}} \text{ for } i = 1..n}{\Gamma \vdash \mathsf{fix}^\nu_n : (\forall X.\, (B_{1..n} \to X) \to B_{1..n} \to A) \to B_{1..n} \to \nu X A}$$

Herein, the signature Σ assigns the following types to constants c:

pair	$: \forall A \forall B.\, A \to B \to A \times B$	inl	$: \forall A \forall B.\, A \to A + B$
fst	$: \forall A \forall B.\, A \times B \to A$	inr	$: \forall A \forall B.\, B \to A + B$
snd	$: \forall A \forall B.\, A \times B \to B$	case	$: \forall A \forall B \forall C.\, A + B \to$
()	$: 1$		$(A \to C) \to (B \to C) \to C$

Example 1. If we drop the guardedness condition in the corecursion rule, then the diverging term $\mathsf{fix}^\mu(\lambda f f)\,(\mathsf{fix}^\nu_0 \lambda\_x)$ can be typed. First observe that $\mathsf{fix}^\mu \lambda f f : \mu X X \to C$ for any C. In the context $x : \mu X X$ we have $\lambda\_x : \forall Y.\, Y \to \mu X X$, hence, $\mathsf{fix}^\nu_0 \lambda\_x : \nu Y \mu X X$. With $\nu Y \mu X X = \mu X X$ we get the typing $x\!:\!\mu X X \vdash \mathsf{fix}^\mu(\lambda f f)\,(\mathsf{fix}^\nu_0 \lambda\_x) : C$. This demonstrates that guardedness is vital for the termination of open expressions when mixing recursion and corecursion. Nonemptiness is not necessary, however; an analogous term constructed with the empty, but guarded type $\nu Y.\, 1 \to \mu X X$ is not diverging.

Let θ now be a finite map from type variables to semantical types and from term variables to terms. We write $\theta \in \llbracket \Gamma \rrbracket$ if additionally to the condition on type variables $\theta(x) \in \llbracket \Gamma(x) \rrbracket_\theta$ for all term variables $x \in \mathsf{dom}(\Gamma)$. Let $t\theta$ denote the simultaneous (capture-avoiding) substitution of all $x \in \mathsf{FV}(t)$ by $\theta(x)$.

Theorem 2 (Soundness of typing). *If* $\Gamma \vdash t : A$ *and* $\theta \in \llbracket \Gamma \rrbracket$ *then* $t\theta \in \llbracket A \rrbracket_\theta$.

Proof. By induction on the typing derivation, using the result of the last section.

In case of fix^μ, assume $s \in \bigcap_{\mathcal{X} \in \mathsf{SAT}_{\mathsf{u}}} \llbracket (X \to C) \to A \to C \rrbracket_{\theta[X \mapsto \mathcal{X}]}$ and show $\mathsf{fix}^\mu\, s \in \llbracket \mu X A \to C \rrbracket_\theta$. Lemma 3 (recursion) is applicable with types $\mathcal{I}(\alpha) =$

$\mu^\alpha(\mathcal{X} \mapsto [\![A]\!]_{\theta[X\mapsto\mathcal{X}]})$ and $\mathcal{C}(\alpha) = [\![C]\!]_\theta$. Since $r \in \mathcal{I}(\lambda) = \bigcup_{\alpha<\lambda} \mathcal{I}(\alpha)$ implies $r \in \mathcal{I}(\alpha)$ for some $\alpha < \lambda$ and $\mathcal{C}$ does not depend on its ordinal argument, the continuity condition is trivially satisfied for $\mathcal{A}(\alpha) = \mathcal{I}(\alpha) \to \mathcal{C}(\alpha)$. For all α, the typing $s \in \mathcal{A}(\alpha) \to \mathcal{A}(\alpha+1)$ requested by the lemma is an instance of the given typing with $\mathcal{X} = \mathcal{I}(\alpha)$, since $[\![A]\!]_{\theta[X\mapsto\mathcal{I}(\alpha)]} = \mathcal{I}(\alpha+1)$.

In case of fix^ν, Lemma 4 is applicable, analogously to the case of fix^μ. The kinding ensures that $\mathcal{C}(\alpha) := \nu^\alpha(\mathcal{X} \mapsto [\![A]\!]_{\theta[X\mapsto\mathcal{X}]})$ is guarded for all $\alpha \leq \infty$. The continuity condition is again trivially satisfied.

Corollary 1 (Strong normalization and consistency). *Each typable term is strongly normalizing. Each closed well-typed term weak-head reduces to a value. No closed term inhabits $\forall X X$.*

Proof. By soundness of typing, letting $\theta(X) = \mathcal{N}$ for all type variables X and $\theta(x) = x$ for all term variables x. Consistency, the last statement, follows since there are no closed terms in $\mathcal{N}$.

Example: Stream Eating with Mendler (Co)Iteration

We first allow ourselves some syntactic sugar: we write (r, s) for $\mathsf{pair}\, r\, s$ and use matching abstraction $\lambda(x, y).t$ as a shorthand for $\lambda z.\, [\mathsf{fst}\, z/x][\mathsf{snd}\, z/y]t$. ML-style pattern matching $\mathsf{match}\ t\ \mathsf{with}\ p_i \mapsto t_i$ for patterns p_i composed from variables, (), pair, inl, and inr, can also be defined easily [5, Sec. 2.4].

To provide some help for type-checking (by the reader and by the machine), we sometimes will use Church-style syntax and allow type-annotations $t : A$ in the example programs:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash (t : A) : A} \qquad \frac{\Gamma, x{:}A \vdash t : B}{\Gamma \vdash \lambda x{:}A.\, t : A \to B}$$

$$\frac{\Gamma, X{:}*_{\mathsf{u}} \vdash t : A}{\Gamma \vdash \Lambda X t : \forall X A} \qquad \frac{\Gamma \vdash t : \forall X A \qquad \Gamma \vdash B : *_b}{\Gamma \vdash t[B] : [B/X]A}$$

Streams $\mathsf{Stream}\, A := \nu X.\, A \times X$ can be constructed by $\mathsf{pair} : \forall A.A \to \mathsf{Stream}\, A \to \mathsf{Stream}\, A$ and destructed by $\mathsf{fst} : \forall A.\, \mathsf{Stream}\, A \to A$ and $\mathsf{snd} : \forall A.\, \mathsf{Stream}\, A \to \mathsf{Stream}\, A$. In Haskell, stream processors are defined as a data type and the code of the mapping function is generally recursive.

```
data SP a b where
  get :: (a -> SP a b) -> SP a b
  put :: b -> SP a b -> SP a b

map :: (a -> b) -> SP a b
map f = get (\ a -> put (f a) (map f))
```

In our system, we define the type of codes for stream processing functions [10] as a interleaved coinductive-inductive type.

$$\mathsf{SP}\, A\, B := \nu X \mu Y.B \times X + (A \to Y)$$

The equi-style enables a direct representation of the constructors:

$$\begin{aligned} \mathsf{put} &:= \mathsf{inl} : \forall A \forall B.\, B \times \mathsf{SP}\,A\,B \to \mathsf{SP}\,A\,B \\ \mathsf{get} &:= \mathsf{inr} : \forall A \forall B.\, (A \to \mathsf{SP}\,A\,B) \to \mathsf{SP}\,A\,B \end{aligned}$$

The code of the stream-mapping function can be defined by Mendler coiteration as follows:

$$\begin{aligned} &\mathsf{map} : \quad \forall A \forall B.\, (A \to B) \to \mathsf{SP}\,A\,B \\ &\mathsf{map} := \Lambda A \Lambda B \lambda f : A \to B. \\ &\qquad \mathsf{fix}^{\nu}_{0}\, \Lambda X \lambda map : X.\, \mathsf{inr}\,(\lambda a : A.\, \mathsf{inl}\,(f\,a,\ map) : \mu Y.\, B \times X + (A \to Y)) \end{aligned}$$

Stream eating executes the code of a stream processor, consuming an input stream and producing an output stream. In Haskell it is again defined by general recursion:

```
eat :: SP a b -> [a] -> [b]
eat (get f) (a:as) = eat (f a) as
eat (put b t)  as  = b : eat t as
```

We define eating by an outer Mendler coiteration on the output stream and an inner Mendler iteration on the stream processor.

$$\begin{aligned} &\mathsf{eat} : \quad \mathsf{SP}\,A\,B \to \mathsf{Stream}A \to \mathsf{Stream}B \\ &\mathsf{eat} := \mathsf{fix}^{\nu}_{2}\, \Lambda X \lambda eat^{\nu} : \mathsf{SP}\,A\,B \to \mathsf{Stream}\,A \to X \\ &\qquad\quad \mathsf{fix}^{\mu}\, \Lambda Y \lambda eat^{\mu} : Y \to \mathsf{Stream}A \to B \times X \\ &\qquad\qquad \lambda t : B \times \mathsf{SP}\,A\,B + (A \to Y).\, \lambda(a, as).\, \mathsf{match}\ t\ \mathsf{with} \\ &\qquad\qquad\quad \mathsf{put}\,(b, t') \mapsto (b,\ eat^{\nu}\,t'\,(a, as)) : B \times X \\ &\qquad\qquad\quad \mathsf{get}\,f \qquad \mapsto eat^{\mu}\,(f\,a : Y)\ as \end{aligned}$$

Some interesting functions, like composition of stream processors, are not (co)iterative, hence cannot be defined directly in the present type systems. Therefore, we introduce a more expressive system of sized types in the next section.

4 A Fancy Type System: Sized Types

Sized types allow a greater flexibility in defining recursive and corecursive functions by mapping the semantics more directly into the syntax of types. In the following, we describe an extension of the type system F^{ω} that makes the following features of semantics available in syntax:

1. Ordinals a and approximations $\mu^a F$ and $\nu^a F$ of inductive and coinductive types. The syntax of ordinals will be restricted to variables, successor and ∞. There is no need to provide notation for limit ordinals.
2. Distinction between guarded ($*_{\mathsf{g}}$) and unguarded types ($*_{\mathsf{u}}$). This feature is new in comparison to previous works [2,8,9].
3. Monotonicity information (polarity) of type constructors. For instance, the function space constructor is antitone in its first argument and monotone in its second argument, thus, it receives kind $*_{\mathsf{u}} \overset{-}{\to} *_{\mathsf{u}} \overset{+}{\to} *_{\mathsf{g}}$. Using polarities, the positivity test for (co)inductive types scales to higher-orders [1].

Kinds classify type constructors. Besides $*_\mathsf{g}$ and $*_\mathsf{u}$ we introduce a kind ord_u of ordinals and a subkind $\mathsf{ord}_\mathsf{g} \leq \mathsf{ord}_\mathsf{u}$ of non-zero ordinals. Function kinds are annotated with a polarity p.

$$\begin{array}{lll}
p, q & ::= \circ & \text{mixed-variant (no monotonicity information)} \\
 & \mid + & \text{covariant (monotone)} \\
 & \mid - & \text{contravariant (antitone)} \\
 & \mid \top & \text{constant (both mono- and antitone)} \\
 & & \\
\kappa & ::= *_\mathsf{u} \mid *_\mathsf{g} \mid \mathsf{ord}_\mathsf{u} \mid \mathsf{ord}_\mathsf{g} & \text{base kind} \\
 & \mid \kappa \xrightarrow{p} \kappa' & \text{function kind}
\end{array}$$

Subkinding $\kappa \leq \kappa'$ is defined inductively by the following rules:

$$\frac{}{*_\mathsf{g} \leq *_\mathsf{u}} \qquad \frac{}{\mathsf{ord}_\mathsf{g} \leq \mathsf{ord}_\mathsf{u}} \qquad \frac{\kappa_1' \leq \kappa_1 \qquad p' \leq p \qquad \kappa_2 \leq \kappa_2'}{\kappa_1 \xrightarrow{p} \kappa_2 \leq \kappa_1' \xrightarrow{p'} \kappa_2'}$$

Herein, the order on polarities is the reflexive-transitive closure of the axioms $\circ \leq p$ and $p \leq \top$. If one composes a function in $\kappa_1 \xrightarrow{p} \kappa_2$ with a function in $\kappa_2 \xrightarrow{q} \kappa_3$ one obtains a function in $\kappa_1 \xrightarrow{pq} \kappa_3$. For the associative and commutative polarity composition pq we have the laws $\top p = \top$, $\circ p = \circ$ (for $p \neq \top$), $+p = p$, and $-- = +$. Inverse application $p^{-1}q$ of a polarity p to a polarity q is defined as the solution of

$$\forall q, q'.\ \ p^{-1}q \leq q' \iff q \leq pq'.$$

Type constructors F are type-level λ-terms over constants C:

$$\begin{array}{ll}
C & ::= \to \mid \times \mid + \mid 1 \mid \forall_\kappa \mid \mu \mid \nu \mid \mathsf{0} \mid \mathsf{s} \mid \infty \\
A, B, F, G & ::= C \mid X \mid \lambda X F \mid F\,G
\end{array}$$

We use $\to$, $\times$, $+$ infix and write $\forall X\!:\!\kappa. A$ for $\forall_\kappa \lambda X A$. If κ is $*_\mathsf{u}$, it can be dropped. We write the ordinal argument a to μ and ν superscript, e.g., $\mu^a F$.

Let Δ denote a finite map from type (constructor) variables X to pairs $p\kappa$ of a polarity p and a kind κ. Inverse application $p^{-1}\Delta$ of a polarity p to Δ is defined by $\Delta(X) = q\kappa \implies (p^{-1}\Delta)(X) = (p^{-1}q)\kappa$. The following kinding rules [1] and kind assignments to constants handle polarities properly:

$$\frac{C : \kappa}{\Delta \vdash C : \kappa} \qquad \frac{\Delta(X) = p\kappa \qquad p \leq +}{\Delta \vdash X : \kappa} \qquad \frac{\Delta, X\!:\!p\kappa \vdash F : \kappa'}{\Delta \vdash \lambda X F : \kappa \xrightarrow{p} \kappa'}$$

$$\frac{\Delta \vdash F : \kappa \xrightarrow{p} \kappa' \qquad p^{-1}\Delta \vdash G : \kappa}{\Delta \vdash F\,G : \kappa'} \qquad \frac{\Delta \vdash F : \kappa \qquad \kappa \leq \kappa'}{\Delta \vdash F : \kappa'}$$

$$\begin{array}{lll}
\mathsf{0} \ : \mathsf{ord}_\mathsf{u} & \to \ : *_\mathsf{u} \xrightarrow{-} *_\mathsf{u} \xrightarrow{+} *_\mathsf{g} & \forall_\kappa : (\kappa \xrightarrow{\circ} *_b) \xrightarrow{+} *_b \\
\mathsf{s} \ : \mathsf{ord}_\mathsf{u} \xrightarrow{+} \mathsf{ord}_\mathsf{g} & \times \ : *_\mathsf{u} \xrightarrow{+} *_\mathsf{u} \xrightarrow{+} *_\mathsf{g} & \mu \ : \mathsf{ord}_b \xrightarrow{+} (*_\mathsf{u} \xrightarrow{+} *_b) \xrightarrow{+} *_b \\
\infty : \mathsf{ord}_\mathsf{g} & + \ : *_\mathsf{u} \xrightarrow{+} *_\mathsf{u} \xrightarrow{+} *_\mathsf{g} & \nu \ : \mathsf{ord}_\mathsf{u} \xrightarrow{-} (*_\mathsf{g} \xrightarrow{+} *_\mathsf{g}) \xrightarrow{+} *_\mathsf{g} \\
 & 1 \ : *_\mathsf{g} &
\end{array}$$

These kindings express, for instance, that successor ordinals $\mathsf{s}\,a$ and the closure ordinal ∞ are "guarded" (i.e., non-zero), each of the proper constructions →, ×, +, and 1 produces guarded types, a universal type $\forall_\kappa \lambda X A$ is guarded if its body A is. Interesting is the kinding of inductive types: $\mu^a F$ is guarded if a is non-zero and $F\,X$ is guarded even for unguarded X. For example, $\mu^0 F$ and $\mu^a \lambda X X$ are always unguarded, $\mu^{\mathsf{s}\,a}\lambda X.\, 1 + X$ is always guarded. Finally, coinductive types $\nu^a F$ are always guarded, but they are only well-kinded if F maps guarded types to guarded types. Hence, the type $\nu^\infty \lambda X X$, which contains only the inhabitant $\mathsf{fix}^\nu_0 \lambda x x$, is allowed, but $\nu^a \lambda X.\mu^\infty \lambda Y Y$ is prohibited, and so is $\nu^a \lambda X.\, \mu^0 F$.

Type equality and subtyping. The judgement $\Delta \vdash F = F' : \kappa$ is the least congruence over the following axioms [1], including a subsumption rule:

$$\frac{\Delta, X : p\kappa \vdash F : \kappa' \qquad p^{-1}\Delta \vdash G : \kappa}{\Delta \vdash (\lambda X F)\, G = [G/X]F : \kappa'} \qquad \frac{\Delta \vdash F : p\kappa \to \kappa'}{\Delta \vdash \lambda X.\, F\, X = F : p\kappa \to \kappa'}\ X \notin \mathsf{FV}(F)$$

$$\frac{\Delta \vdash F : \top\kappa \to \kappa' \qquad \Delta \vdash G : \kappa \qquad \Delta \vdash G' : \kappa}{\Delta \vdash F\, G = F\, G' : \kappa'}$$

$$\frac{}{\Delta \vdash \mathsf{s}\,\infty = \infty : \mathsf{ord_g}} \qquad \frac{\Delta \vdash a : \mathsf{ord_u} \qquad b \in \{\mathsf{u}, \mathsf{g}\}}{\Delta \vdash \mu^{\mathsf{s}a} = \lambda F.\, F\,(\mu^a\, F) : (*_\mathsf{u} \xrightarrow{+} *_b) \xrightarrow{+} *_b}$$

$$\frac{\Delta \vdash a : \mathsf{ord_u}}{\Delta \vdash \nu^{\mathsf{s}a} = \lambda F.\, F\,(\nu^a\, F) : (*_\mathsf{g} \xrightarrow{+} *_\mathsf{g}) \xrightarrow{+} *_\mathsf{g}}$$

Subtyping $\Delta \vdash F \leq F' : \kappa$ is induced by axioms expressing relations between ordinals and equipped with congruence rules that respect polarities.

$$\frac{\Delta \vdash a : \mathsf{ord_u}}{\Delta \vdash 0 \leq a : \mathsf{ord_u}} \qquad \frac{\Delta \vdash a : \mathsf{ord}_b}{\Delta \vdash a \leq \mathsf{s}\,a : \mathsf{ord}_b} \qquad \frac{\Delta \vdash a : \mathsf{ord}_b}{\Delta \vdash a \leq \infty : \mathsf{ord}_b}$$

$$\frac{\Delta \vdash F \leq F' : \kappa \xrightarrow{p} \kappa' \qquad p^{-1}\Delta \vdash G : \kappa}{\Delta \vdash F\, G \leq F'\, G : \kappa'}$$

$$\frac{\Delta \vdash F : \kappa \xrightarrow{+} \kappa' \qquad \Delta \vdash G \leq G' : \kappa}{\Delta \vdash F\, G \leq F\, G' : \kappa'} \qquad \frac{\Delta \vdash F : \kappa \xrightarrow{-} \kappa' \qquad \Delta \vdash G' \leq G : \kappa}{\Delta \vdash F\, G \leq F\, G' : \kappa'}$$

Additionally, we have a congruence rule for λ-abstraction and rules for reflexivity, transitivity, antisymmetry, and subsumption. Typically, we will use subtyping to derive $\mu^a F \leq \mu^{\mathsf{s}\,a} F \leq \mu^\infty F$ and $\nu^\infty F \leq \nu^{\mathsf{s}\,a} F \leq \nu^a F$.

Kind interpretation. Kinds are interpreted as expected: $[\![*_\mathsf{u}]\!] = \mathsf{SAT_u}$, $[\![*_\mathsf{g}]\!] = \mathsf{SAT_g}$, $[\![\mathsf{ord_u}]\!] = \{\alpha \mid 0 \leq \alpha \leq \infty\}$, $[\![\mathsf{ord_g}]\!] = \{\alpha \mid 0 < \alpha \leq \infty\}$, and $[\![\kappa \xrightarrow{p} \kappa']\!]$ is the space of p-variant operators from $[\![\kappa]\!]$ to $[\![\kappa']\!]$. For base kinds κ_0 let $\mathcal{A} \sqsubseteq_{\kappa 0} \mathcal{A}'$ hold iff $\mathcal{A} \subseteq \mathcal{A}'$. For higher kinds, let $\mathcal{F} \sqsubseteq_{\kappa \xrightarrow{p} \kappa'} \mathcal{F}'$ iff $\mathcal{F}(\mathcal{G}) \sqsubseteq_{\kappa'} \mathcal{F}'(\mathcal{G})$ for all $\mathcal{G} \in [\![\kappa]\!]$. With these definitions, we can set

$$[\![\kappa \xrightarrow{p} \kappa']\!] = \{\mathcal{F} \in [\![\kappa]\!] \to [\![\kappa']\!] \mid \mathcal{F}(\mathcal{G}) \sqsubseteq \mathcal{F}(\mathcal{G}') \text{ for all } \mathcal{G} \sqsubseteq^p \mathcal{G}' \in [\![\kappa]\!]\}.$$

Herein, $\sqsubseteq^{+}$ denotes $\sqsubseteq$, $\sqsubseteq^{-}$ denotes $\sqsupseteq$, $\sqsubseteq^{\circ}$ denotes equality, and $\mathcal{G} \sqsubseteq^{\top} \mathcal{G}'$ always holds.

Lemma 6 (Soundness of subkinding). *If $\kappa \leq \kappa'$ then $[\![\kappa]\!] \subseteq [\![\kappa']\!]$.*

Type interpretation. We interpret the type constants C as follows:

$$\begin{array}{lllll}
[\![0]\!] & = 0 & [\![\forall_\kappa]\!](\mathcal{F}) = \bigcap_{\mathcal{G} \in [\![\kappa]\!]} \mathcal{F}(\mathcal{G}) & & \\
[\![\mathsf{s}]\!](\infty) & = \infty & [\![C]\!] \quad\;\; = C & & \text{for } C \in \{\to, \times, +, 1, \mu, \nu\} \\
[\![\mathsf{s}]\!](\alpha < \infty) & = \alpha + 1 & & & \\
[\![\infty]\!] & = \infty & & &
\end{array}$$

This interpretation can be lifted to an interpretation $[\![F]\!]_\theta$ of well-kinded constructors F. We let $\theta \sqsubseteq \theta' \in [\![\Delta]\!]$ if $\theta(X) \sqsubseteq^{p} \theta'(X) \in [\![\kappa]\!]$ for all $(X : p\kappa) \in \Delta$.

Theorem 3 (Soundness of kinding, equality, and subtyping). *Let $\theta \sqsubseteq \theta' \in [\![\Delta]\!]$.*

1. *If $\Delta \vdash F : \kappa$ then $[\![F]\!]_\theta \sqsubseteq [\![F]\!]_{\theta'} \in [\![\kappa]\!]$.*
2. *If $\Delta \vdash F = F' : \kappa$ then $[\![F]\!]_\theta \sqsubseteq [\![F']\!]_{\theta'} \in [\![\kappa]\!]$.*
3. *If $\Delta \vdash F \leq F' : \kappa$ then $[\![F]\!]_\theta \sqsubseteq [\![F']\!]_{\theta'} \in [\![\kappa]\!]$.*

Typing. The rules for λ-abstraction, application, basic constants c remain in place. The type conversion rule is replaced by a subsumption rule, and the generalization and instantiation rules for universal types are now higher-kinded.

$$\frac{\Gamma \vdash \Gamma(x) : *_{\mathsf{u}}}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A \leq B : *_{\mathsf{u}}}{\Gamma \vdash t : B}$$

$$\frac{\Gamma, X : \kappa \vdash t : F\,X}{\Gamma \vdash t : \forall_\kappa F}\; X \notin \mathsf{FV}(F) \qquad \frac{\Gamma \vdash t : \forall_\kappa F \qquad \Gamma \vdash G : \kappa}{\Gamma \vdash t : F\,G}$$

$$\frac{\Gamma \vdash F : *_{\mathsf{u}} \xrightarrow{+} *_{\mathsf{u}} \qquad \Gamma \vdash G : \mathsf{ord}_{\mathsf{u}} \xrightarrow{\circ} *_{\mathsf{u}} \qquad \Gamma \vdash a : \mathsf{ord}_{\mathsf{u}}}{\Gamma \vdash \mathsf{fix}^{\mu} : (\forall \imath\!:\!\mathsf{ord}_{\mathsf{u}}.\, (\mu^{\imath} F \to G\,\imath) \to \mu^{\mathsf{s}\,\imath} F \to G(\mathsf{s}\,\imath)) \to \mu^{a} F \to G\,a}\; adm^{\mu}$$

$$\frac{\Gamma \vdash F : *_{\mathsf{g}} \xrightarrow{+} *_{\mathsf{g}} \qquad \Gamma \vdash G_i : \mathsf{ord}_{\mathsf{u}} \xrightarrow{\circ} *_{\mathsf{g}} \text{ for } i = 1..n \qquad \Gamma \vdash a : \mathsf{ord}_{\mathsf{u}}}{\Gamma \vdash \mathsf{fix}^{\nu}_{n} : (\forall \imath\!:\!\mathsf{ord}_{\mathsf{u}}.\, (G_{1..n}\,\imath \to \nu^{\imath} F) \to G_{1..n}\,(\mathsf{s}\,\imath) \to \nu^{\mathsf{s}\,\imath} F) \to G_{1..n}\,a \to \nu^{a} F}\; adm^{\nu}$$

In the recursion rule, the side condition adm^{μ} needs to ensure that the type $\lambda\imath.\, \mu^{\imath} F \to G\,\imath$ is continuous in the sense of Lemma 3. Systematic criteria have been developed based on a saturated-set semantics in the context of iso-(co)inductive types [2], and these criteria are directly applicable for the equi-setting described in this article. Due to space restrictions, we only give a sound approximation here: There must be an $n \geq 0$, $\Gamma \vdash F_i : *_{\mathsf{u}} \xrightarrow{+} *_{\mathsf{u}}$ for $i = 1..n$ and $\Gamma \vdash B : \mathsf{ord}_{\mathsf{u}} \xrightarrow{+} *_{\mathsf{u}}$ such that $\Gamma \vdash G\,\imath = \mu^{\imath} F_1 \to \cdots \to \mu^{\imath} F_n \to B\,\imath : *_{\mathsf{u}}$.

For the criterion adm^{ν} we give the following sound approximation: For each $j = 1..n$, either $\Gamma \vdash G_j : \mathsf{ord}_{\mathsf{u}} \xrightarrow{-} *_{\mathsf{u}}$, or there exists $\Gamma \vdash F_j : *_{\mathsf{u}} \xrightarrow{+} *_{\mathsf{u}}$ such that $\Gamma \vdash G_j\,\imath = \mu^{\imath} F_j : *_{\mathsf{u}}$.

Theorem 4 (Soundness of typing). *If $\Gamma \vdash t : A$ and $\theta \in [\![\Gamma]\!]$ then $t\theta \in [\![A]\!]_\theta$.*

Proof. By induction on the typing derivation. The connection of the typing rules for recursion and corecursion to lemmata 3 and 4 is now immediate.

Corollary 2 (Strong normalization and consistency). *Each typable term is strongly normalizing. Each closed well-typed term weak-head reduces to a value. No closed term inhabits $\forall X X$.*

Example: Composition of Stream Processors

The sized type system encompasses a number of recursion schemes: primitive recursion, Mendler (co)recursion, course-of-value recursion, and indirect recursion (where the recursive arguments are obtained via another function, like the filtering function in case of quicksort). In the following, we implement composition comp of stream processors such that $\mathsf{eat}\,(\mathsf{comp}\; t_1\, t_2) = \mathsf{eat}\, t_2 \circ \mathsf{eat}\, t_1$. There are two possible implementations for the case that t_1 wants to read an element and t_2 wants to output one. We give the latter priority and arrive at the following Haskell code:

```
comp :: SP a b -> SP b c -> SP a c
comp  t1      (put c t2) = put c (comp t1 t2)
comp (put b t1) (get f2) = comp t1 (f2 b)
comp (get f1)        t2  = get (\ a -> comp (f1 a) t2)
```

We express SP through sized types and define two useful approximations of this type.

$$\begin{array}{lcl}
\mathsf{SP}\, A\, B & := & \nu^\infty \lambda X.\, \mu^\infty \lambda Y.\, B \times X + (A \to Y) \\[1ex]
\mathsf{SP}^\imath\, A\, B & := & \nu^\imath \lambda X.\, \mu^\infty \lambda Y.\, B \times X + (A \to Y) \\
\mathsf{SP}^{\mathsf{s}\,\imath}\, A\, B & = & B \times \mathsf{SP}^\imath\, A\, B + (A \to \mathsf{SP}^{\mathsf{s}\,\imath}\, A\, B) \\
\mathsf{put} & : & B \times \mathsf{SP}^\imath\, A\, B \to \mathsf{SP}^{\mathsf{s}\,\imath}\, A\, B \\
\mathsf{get} & : & (A \to \mathsf{SP}^{\mathsf{s}\,\imath}\, A\, B) \to \mathsf{SP}^{\mathsf{s}\,\imath}\, A\, B \\[1ex]
\mathsf{SP}_\jmath\, A\, B & := & \mu^\jmath \lambda Y.\, B \times \mathsf{SP}\, A\, B + (A \to Y) \\
\mathsf{SP}_{\mathsf{s}\,\jmath}\, A\, B & = & B \times \mathsf{SP}\, A\, B + (A \to \mathsf{SP}_\jmath\, A\, B) \\
\mathsf{get} & : & (A \to \mathsf{SP}_\jmath\, A\, B) \to \mathsf{SP}_{\mathsf{s}\,\jmath}\, A\, B \\
\mathsf{put} & : & B \times \mathsf{SP}_\infty\, A\, B \to \mathsf{SP}_{\mathsf{s}\,\jmath}\, A\, B
\end{array}$$

We will use the derived types of the constructors put and get below. Note the asymmetry between $\mathsf{SP}^\imath$ and $\mathsf{SP}_\jmath$, which shows in the last type of put.

In our analysis, $\mathsf{comp}\, t_1\, t_2$ is defined by corecursion into $\mathsf{SP} A\, C$ using a lexicographic recursion on (t_2, t_1). It is conveniently coded with a generalized recursor fix^μ_n, which recurses on the $n+1$st argument and is definable from fix^μ [3].

$$\begin{array}{l}
\mathsf{comp} \\
\quad :\;\; \mathsf{SP}\, A\, B \to \mathsf{SP}\, B\, C \to \mathsf{SP}\, A\, C \\
\quad := \mathsf{fix}^\nu_2 \Lambda\imath.\, \lambda comp^\nu : \mathsf{SP}\, A\, B \to \mathsf{SP}\, B\, C \to \mathsf{SP}^\imath\, A\, C. \\
\qquad\quad \mathsf{fix}^\mu_1 \Lambda\jmath.\, \lambda comp^\mu_1 : \mathsf{SP}\, A\, B \to \mathsf{SP}_\jmath\, B\, C \to \mathsf{SP}^{\mathsf{s}\,\imath} A\, C. \\
\qquad\qquad \mathsf{fix}^\mu_0 \Lambda k.\, \lambda comp^\mu_2 : \mathsf{SP}_k\, A\, B \to \mathsf{SP}_{\mathsf{s}\,\jmath}\, B\, C \to \mathsf{SP}^{\mathsf{s}\,\imath} A\, C.
\end{array}$$

$$\begin{array}{l}
\lambda t_1 : \mathsf{SP}_{\mathsf{s}\,k}\, A\, B.\, \lambda t_2 : \mathsf{SP}_{\mathsf{s}\,\jmath}\, B\, C.\, \mathsf{match}\ t_2\ \mathsf{with} \\
\quad \begin{array}{ll}
\mathsf{put}\,(c,\, t_2' : \mathsf{SP}\, B\, C) & \mapsto \mathsf{put}\,(c,\, \mathit{comp}^{\nu}\, t_1\, t_2') : \mathsf{SP}^{\mathsf{s}\,\imath}\, A\, C \\
\mathsf{get}\,(f_2 : B \to \mathsf{SP}_{\jmath}\, B\, C) & \mapsto \mathsf{match}\ t_1\ \mathsf{with}
\end{array} \\
\qquad \begin{array}{ll}
\mathsf{put}\,(b,\, t_1' : \mathsf{SP}\, A\, B) & \mapsto \mathit{comp}_1^{\mu}\, t_1'\, (f_2\, b) : \mathsf{SP}^{\mathsf{s}\,\imath}\, A\, C \\
\mathsf{get}\,(f_1 : A \to \mathsf{SP}_k\, A\, B) & \mapsto \mathsf{get}\,(\lambda a.\, \mathit{comp}_2^{\mu}\, (f_1\, a)\, t_2) : \mathsf{SP}^{\mathsf{s}\,\imath}\, A\, C
\end{array}
\end{array}$$

In the corecursive call to comp^{ν}, the first argument is casted from $\mathsf{SP}_{\mathsf{s}\,k}\, A\, B$ to $\mathsf{SP}_{\infty}\, A\, B$ using subtyping of inductive types. Such a cast is not available in Mendler iteration, but could be simulated with Mendler recursion. Hence, comp is a mixed coiterative/recursive/recursive function.

5 Conclusions

We have presented a construction of saturated sets for equi-inductive and coinductive types and derived two type systems which guarantee termination of recursion and corecursion under lazy unfolding. In contrast to candidates of reducibility or biorthogonals, saturated sets are closed under unions, hence, the continuity criteria for sized iso-(co)inductive types developed in previous work [2] are directly transferable to the equi-setting.

We have given two type systems for terminating (co)recursion in the presence of equi-(co)inductive types and showed by some examples that they handle mixed inductive/coinductive types properly. The system of sized types is ready for extension to higher-kinded (co)inductive types.

Although the operational semantics of corecursive values in the equi-setting suggests a semantics using biorthogonals, we have succeeded to apply a modification of the saturated sets approach. This substantiated the conjecture Colin Riba made to me, namely, *biorthogonals are only required to justify languages inspired by classical logic.*

Acknowledgments. The idea how to formulate guarded saturated sets came to me during a visit to the Protheo team at LORIA, Nancy in February 2007. Thanks to Frédéric Blanqui and Colin Riba for the invitation. I am also grateful to Thorsten Altenkirch who communicated the type of stream processors to me. Thanks to the anonymous referees for their valuable feedback.

References

1. Abel, A.: Polarized subtyping for sized types. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 381–392. Springer, Heidelberg (2006)
2. Abel, A.: Semi-continuous sized types and termination. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 72–88. Springer, Heidelberg (2006)
3. Abel, A.: Towards generic programming with sized types. In: Uustalu, T. (ed.) MPC 2006. LNCS, vol. 4014, pp. 10–28. Springer, Heidelberg (2006)
4. Abel, A.: Strong normalization and equi-(co)inductive types. In: Ronchi Della Rocca, S. (ed.) TLCA 2007. LNCS, vol. 4583, pp. 8–22. Springer, Heidelberg (2007)
5. Abel, A., Matthes, R., Uustalu, T.: Iteration schemes for higher-order and nested datatypes. Theor. Comput. Sci. 333, 3–66 (2005)

6. Amadio, R.M., Coupet-Grimal, S.: Analysis of a guard condition in type theory (extended abstract). In: Nivat, M. (ed.) ETAPS 1998 and FOSSACS 1998. LNCS, vol. 1378, pp. 48–62. Springer, Heidelberg (1998)
7. Barthe, G., Frade, M.J., Giménez, E., Pinto, L., Uustalu, T.: Type-based termination of recursive definitions. Math. Struct. in Comput. Sci. 14, 1–45 (2004)
8. Barthe, G., Grégoire, B., Pastawski, F.: CICˆ: Type-based termination of recursive definitions in the Calculus of Inductive Constructions. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 257–271. Springer, Heidelberg (2006)
9. 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.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 105–119. Springer, Heidelberg (2006)
10. Ghani, N., Hancock, P., Pattinson, D.: Continuous functions on final coalgebras. Electr. Notes in Theor. Comp. Sci. 164, 141–155 (2006)
11. Hughes, J., Pareto, L., Sabry, A.: Proving the correctness of reactive systems using sized types. In: POPL 1996, pp. 410–423 (1996)
12. Mendler, N.P.: Inductive types and type constraints in the second-order lambda calculus. Annals of Pure and Applied Logic 51, 159–172 (1991)
13. Parigot, M.: Recursive programming with proofs. Theor. Comput. Sci. 94, 335–356 (1992)
14. Parigot, M.: Proofs of strong normalization for second order classical natural deduction. The Journal of Symbolic Logic 62, 1461–1479 (1997)
15. Raffalli, C.: Data types, infinity and equality in system AF_2. In: Meinke, K., Börger, E., Gurevich, Y. (eds.) CSL 1993. LNCS, vol. 832, pp. 280–294. Springer, Heidelberg (1994)
16. Riba, C.: On the stability by union of reducibility candidates. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 317–331. Springer, Heidelberg (2007)
17. Swierstra, W.: I/O in a dependently typed programming language. Talk presented at TYPES 2007 (2007)