Tarski's fixed-point theorem and lambda calculi with monotone inductive types

Ralph Matthes*
matthes@informatik.uni-muenchen.de
Lehr- und Forschungseinheit für theoretische Informatik
Institut für Informatik der Universität München, Germany

December 7, 1999

Abstract

The new concept of lambda calculi with monotone inductive types is introduced by help of motivations drawn from Tarski's fixed-point theorem (in preorder theory) and initial algebras and initial recursive algebras from category theory. They are intended to serve as formalisms for studying iteration and primitive recursion on general inductively given structures. Special accent is put on the behaviour of the rewrite rules motivated by the categorical approach, most notably on the question of strong normalization (i. e., the impossibility of an infinite sequence of successive rewrite steps). It is shown that this key property hinges on the concrete formulation. The canonical system of monotone inductive types, where monotonicity is expressed by a *monotonicity witness* being a term expressing monotonicity through its type, enjoys strong normalization shown by an embedding into the traditional system of non-interleaving positive inductive types which, however, has to be enriched by the parametric polymorphism of system F. Restrictions to iteration on monotone inductive types already embed into system F alone, hence clearly displaying the difference between iteration and primitive recursion with respect to algorithms despite the fact that, classically, recursion is only a concept derived from iteration.

1 Introduction and Overview

The aim is to generalize the notion of function defined by iteration or primitive recursion on $\mathbb{N}$ in the direction of more complex inductively generated

*I am thankful for support by the Volkswagenstiftung.

structures than the naturals. As a secondary aim, the difference between iteration and primitive recursion shall be elucidated.

Our generalization draws its motivation from Tarski's theorem guaranteeing for any given monotone function on a complete lattice an associated structure which allows reasoning by (generalized) induction. However, Tarski's result has no direct connection to the notion of algorithm (as is required for studying iteration and primitive recursion). As a constructive refinement on the way to a syntactical analysis of algorithms, we review initial algebras and initial recursive algebras with respect to some functor on a category. To that aim, a minimal background on category theory is provided. Those semantical structures may be more easily studied by formalisms having only the bare bones of the semantical structures. The formalisms at hand are lambda calculi with inductive types. Again, a minimal background on lambda calculi will be given. The first inductive types come by examples which all stick to the traditional paradigm of positivity: They are structures where the monotonicity of the induced operator on the complete lattice (à la Tarski) may be simply read off the shape of the inductive type (which hence is a trivial syntactic check).

However, it is also possible to introduce systems having inductive types without that severe syntactic restriction. In order to formulate them, on has to introduce parametric polymorphism, i. e., universally quantified types. By their help, monotonicity of the type ρ with respect to the type variable α may be expressed as the type

$$\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta].$$

If one writes $\rho(\alpha)$ for ρ and $\rho(\beta)$ for $\rho[\alpha := \beta]$, this may be interpreted that for any α and β, whenever α is "contained" in β, then $\rho(\alpha)$ is "contained" in $\rho(\beta)$.[1] The known systems of inductive types had syntactic restrictions on ρ and α which always guaranteed that closed terms of type $\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$ existed whenever ρ and α were used to introduce an inductive type.

The new system of monotone inductive types allows any ρ and α to give rise to an inductive type.

However, the behaviour of iteration and primitive recursion are controlled by arbitrary terms of type $\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$ which need not exist beforehand but may already use iteration and primitive recursion without restriction. Moreover, those *monotonicity witnesses* need

[1] Note that this suggests two different readings of the same $\to$: implication and containment.

not be closed. *A priori*, this approach gives much more freedom, and really seems to be the most far-reaching generalization of lambda calculus (without dependent types) possible in the spirit of Tarski's theorem.

However, if we restrict the system of monotone inductive types to allow iteration only, we can embed it into system F (which is the basic system of impredicative polymorphism without any inductive types). Therefore, it is crucial to have primitive recursion in the system. Nevertheless, also the full system embeds into the system of non-interleaving positive inductive types showing that all the subtleties which were added to that relatively simple system do not even add algorithmic power.

Note, that the system of positive inductive types had to be extended by the capabilities of system F in order to make the embedding possible. Hence, it is already fully impredicative and therefore still inaccessible to proof theoretic analysis. The definition and analysis of monotone inductive types without system F is still an area full of open questions.

In section 2 we state Tarski's theorem, give an example with natural numbers and introduce "extended induction" which will later be the motivation for primitive recursion. Section 3 defines categories, functors and initial algebras for functors. Then a lambda calculus is developed to cover those notions syntactically. We leave the problem open how to model the morphism part of the functors. In section 4 examples are given which show how rich even non-interleaving positive inductive types are, and even more those with interleaving. Also system F is defined and a first solution is given to the mentioned problem of finding the action of types on morphisms. In section 5 primitive recursion as opposed to iteration is introduced. Section 6 proceeds with original material from [Mat98] spelt out in a more transparent manner. First the reference system of monotone inductive types is given, then variants are dealt with showing that reasonable formulations may lead to non-termination due to a lack of uniformity in the monotonicity witness, finally it is explained how an easy interpolation argument leads to the embedding of the term rewrite system of monotone inductive types into the (a priori much smaller) system of non-interleaving positive inductive types. In section 7 an attempt is made to relate this all to other research.

In principle, the presentation is self-contained: Categories, functors, initial algebras, lambda calculus and inductive types are defined. Since most of the structures are intended as a motivation for the definitions in section 6, no results are proved. Also in that section 6 we do not aim at proving results in full detail. This has all been done in my Ph.D. thesis [Mat98] which, however, never tried to be readable to readers outside the field.

2 Tarski's Fixed Point Theorem

We formulate only the part of Tarski's theorem [Tar55] which will be used later.

Theorem 1 (Tarski 1939) *Let $(U, \subseteq, \wedge, \vee)$ be a complete lattice of sets (subsets of some fixed set) with $\subseteq$ being set inclusion, and $\Phi : U \to U$ be monotone. Then*

$$\mu\Phi := \bigwedge\{S \mid \Phi(S) \subseteq S\} \quad \textit{(the infimum of the pre-fixed-points)}$$

is a pre-fixed-point of Φ, i. e. $\Phi(\mu\Phi) \subseteq \mu\Phi$.

The set $\mu\Phi$ is called the set which is inductively generated by Φ.

Remark that we do not need U to be formed of sets with set inclusion as partial order, and we even have that $\mu\Phi$ is a fixed point of Φ.

As an example consider $U := \mathcal{P}(\mathbb{R})$ with settheoretic infimum and supremum, and with

$$\Phi(S) := \{0\} \cup \{r + 1 \mid r \in S\}.$$

Obviously, Φ is monotone. Set $\mathbb{N} := \mu\Phi$. By the theorem, $0 \in \mathbb{N}$ and for all $r \in \mathbb{N}$, also $r + 1 \in \mathbb{N}$.

We now want to show that $r \in \mathbb{N} \Rightarrow r^2 \in \mathbb{N}$. We try to use the fact that $\mathbb{N}$ is the infimum of all pre-fixed-points of Φ. Set $S := \{r|r^2 \in \mathbb{N}\}$. If we manage to show that $\Phi(S) \subseteq S$, then we may infer $\mathbb{N} \subseteq S$. Clearly, $0 \in S$. Now let $r \in S$. We have to show that $r + 1 \in S$, i.e. $(r + 1)^2 = r^2 + r + r + 1 \in \mathbb{N}$. We may assume that we proved beforehand that addition does not lead out of $\mathbb{N}$. We already know that $r^2 \in \mathbb{N}$. But we do not know whether $r \in \mathbb{N}$!

As a remedy to this failure of proof consider $S' := \mathbb{N} \cap S$ and show $\Phi(S') \subseteq S$ because then due to Tarski's theorem and the monotonicity of Φ we also get $\Phi(S') \subseteq S'$ and therefore $\mathbb{N} \subseteq S'$ which in turn implies $\mathbb{N} \subseteq S$. It is clear that $\Phi(S') \subseteq S$ holds because we now have $r \in \mathbb{N}$ in the crucial proof step.

It should be remarked that mathematicians usually take the second approach when showing a statement by induction. They do it intuitively and often even when the first approach would be more to the point—a fact hidden for mathematical induction because the natural numbers are commonly seen as given in total and not cut out of some larger universe of objects. It will become important for the aim of this work to make a difference.

Remind the following three properties of $\mu\Phi$:

(μ-I) $\Phi(\mu\Phi) \subseteq \mu\Phi$.

(μ-E) $\Phi(S) \subseteq S \Rightarrow \mu\Phi \subseteq S$.

(μ-E$^+$) $\Phi(\mu\Phi \wedge S) \subseteq S \Rightarrow \mu\Phi \subseteq S$.

The last rule shall be called "extended induction" and may easily be inferred from (μ-I), (μ-E) and the monotonicity of Φ like in the example of natural numbers. (μ-E) is the rule of induction.

3 Initial Algebras and Iteration

In the previous section we argued strictly classically. Let us now look more closely at our relation $\subseteq$. We want to add information why $\subseteq$ holds by adding witnesses of $\subseteq$. This may be seen as the generalization of the theory of preorders to category theory (in the sense of Eilenberg and Mac Lane, first published in 1942 [EL42][2]): We do not only have a morphism from some object A to an object B or do not have it, but consider in general a class of morphisms from A to B. Instead of least pre-fixed-points of some monotone "operator" initial algebras w. r. t. some functor are studied.

A category consists of

- objects
- morphisms from some object to some object
- composition of morphisms

having several properties:

- The class of morphisms from an object A to A itself always contains a morphism called the identity on A.
- Composition respects types, i. e., one may compose morphisms iff the source object of the first morphism g is the same as the target object of the second morphism f, and the result $g \circ f$ is a morphism from the source object of f to the target object of g.
- Composition is associative and the identities are neutral elements with respect to composition (as long as composition with them is defined).

[2] With respect to category theory, the present article is self-contained. For a textbook on category theory see [Lan98].

In case there is at most one morphism from any object to any object, this gives precisely the preorders (= reflexive and transitive binary relations) on the objects (the identities correspond to reflexivity, composability to transitivity).

A functor is

- a mapping from some category to a category,
- mapping objects to objects and morphisms to morphisms

and fulfilling several compatibility properties:

- A morphism is mapped to a morphism from the mapped source object to the mapped target object.
- Identities are mapped to identities.
- The composition of morphisms is mapped to the composition of the mapped morphisms.

In case of preorders this concept reduces to the monotone mappings (reflecting only the first compatibility property).

Instead of least pre-fixed-points of monotone operators we now study initial algebras of endofunctors. Let F be an endofunctor on some category. An initial F-algebra is a pair (M, c) with M an object of the category and c a morphism from $F(M)$ to M such that for any other F-algebra, i. e. any pair (S, s) with S an object and s a morphism from $F(S)$ to S there is a unique morphism Es from M to S such that $Es \circ c = s \circ F(Es)$, where the morphism part of F is applied to Es. The last equation may be depicted via the commuting diagram in figure 1. Note that this equation trivially holds in case of a preorder.

$$\begin{tikzcd}
F(M) \arrow[r, "c"] \arrow[d, "F(Es)"'] & M \arrow[d, "Es"] \\
F(S) \arrow[r, "s"'] & S
\end{tikzcd}$$

Figure 1: Initial F-Algebra

Earlier we had $\Phi(\mu\Phi) \subseteq \mu\Phi$, now we have a morphism from $F(M)$ to M, hence c is the witness for the validity of (μ-I). Earlier we had to prove

$\Phi(S) \subseteq S$ in order to conclude $\mu\Phi \subseteq S$, now we have to provide a morphism s from $F(S)$ to S in order to get a morphism from M to S: Es.

Our aim is the syntactic study of initial algebras via systems of terms with rewrite rules. We have to model the categorical notions such as morphism and composition. We use the lambda calculus as basic formalism for dealing with function(al)s.

Instead of objects we simply have types which are nothing but strings of symbols built up from symbols for variables by help of type formers—most notably the binary $\to$ allowing to form the type $\rho \to \sigma$ from types ρ and σ. The type $\rho \to \sigma$ should be thought of as the type of functions from ρ to σ, i. e. of functionals whose arguments have type ρ and whose results have type σ. "Functionals" is the preferred name for those intuitive objects because the arguments may already be functions (and even functionals).

Instead of morphisms from A to B we study terms of type $\rho \to \sigma$ where terms of any type ρ should be thought of as denoting functionals of type ρ. Terms are again only syntactic objects. We have

- typed variables, denoted like x^ρ,
- lambda abstraction $\lambda x^\rho r^\sigma : \rho \to \sigma$ (if r already has type σ; after a colon or as a superscript we indicate the type of the term) modelling the functional $x \mapsto r(x)$ in general mathematical language ($r(x)$ is r but with the dependency on x indicated, hence it is only metasyntactically blown up.), in other words: lambda abstraction is functional abstraction,
- and application $r^{\rho\to\sigma} s^\rho : \sigma$ modelling application of a function taking arguments of type ρ to some argument of that type. (In general mathematics one would perhaps prefer to write $r(s)$.) The identity of the argument type of the functional r and the type of the argument s is *the* restriction imposed on term formation by the type system.

At present there is no life in the system. Life is added by term rewriting: Clearly, if one applies $x \mapsto r(x)$ to s, the result is $r(s)$. In the term system this is modelled by the rule of beta reduction[3] which allows to **replace** $(\lambda xr)s$ by $r[x := s]$ which is the term r after replacing any occurrence of the free variable x in r by s. We not only allow to replace the whole term in this fashion (which will be indicated by the binary relation $\rhd$, hence being defined by $(\lambda xr)s \rhd r[x := s]$) but may do the replacement to some subterm of

[3]It is a *reduction* towards a term which cannot be rewritten any further (a normal form) due to strong normalization mentioned below.

the term under consideration, and arrive at the binary one step relation $\to$ on terms. It is well-known that for simply-typed lambda calculus (only type variables and $\to$) those replacements cannot go on forever—independent of the strategy of replacement in case of choice. This fact is known as strong normalization of simply typed lambda calculus[4]. Together with confluence (which means that if one has a choice in replacement then both possible outcomes of replacement can be joined again by some further replacements) normalization implies that the least equivalence relation making a term and its replacement equivalent is decidable. This equivalence is called beta equality.[5]

Let us model initial F-algebras syntactically [Geu92]. Fix a type ρ and some variable α which might occur in ρ. The mapping $\alpha \mapsto \rho$ shall take the position of the functor F. (Its action on types/objects is given by substitution, its action on terms of function type/morphisms will be discussed in the next section.) Instead of the object M we include the inductive type $\mu\alpha\rho$ in our type system the interpretation of which is the least pre-fixed-point of $\alpha \mapsto \rho$. First look at figure 2.

$$\begin{tikzcd}
\rho[\alpha := \mu\alpha\rho] \arrow[r, "\mathrm{C}_{\mu\alpha\rho}\cdot"] \arrow[d, "(\alpha\mapsto\rho)(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu s)"'] & \mu\alpha\rho \arrow[d, "\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu s"] \\
\rho[\alpha := \sigma] \arrow[r, "s"'] & \sigma
\end{tikzcd}$$

Figure 2: Inductive Type with Iteration

The morphism part of the initial F-algebra is replaced by a term formation rule:

(μ-I) If t is a term of type $\rho[\alpha := \mu\alpha\rho]$, then $\mathrm{C}_{\mu\alpha\rho}t$ is a term of type $\mu\alpha\rho$ (in the diagram we used another mathematical means of expressing lambda abstraction: the dot).

[4] Without types normalization does not hold as may be seen from the generic example $(\lambda x.xx)(\lambda x.xx) \triangleright (xx)[x := \lambda x.xx] \equiv (\lambda x.xx)(\lambda x.xx)$ which even loops.

[5] There is a huge body of knowledge on first-order term rewrite systems not involving any binding of variables. The lambda calculi studied in this article heavily rest on the beta rule relating variable binding and substitution which makes them higher-order term rewrite systems. However, there is also a more subtle notion of higher-order term rewriting using lambda calculus as a framework, and consequently formulating the rewrite rules only up to beta (and eta) equality. See [MN98] for confluence and [vdPS95] for normalization of systems of this kind.

Weak[6] initiality is expressed by a term formation rule and a term rewrite rule:

(μ-E) If s is a term of type $\rho[\alpha := \sigma] \to \sigma$ and r is a term of type $\mu\alpha\rho$, then $r\mathrm{E}_\mu s$ is a term of type σ. (In the diagram, we use lambda abstraction instead of the notation $.\mathrm{E}_\mu s$ with the dot as means of abstraction.)

(β_μ) The new rewrite rule (added to beta reduction for $\to$):

$$(\mathrm{C}_{\mu\alpha\rho} t^{\rho[\alpha:=\mu\alpha\rho]})\mathrm{E}_\mu s \rhd s\Big((\alpha \mapsto \rho)(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu s)t\Big).$$

($r \rhd r'$ means that a subterm r may be replaced by r' in order to carry out one rewrite step $\to$.) It will later become clear that (β_μ) expresses iteration on $\mu\alpha\rho$.

This is not yet a proper definition unless we answer the following question: What is the counterpart of the functor on morphisms in our term world? More precisely, we have to give the definition of $(\alpha \mapsto \rho)(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu s)$ which has to be a term of type $\rho[\alpha := \mu\alpha\rho] \to \rho[\alpha := \sigma]$.

4 Positive Inductive Types

In order to formulate interesting examples which help us in finding the definition of $(\alpha \mapsto \rho)(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu s)$ we first extend the type system by the type 1 and sum types, and the term system by the appropriate introduction and elimination rules for those type constructs. The rewrite rules are extended accordingly.[7]

We include 1 in our type system which serves as a singleton set, and also $\rho + \sigma$ modelling the disjoint sum for any two types ρ and σ. The only inhabitant of 1 will be IN1: Add the term IN1 of type 1 (as an initial rule of the term system). It makes no sense to give an elimination rule for 1 and hence there there is also no beta reduction rule for 1. Inhabitants of type $\rho + \sigma$ are left injections of inhabitants of type ρ or right injections of inhabitants of type σ. This gives rise to the following term formation rules:

- If r is a term of type ρ, then $\mathrm{INL}_\sigma r$ is a term of type $\rho + \sigma$.

[6] We do not require uniqueness.

[7] Clearly, 1 and sum types are the counterparts to the categorical notions of terminal object and coproduct, respectively [Geu92]. Finite sums would be finite coproducts and the empty sum be an initial object (typically represented by 0). Terminal objects are empty products. Binary products are needed for the definition of primitive recursion in section 5.

- If r is a term of type σ, then $\mathrm{INR}_\rho r$ is a term of type $\rho + \sigma$.

Elimination of terms of sum type is done by case analysis written as a term formation rule:

- If r is a term of type $\rho + \sigma$ and s and t are terms of type τ, then $r(x^\rho.s, y^\sigma.t)$ is a term of type τ. (The dot serves as an indication of binding, i. e., x and y are bound variables.)

The meaning of these rules is displayed by the beta reduction rules for sums:

- $(\mathrm{INL}_\sigma r)(x^\rho.s, y^\sigma.t) \triangleright s[x := r]$.
- $(\mathrm{INR}_\rho r)(x^\rho.s, y^\sigma.t) \triangleright t[y := r]$.

Let us consider several examples of inductive types: First define the type of natural numbers by $\mathrm{nat} := \mu\alpha.1 + \alpha$. Our introduction rule gives us a term of type nat for any term of type $1 + \mathrm{nat}$. Thus zero and the successor are put together in this rule. We get them isolated by defining:

- $0 := \mathrm{C}_{\mathrm{nat}}(\mathrm{INL}_{\mathrm{nat}}\mathrm{IN1}) : \mathrm{nat}$.
- $\mathrm{S} := \lambda x^{\mathrm{nat}}.\mathrm{C}_{\mathrm{nat}}(\mathrm{INR}_1 x) : \mathrm{nat} \to \mathrm{nat}$.

Likewise, a term s of type $1 + \sigma \to \sigma$ in the elimination rule pairs up the initial value of the iteration and the step function. We again isolate them by:

- $s_0 := s(\mathrm{INL}_\sigma\mathrm{IN1}) : \sigma$.
- $s_{\mathrm{S}} := \lambda x^\sigma.s(\mathrm{INR}_1 x) : \sigma \to \sigma$.

We now want to have that $0\mathrm{E}_\mu s$ rewrites to s_0, and $(\mathrm{S}t)\mathrm{E}_\mu s$ and $s_{\mathrm{S}}(t\mathrm{E}_\mu s)$ rewrite to the same term, so that E_μ behaves (equationally) as an iterator on naturals meaning that $(\underbrace{\mathrm{S}\dots(\mathrm{S}}_{n}\,0)\dots)\mathrm{E}_\mu s$ and $\underbrace{s_{\mathrm{S}}(\dots(s_{\mathrm{S}}}_{n}\,s_0)\dots)$ are equal in some sense (here: that they have a common reduct). An easy calculation shows that this is achieved by setting

$$(\alpha \mapsto 1{+}\alpha)(\lambda x^{\mathrm{nat}}.x\mathrm{E}_\mu s) := \lambda z^{1+\mathrm{nat}}.z\Big(y^1.\mathrm{INL}_\sigma y, y^{\mathrm{nat}}.\mathrm{INR}_1((\lambda x^{\mathrm{nat}}.x\mathrm{E}_\mu s)y)\Big).$$

This will be an instance of the general method indicated below. $\alpha \mapsto 1 + \alpha$ is monotone in quite trivial a sense: It is even strictly positive, i. e., α occurs only strictly positively (never to the left side of a $\to$) in $1 + \alpha$.

Now consider the type cont $:= \mu\alpha.1 + ((\alpha \to \mathrm{nat}) \to \mathrm{nat})$ which may be seen as the type of continuations for programs whose results are natural numbers.[8] Let us ignore the tasks of isolation dealt with in the previous example. Then we have as constructors of type cont the terms $\mathrm{D} : \mathrm{cont}$ and $\mathrm{C} : ((\mathrm{cont} \to \mathrm{nat}) \to \mathrm{nat}) \to \mathrm{cont}$. We would like to have an inhabitant e of type cont $\to$ nat which completes in some sense the continuation it gets as an argument. It is specified as ($=$ shall denote the equivalence imposed by rewriting):

- $e\mathrm{D} = 0$.
- $e(\mathrm{C}f) = fe$ for any term f of type $(\mathrm{cont} \to \mathrm{nat}) \to \mathrm{nat}$.

The trivial continuation is "completed" to give the trivial result 0 and a continuation given by some f representing a functional which calculates numbers given some function of type cont $\to$ nat, i. e., some "completion function", is completed simply by applying this f to the "completion function" e being constructed. We see that the recursive call is by no means to e with some term smaller than $\mathrm{C}f$ in any sense. The term e which represents the function about to be defined is even fed in as an argument to the term f. But nevertheless this is an instance of iteration on cont if one defines $(\alpha \mapsto 1 + ((\alpha \to \mathrm{nat}) \to \mathrm{nat}))r^{\rho\to\sigma}$ for arbitrary terms $r : \rho \to \sigma$ as

$$\lambda x^{1+(\rho\to\mathrm{nat})\to\mathrm{nat}}.x(y^1.\mathrm{INL}_{(\sigma\to\mathrm{nat})\to\mathrm{nat}}y, y^{(\rho\to\mathrm{nat})\to\mathrm{nat}}.\mathrm{INR}_1(\lambda z^{\sigma\to\mathrm{nat}}.y(\lambda z_1^\rho.z(rz_1)))).$$

Later we cite a result that the definition given as above, but read as a term rewrite system (from left to right) is strongly normalizing. Note that cont is a nested inductive type in the sense that the inductive type nat appears in its definition. Note also that for cardinality reasons there cannot be a set-theoretic model interpreting the function type as the full set-theoretic function space.[9]

Nevertheless, the monotonicity of $\alpha \mapsto 1 + ((\alpha \to \mathrm{nat}) \to \mathrm{nat}))$ is seen quite easily: α occurs only positively in $1 + ((\alpha \to \mathrm{nat}) \to \mathrm{nat}))$ because every occurrence is to the left of an even number of $\to$. Clearly, this type is not strictly positive.

A third example is given by the ρ-branching well-founded trees (for arbitrary type ρ)

$$\mathrm{tree}(\rho) := \mu\beta.1 + (\rho \to \beta).$$

[8] Taken from an unpublished manuscript by Martin Hofmann.

[9] Of course, the recursion-theoretic model of lambda calculus allows to define the function e the totality of which may then be proved. There is a wealth of models for types such as cont studied in the field of domain theory.

(β is assumed not to be free in ρ.) The dependency on β is strictly positive.

As a final example consider the highly branching trees after U. Berger:

$$\mathrm{Tree} := \mu\alpha.1 + (\mathrm{tree}(\alpha) \to \alpha)$$

consists of the well-founded trees whose branching degree is tree(Tree), i. e., the whole type of well-founded trees branching over Tree about to be defined! Monotonicity is still seen by one glance at the definition: α occurs only positively in $1 + (\mathrm{tree}(\alpha) \to \alpha)$ (because α occurs negatively in $\mathrm{tree}(\alpha)$). Because the fixed-point is formed with respect to the variable α occurring free (as a parameter) of the type $\mathrm{tree}(\alpha)$ showing up in the definition of Tree, the latter type is called an interleaving non-strictly positive inductive type.

Lemma 1 *If $\mu\alpha\rho$ is only formed in case α occurs in ρ only positively (and ρ obeys hereditarily to this restriction), then there are closed terms*

$$\mathrm{map}_{\alpha\mapsto\rho,\tau_1,\tau_2} : (\tau_1 \to \tau_2) \to \rho[\alpha := \tau_1] \to \rho[\alpha := \tau_2].$$

This is well-known [Lei90, How92]. For a careful definition yielding normal terms map see [Mat98, 5.1.1]. As long as there is no interleaving of inductive types (which rules out Tree) the definition is quite straightforward. Interleaving requires to use (μ-I) and (μ-E) in the definition of map.[10] Now we may set for those types:

$$(\alpha \mapsto \rho)(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu s) := \mathrm{map}_{\alpha\mapsto\rho,\mu\alpha\rho,\sigma}(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu s).$$

Theorem 2 *The term rewrite system of positive inductive types is strongly normalizing.*

Proof Reduction-preserving embedding into Girard's [Gir72] (and Reynolds' [Rey74]) system F.

System F is the polymorphic lambda calculus. We do not have inductive types but universal types $\forall\alpha\rho$ (for any type ρ and any type variable α). The corresponding term formation rules are as follows:

- If r is a term of type ρ, then $\Lambda\alpha r$ is a term of type $\forall\alpha\rho$, provided that α does not occur free in a type of a variable free in r (well motivated by the Curry-Howard-correspondence [How80] with second-order propositional logic).

[10] Nevertheless there is not much freedom left in the definition. See the discussion at the very end of the article.

- If r is a term of type $\forall\alpha\rho$, then $r\sigma$ is a term of type $\rho[\alpha := \sigma]$.

The beta reduction rule for universal types: $(\Lambda\alpha r)\sigma \rhd r[\alpha := \sigma]$.

It was Girard who found a normalization proof for system F which is the impredicative variant of the Tait computability method [Tai67]. Since Girard's proof numerous proof variants have been published. They are inherently difficult because their means of proof have to go beyond second-order arithmetic. Nevertheless, there are quite short, modular and clear proofs available.[11]

Let an **embedding** be a translation of a term rewrite system to another, where the types and terms are translated in a compatible fashion such that any rewrite step in the source system gives rise to at least one rewrite step between the translated terms in the target system.

The most important fact concerning embeddings for our purposes is the preservation of strong normalization: If the target system of an embedding is strongly normalizing, so is the source system.

The proof that our system of positive inductive types embeds into system F may be done as usual. (Note however, that most authors do not really care about the simulation of rewrite steps and are content with preservation of beta equality and even neglect the need for lemmas on interchangeability of the translation function with substitution. For a careful presentation see [Mat98].)

5 Primitive Recursion

Full primitive recursion is more than mere iteration as given by (β_μ). E.g., in Gödel's T (for a thorough presentation see [GLT89]) which has primitive recursion on natural numbers we may also access the previous number argument as an argument to the step function calculating the value of the function(al) for some number argument[12]: For any terms $s_0 : \sigma$ and $s_{\mathrm{S}} : \mathrm{nat} \to \sigma \to \sigma$ there is a term $F : \mathrm{nat} \to \sigma$ (with free variables among those of s_0 and s_{S}) such that $F0 = s_0$ and $F(St) = s_{\mathrm{S}}t(Ft)$ ($=$ denotes the equality relation induced by the rewrite rules of Gödel's T).[13] Of course, it

[11] For an introduction to this subject see [GLT89].

[12] A typical example would be the factorial function $! : \mathbb{N} \to \mathbb{N}$ specified by $0! := 1$ and $(n+1)! := n! \cdot (n+1)$.

[13] Note that σ may be an arbitrarily complex type, hence giving rise to a wider class of representable functions than only the classical primitive recursive functions. By taking $\sigma := \mathrm{nat} \to \mathrm{nat}$ one can already define Ackermann's function which therefore is primitive recursive in our sense.

is possible to represent functions defined by primitive recursion by functions defined by iteration only. However, the behaviour of this encoding w. r. t. rewriting is not satisfactory: It does only work for numerals (instead of arbitrary terms of type nat) and e. g. the encoded predecessor function needs a linear number of steps and not only a constant number.[14]

In general, primitive recursion directly corresponds to "extended induction" as defined in section 2: $\Phi(\mu\Phi \wedge S) \subseteq S \Rightarrow \mu\Phi \subseteq S$. This principle has been derived from ordinary induction plus monotonicity plus the Tarski theorem. The derivation might be cast in the term calculus but again has an unsatisfactory rewriting behaviour (the problems with Gödel's T mentioned above are nothing but an instance of this phenomenon). Therefore, we include the counterpart of "extended induction" as a new primitive. The formation of the binary infimum will be modelled by the categorical product and by product types: A product of the objects A and B of some category consists of some object $A \times B$ and (projection) morphisms $\pi_1^{A,B}$ from $A \times B$ to A and $\pi_2^{A,B}$ from $A \times B$ to B such that for any object C and morphisms l from C to A and r from C to B there is a unique morphism $\langle l, r\rangle$ from C to $A \times B$ such that $\pi_1^{A,B} \circ \langle l, r\rangle = l$ and $\pi_2^{A,B} \circ \langle l, r\rangle = r$. Therefore, we extend our lambda calculus by types $\rho \times \sigma$ for any types ρ and σ and term formation rules as follows:

- If r is a term of type ρ and s is a term of type σ, then $\langle r, s\rangle$ is a term of type $\rho \times \sigma$.
- If r is a term of type $\rho \times \sigma$, then $r\mathrm{L}$ is a term of type ρ and $r\mathrm{R}$ is a term of type σ.

The categorical equalities become beta reduction rules: $\langle r, s\rangle\mathrm{L} \rhd r$ and $\langle r, s\rangle\mathrm{R} \rhd s$.

On the categorical side we now require that our initial F-algebra is also an initial recursive F-algebra [Geu92]: For any object S and morphism s from $F(M \times S)$ to S there is a unique morphism $\mathrm{E}^+ s$ from M to S such that $\mathrm{E}^+ s \circ c = s \circ F(\langle \mathrm{Id}, \mathrm{E}^+ s\rangle)$ where Id is the identity morphism on M. The equality is depicted in figure 3.

On the level of terms this gives a new formation rule:

- If r is a term of type $\mu\alpha\rho$ and s is a term of type $\rho[\alpha := \mu\alpha\rho \times \sigma] \to \sigma$, then $r\mathrm{E}_\mu^+ s$ is a term of type σ.

The new rewrite rule is read off the diagram in figure 4.

[14]In [SU] the main theorem is that there cannot be any reasonable encoding of inductive types into system F having a constant time predecessor. As a consequence, primitive recursion cannot be simulated by iteration in any reasonable encoding.

$$
\begin{tikzcd}
F(M) \arrow[r, "c"] \arrow[d, "F(\langle \mathrm{Id}, E^+ s\rangle)"'] & M \arrow[d, "E^+ s"] \\
F(M \times S) \arrow[r, "s"'] & S
\end{tikzcd}
$$

Figure 3: Initial Recursive F-Algebra

$$
\begin{tikzcd}
\rho[\alpha := \mu\alpha\rho] \arrow[r, "\mathrm{C}_{\mu\alpha\rho}\cdot"] \arrow[d, "(\alpha\mapsto\rho)(\langle \mathrm{Id}, \lambda x^{\mu\alpha\rho}.x\mathrm{E}^+_\mu s\rangle)"'] & \mu\alpha\rho \arrow[d, "\lambda x^{\mu\alpha\rho}.x\mathrm{E}^+_\mu s"] \\
\rho[\alpha := \mu\alpha\rho \times \sigma] \arrow[r, "s"'] & \sigma
\end{tikzcd}
$$

Figure 4: Inductive Type with Primitive Recursion

If we restrict ourselves to positive inductive types as in lemma 1 we may now give the rule (β^+_μ) of primitive recursion as follows:

$$(\mathrm{C}_{\mu\alpha\rho} t)\mathrm{E}^+_\mu s \rhd s\Big(\mathrm{map}_{\alpha\mapsto\rho,\mu\alpha\rho,\mu\alpha\rho\times\sigma}\Big(\lambda x^{\mu\alpha\rho}.\langle x, (\lambda x^{\mu\alpha\rho}.x\mathrm{E}^+_\mu s)x\rangle\Big)t\Big)$$

Theorem 3 *The extended system of positive inductive types with iteration*[15] *and primitive recursion is strongly normalizing.*

Proof E. g. by an extension of an appropriate proof for system F. (One may also show—cf. [Mat98, 4.2.2]—an embedding into monotone inductive types explained in the next section and infer strong normalization from that system.)

It is preferable (and will be assumed in the sequel) that system F's universal type quantification be included into the system of positive inductive types. It is easy to see that lemma 1 may then be expressed more neatly by the existence of closed terms $\mathrm{map}_{\alpha\mapsto\rho}$ of type $\forall\alpha\forall\beta.(\alpha\to\beta)\to\rho\to\rho[\alpha:=\beta]$. (Simply take $\Lambda\alpha\Lambda\beta.\mathrm{map}_{\alpha\mapsto\rho,\alpha,\beta}$ and also consider the case of universal quantification in the proof of that lemma.) The term $\mathrm{map}_{\alpha\mapsto\rho}$ obviously witnesses the monotonicity of $\alpha\mapsto\rho$ as displayed by its type. And this monotonicity

[15] As remarked earlier, (μ-E) appears in the definition of the terms map in case of interleaving inductive types, hence we need both iteration and recursion in our system. Still, it is possible to formulate a system without iteration [Mat98, 5.1.3]. One would hardly like to use it.

is derived from the syntactic requirement of positivity in the formation of $\mu\alpha\rho$. This directly leads us to the generalization studied in the next section.

6 Monotone Inductive Types

The observation central to this article:[16]

Strong normalization of systems of inductive types does not depend on a specific definition of the monotonicity witnesses $\mathrm{map}_{\alpha\mapsto\rho}$.

And further: The witness even need not be closed (*conditional monotonicity*). It moreover may be nothing but a variable (*hypothetical monotonicity*). However, in these cases we have to include the witnesses in our term formation rules (*attached witnesses of monotonicity*) such that the rewrite rules of iteration and primitive recursion have access to them.

6.1 A System of Monotone Inductive Types

It is like the extension of system F by positive inductive types, but with the following specification:

- $\mu\alpha\rho$ may be formed without any restriction.
- The introduction rule for terms of type $\mu\alpha\rho$ is changed to: If m is a term of type $\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$ and t is a term of type $\rho[\alpha := \mu\alpha\rho]$, then $\mathrm{C}_{\mu\alpha\rho}mt$ is a term of type $\mu\alpha\rho$.
- The formation rules concerning E_μ and E_μ^+ are unchanged.
- Replace in both rewrite rules (pertaining to iteration and primitive recursion) the term $\mathrm{map}_{\alpha\mapsto\rho,\mu\alpha\rho,\tau}$ by $m(\mu\alpha\rho)\tau$ (for the types $\tau = \sigma$ and $\tau = \mu\alpha\rho \times \sigma$, respectively).

As mentioned above, the hypothesized monotonicity as expressed by an arbitrary term of type $\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$ has to be accessible in the beta (i. e., iteration and primitive recursion) rules for $\mu\alpha\rho$. Here, we attach the monotonicity witness to the constructor $\mathrm{C}_{\mu\alpha\rho}$.

Theorem 4 *This term rewrite system is strongly normalizing.*

Proof Without (β_μ^+) this could be done by an embedding into system F. The full system may be treated by a technically demanding extension of an appropriate normalization proof for system F [Mat98, 9.5.1]. (Below we will see that an easy embedding suffices to establish the result.)

[16] A precise formulation is given by the systems ESMIT and ISMIT in [Mat98].

6.2 Other Systems of Monotone Inductive Types

Normalization of the previous system does not carry over to arbitrary other reasonable definitions of systems of monotone inductive types even with iteration only. We consider two modifications with the original introduction rule (if $t : \rho[\alpha := \mu\alpha\rho]$, then $\mathrm{C}_{\mu\alpha\rho}t : \mu\alpha\rho$) but with witnesses of some weakened forms of monotonicity in the elimination rule for iteration. The choices are:

- $m : \forall\beta.(\mu\alpha\rho \to \beta) \to \rho[\alpha := \mu\alpha\rho] \to \rho[\alpha := \beta]$
- $m : \forall\alpha.(\alpha \to \sigma) \to \rho \to \rho[\alpha := \sigma]$

In the first case, we are content with the first argument being fixed to $\mu\alpha\rho$, in the second case, we fix the second argument to the type σ of the functional defined by iteration on $\mu\alpha\rho$.

In both cases we get as term formation rule: If t is a term of type $\mu\alpha\rho$, m is given as above and s is a term of type $\rho[\alpha := \sigma] \to \sigma$, then $r\mathrm{E}_\mu ms$ is a term of type σ. We may now easily give modified iteration rules (because in the original system we always only used $\mathrm{map}_{\alpha\mapsto\rho,\mu\alpha\rho,\sigma}$, hence with first argument $\mu\alpha\rho$ and second argument σ)[17]:

- $(C_{\mu\alpha\rho}t)\mathrm{E}_\mu ms \rhd s\Big(m\sigma(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu ms)t\Big)$
- $(C_{\mu\alpha\rho}t)\mathrm{E}_\mu ms \rhd s\Big(m(\mu\alpha\rho)(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu ms)t\Big)$

Lemma 2 *One of the systems is strongly normalizing (even with primitive recursion added), the other not even weakly normalizing, i. e., there are terms of the other system which cannot be rewritten to a normal form (a term which cannot be rewritten any further).*

Proof Again e. g. by an intricate extension of the normalization proof for system F in the positive case [Mat98, 9.5.4][18], and by a counterexample in the negative case.

The counterexample will now be given for the first system: Set $\rho := \alpha$, choose arbitrary $t^{\mu\alpha\alpha}$ and σ and set $s := \lambda y^\sigma y$. Define

$$m := \Lambda\beta\lambda z^{\mu\alpha\alpha\to\beta}\lambda u^{\mu\alpha\alpha}.z(\mathrm{C}_{\mu\alpha\alpha}u).$$

[17]Therefore, one could even think of a version with monotonicity witness of type $(\mu\alpha\rho \to \sigma) \to \rho[\alpha := \mu\alpha\rho] \to \rho[\alpha := \sigma]$ which by the next lemma would also induce a non-normalizing system.

[18]The main theorem below will even cover this situation, hence there is in fact no need for that intricate extension.

m is even closed. If one left out $\mathrm{C}_{\mu\alpha\alpha}$ of the definition of m, we would have in essence the canonical monotonicity witness guaranteed by lemma 1, and we would no longer profit from the special setting of the first argument.

$(\mathrm{C}_{\mu\alpha\alpha}t)\mathrm{E}_\mu ms$ rewrites in two steps to $m\sigma(\lambda x^{\mu\alpha\alpha}.x\mathrm{E}_\mu ms)t$ and in three more steps to $(\lambda x^{\mu\alpha\alpha}.x\mathrm{E}_\mu ms)(\mathrm{C}_{\mu\alpha\alpha}t)$, hence in a total of six steps to the initial term. This gives rise to an infinite loop which cannot be avoided by some other strategy of choosing the subterms to replace.

What is the reason for such an unfortunate behaviour of the first system? It is a lack of uniformity which is exploited in the definition of the monotonicity witness m shown above. The canonical monotonicity witness would essentially do the same at any type, i. e., it would be *parametric*, the present m only works for the type $\mu\alpha\alpha$ (as first argument).

Only recently there has even been made a proposal for a non-normalizing extension of system F by some non-parametric rewrite rule for terms of an already inhabited type of system F [HM99]: Set

$$\sigma := \forall\alpha\forall\beta.(\alpha \to \alpha) \to (\beta \to \beta).$$

Trivially, σ is inhabited by the closed term $t := \Lambda\alpha\Lambda\beta\lambda x^{\alpha\to\alpha}\lambda y^\beta.y$. A new constant J' of type σ is added to system F along with the following rewrite rule: $J'\rho\rho r^{\rho\to\rho} \rhd r$ for any type ρ. Clearly, this definition exploits the equality of both type arguments. Our term t would behave quite differently: $t\rho\rho r^{\rho\to\rho}$ rewrites in three steps to $\lambda y^\rho.y$ and does not profit from the identity of the type arguments. The innocent-looking rule for J'—after all, it is just a projection—also gives rise to an infinite loop: Set $\rho := \forall\alpha.\alpha \to \alpha$ and $\omega := \Lambda\alpha.J'\rho\alpha(\lambda x^\rho.x\rho x)$ of type ρ. Then $\omega\rho\omega$ rewrites to $J'\rho\rho(\lambda x^\rho.x\rho x)\omega$ by beta reduction for universal types. The J'-rule applies and yields $(\lambda x^\rho.x\rho x)\omega$ which in turn rewrites to the initial term $\omega\rho\omega$.

I guess, the second variation on monotone inductive types discussed above is well-behaved simply because it is impossible to make non-parametric use of the additional freedom given by the less uniform type of the required monotonicity witnesses.

6.3 Embedding Monotone Inductive Types into Positive Inductive Types

Recall that we call types "non-interleaving" if every subexpression $\mu\alpha\rho$ has no subexpression of the form $\mu\beta\sigma$ with α free in $\mu\beta\sigma$.

Theorem 5 (Main Theorem) *The system of monotone inductive types with (full) primitive recursion and iteration defined in 6.1 embeds into the* *extension of system F by non-interleaving positive inductive types with (full) primitive recursion and iteration*[19].

Note that the reverse embedding is quite easy. Hence, both systems are equivalent with respect to rewrite behaviour, hence intensionally equivalent.

Proof An amazingly simple transformation which is clarified by the concept of interpolation in second-order intuitionistic propositional logic.[20] Extensions of system F correspond via the Curry-Howard-isomorphism [How80] to intuitionistic second-order propositional logics in natural deduction formulation: the types are seen as the formulas, and the typed terms are considered to be the formal proofs of their respective types/formulas. Fix α and ρ and let $m : \forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$. Relative to the free variables of m, ρ is provably monotone in α. Relative to the same free variables, $m\beta\alpha$ proves $(\beta \to \alpha) \to \rho[\alpha := \beta] \to \rho$. That is, ρ is proved from the two assumptions $\beta \to \alpha$ and $\rho[\alpha := \beta]$. Interpolation would mean that one finds an intermediate formula/type $\rho_\forall$ such that $\rho_\forall$ only has free type variables which are common to the assumptions *and* the conclusion,[21] and such that the assumptions imply $\rho_\forall$, and $\rho_\forall$ in turn implies the conclusion. In intuitionistic second-order propositional logic, this is easily achieved by setting

$$\rho_\forall := \forall\gamma.(\alpha \to \gamma) \to \rho[\alpha := \gamma]$$

(for a new type variable γ) provided that α occurs free in ρ. Otherwise, a trivial solution would be to take ρ as intermediate formula.

Nevertheless, we always take $\rho_\forall$ and check that there are terms of types $(\beta \to \alpha) \to \rho[\alpha := \beta] \to \rho_\forall$ and $\rho_\forall \to \rho$ with at most the free variables of m:

$$\lambda z^{\beta\to\alpha}\lambda x^{\rho[\alpha:=\beta]}\Lambda\gamma\lambda y^{\alpha\to\gamma}.m\beta\gamma(\lambda h^{\beta}.y(zh))x : (\beta \to \alpha) \to \rho[\alpha := \beta] \to \rho_\forall$$

$$\text{and} \quad q_\forall := \lambda x^{\rho_\forall}.x\alpha(\lambda y^{\alpha}.y) : \rho_\forall \to \rho.$$

[19] Some more sophistication even shows that we do not need iteration in the target system of our embedding [Mat98, 4.5.2 and 7]

[20] Originally, the embedding has been found by defining a “dual” of a system proposed by Mendler [Men87a] and a modified realizability interpretation inspired by [Ber93] of some direct proof of normalization by means of saturated sets à la Tait [Tai75]. In [Mat98] the intermediate system is motivated by help of an easy lattice-theoretic observation. It was my colleague Thorsten Altenkirch who suggested to recast the proof in terms of interpolation.

[21] In first-order logic, one instead requires that the function symbols of the interpolating formula occur in the assumptions as well as the conclusion.

We now substitute β by α in the first term, apply it to the identity on α and simplify in order to get the term

$$p_\forall := \lambda x^\rho \Lambda\gamma\lambda y^{\alpha\to\gamma}.m\alpha\gamma y x : \rho \to \rho_\forall.$$

Therefore, ρ and $\rho_\forall$ are even logically equivalent relative to the assumptions made in m. Note that $\rho_\forall$ has only one free occurrence of α, and the occurrence is non-strictly positive (two times to the left of a $\to$).

Finally, we are in the position to define the transformation of the types and the terms of the system of monotone inductive types in 6.1 into the system of non-interleaving positive inductive types. The only non-trivial case of the type transformation is $(\mu\alpha\rho)' := \mu\alpha.(\rho')_\forall$, i. e., if ρ has already been transformed into the non-interleaving positive inductive type ρ', we transform $\mu\alpha\rho$ into $\mu\alpha\forall\gamma.(\alpha\to\gamma)\to\rho'[\alpha:=\gamma]$. Since α does not occur free in $\rho'[\alpha:=\gamma]$, we rule out interleaving by this transformation. The terms are transformed trivially (i. e., homomorphically) except for the cases pertaining to μ:

- $(\mathrm{C}_{\mu\alpha\rho}mt)' := \mathrm{C}_{(\mu\alpha\rho)'}\hat{t}$ with $\hat{t}$ as follows: We assume that we already have translated t into t' of type $\rho'[\alpha:=(\mu\alpha\rho)']$ and m into m' and henceforth $p_\forall$ into $p'_\forall$ of type $\rho'\to\rho'_\forall$, hence we may define $\hat{t} := p'_\forall[\alpha:=(\mu\alpha\rho)']t'$ of type $\rho'_\forall[\alpha:=(\mu\alpha\rho)']$.
- $(r\mathrm{E}_\mu s)' := r'\mathrm{E}_\mu\hat{s}$ with r' the result of translating r and $\hat{s}$ as follows: We assume that we already have s' of type $\rho'[\alpha:=\sigma']\to\sigma'$ as the result of translating s and similarly $q'_\forall$ of type $\rho'_\forall\to\rho'$ and set $\hat{s} := \lambda z^{\rho'_\forall[\alpha:=\sigma']}.s'(q'_\forall[\alpha:=\sigma']z)$ of type $\rho'_\forall[\alpha:=\sigma']\to\sigma'$.
- $(r\mathrm{E}^+_\mu s)' := r'\mathrm{E}^+_\mu\Big(\lambda z^{\rho'_\forall[\alpha:=(\mu\alpha\rho)'\times\sigma']}.s'(q'_\forall[\alpha:=(\mu\alpha\rho)'\times\sigma']z)\Big)$, justified by a similar argument.

Notice that $\hat{t}$ needs m' (via $p'_\forall$) and that $\hat{s}$ does not need m' (because it only requires $q'_\forall$). It is thus essential to have the monotonicity witnesses attached to the introduction rule (μ-I).[22]

Is this translation of the type and term systems also an embedding of the **term rewrite systems**? It is indeed a simple consequence of the following

Lemma 3 *Let $h^{\alpha\to\beta}$ be a lambda abstraction and ρ be a positive inductive type. Then*

$$q_\forall[\alpha:=\beta](\mathrm{map}_{\alpha\mapsto\rho_\forall,\alpha,\beta}h(p_\forall x^\rho)) \to^* m\alpha\beta h x,$$

[22]However, a dual translation involving the existential quantifier allows to embed systems of monotone inductive types with monotonicity witnesses attached to the elimination rules into the system of non-interleaving positive inductive types [Mat98, 6.2.3].

where $\to^*$ means finitely many reduction steps of system F.

Proof Note that $\mathrm{map}_{\alpha\mapsto\rho_\forall,\alpha,\beta} = \lambda f^{\alpha\to\beta}\lambda x^{\rho_\forall}\Lambda\gamma\lambda y^{\beta\to\gamma}.x\gamma(\lambda z^\alpha.y(fz))$. The rest is nothing but beta reduction in system F.

The proof of the main theorem is finished by checking that

$$(\mathrm{C}_{\mu\alpha\rho}mt\mathrm{E}_\mu s)' \to^+ \Big(s\Big(m(\mu\alpha\rho)\sigma(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu s)t\Big)\Big)' \quad \text{and that}$$

$$(\mathrm{C}_{\mu\alpha\rho}mt\mathrm{E}^+_\mu s)' \to^+ \Big(s\Big(m(\mu\alpha\rho)(\mu\alpha\rho\times\sigma)\Big(\lambda x^{\mu\alpha\rho}.\langle x,(\lambda x^{\mu\alpha\rho}.x\mathrm{E}^+_\mu s)x\rangle\Big)t\Big)\Big)',$$

where $\to^+$ denotes at least one rewrite step of the system of positive inductive types. (In order to get a mathematically proper proof one has to state several lemmata on compatibility with substitution.)

To sum up: In the world of interpolation it seems to be trivial, but for inductive types we get the collapse of monotone inductive types with (full) primitive recursion to non-interleaving positive inductive types (in presence of system F's expressive power being crucial for the type transformation $\mu\alpha\rho \mapsto \mu\alpha\forall\gamma(\alpha\to\gamma)\to\rho[\alpha:=\gamma]$).

7 Related work and conclusions

It seems that Mendler first tried to give a definition of monotone inductive types in a system of dependent types [Men87b]. Unfortunately, the monotonicity proof has not been used in the definition of the additional rewrite rule for those types. One could ask how this system could be justified at all. There is an easy answer: The specific form of recursion used in that system is the same as that of the extension of system F also presented in [Men87a]. And Uustalu and Vene [UV97] discovered—perhaps by understanding what Leivant had in mind in [Lei90, p. 308]—that Mendler's (small) system (which has the superfluous requirement of positivity) embeds into non-interleaving positive inductive types. Thorsten Altenkirch proposed in 1993 to formulate the recursion rule by using an arbitrary monotonicity witness ("term of functorial strength") given beforehand. He also sketched a normalization proof by means of saturated sets.

The systems studied in this article correspond via the Curry-Howard-isomorphism to (second-order) propositional logics. Real applications demand predicate logics, the inductive types then become inductively defined predicates. Term rewriting becomes proof transformation. Uustalu studies a great variety of systems with inductively and coinductively defined predicates and gives embeddings between those systems [Uus98]. The inductive types could then serve as the term language for program extraction from proofs carried out in those systems.

On the side of systems with monotonicity witnesses there is also the notion of monotone fixed-point types [Mat99] (reducible to non-interleaving positive fixed-point types which have a very nice direct proof of strong normalization) and the reduction of monotone inductive and coinductive types to non-interleaving positive fixed-point types [Mat00].

A major open question concerns the reducibility of monotone inductive types to positive inductive types in the absence of universal types. A first result in this direction by Altenkirch [Alt99] shows that in parametric equality theory the witnesses for positive inductive types are necessarily those found in lemma 1 if they map the identity to the identity—something which the known witnesses for non-positive $\alpha \mapsto \rho$ fail to do.

Although there are still open questions I cannot imagine how to find a system expressing the results of Tarski's theorem more generally than the systems of monotone inductive types, unless one introduces dependent types which are indeed an enormous realm of further study.

Acknowledgements to Thorsten Altenkirch for many fruitful discussions and especially for his hint on using interpolation to explain the embedding. I am also thankful to the anonymous referee who provided so much useful criticism.

References

[Alt99] Thorsten Altenkirch. Logical relations and inductive/coinductive types. In Gottlob et al. [GGS99], pages 343–354.

[Ber93] Ulrich Berger. Program extraction from normalization proofs. In Marc Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 91–106. Springer Verlag, 1993.

[EL42] Samuel Eilenberg and Saunders Mac Lane. Natural isomorphisms in group theory. *Proc. Nat. Acad. Sci. U.S.A.*, 28:537–543, 1942.

[Geu92] Herman Geuvers. Inductive and coinductive types with iteration and recursion. In Bengt Nordström, Kent Petersson, and Gordon Plotkin, editors, *Proceedings of the 1992 Workshop on Types for Proofs and Programs, Båstad,* *Sweden, June 1992*, pages 193–217, 1992. Only published via `ftp://ftp.cs.chalmers.se/pub/cs-reports/baastad.92/proc.dvi.Z`.

[GGS99] Georg Gottlob, Etienne Grandjean, and Katrin Seyr, editors. *Computer Science Logic, 12th International Workshop, Brno, Czech Republic, August 24–28, 1998*, volume 1584 of *Lecture Notes in Computer Science*. Springer Verlag, 1999.

[Gir72] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. Thèse de Doctorat d'État, Université de Paris VII, 1972.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[HM99] Robert Harper and John C. Mitchell. Parametricity and variants of girard's j operator. *Information Processing Letters*, 70:1–5, 1999.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[How92] Brian Howard. *Fixed Points and Extensionality in Typed Functional Programming Languages*. PhD thesis, Stanford University, 1992.

[Lan98] Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer Verlag, second edition, 1998.

[Lei90] Daniel Leivant. Contracting proofs to programs. In Piergiorgio Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Studies in Data Processing*, pages 279–327. Academic Press, 1990.

[Mat98] Ralph Matthes. *Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types*. Doktorarbeit (PhD thesis), University of Munich, 1998. Available via the homepage `http://www.tcs.informatik.uni-muenchen.de/~matthes/`.

[Mat99] Ralph Matthes. Monotone fixed-point types and strong normalization. In Gottlob et al. [GGS99], pages 298–312.

[Mat00] Ralph Matthes. Monotone (co)inductive types and positive fixed-point types. Accepted for publication in the FICS '98 Proceedings appearing in Theoretical Informatics and Applications, 2000.

[Men87a] Nax P. Mendler. Recursive types and type constraints in second-order lambda calculus. In *Proceedings of the Second Annual IEEE Symposium on Logic in Computer Science, Ithaca, N.Y.*, pages 30–36. IEEE Computer Society Press, 1987. Forms a part of [Men87b].

[Men87b] Paul F. Mendler. Inductive definition in type theory. Technical Report 87-870, Cornell University, Ithaca, N.Y., September 1987. Ph.D. Thesis (Paul F. Mendler = Nax P. Mendler).

[MN98] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.

[Rey74] John C. Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Programming Symposium*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425, Berlin, 1974. Springer-Verlag.

[SU] Zdzisław Spławski and Paweł Urzyczyn. Type Fixpoints: Iteration vs. Recursion. In *Proceedings of the 1999 International Conference on Functional Programming (ICFP), Paris, France.* To appear.

[Tai67] William W. Tait. Intensional interpretations of functionals of finite type I. *The Journal of Symbolic Logic*, 32(2):198–212, 1967.

[Tai75] William W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Logic Colloquium Boston 1971/72*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer Verlag, 1975.

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Uus98] Tarmo Uustalu. *Natural Deduction for Intuitionistic Least and Greatest Fixedpoint Logics, with an Application to Program Construction.* PhD thesis, Royal Institute of Technology, Kista, Sweden, 1998.

[UV97] Tarmo Uustalu and Varmo Vene. A cube of proof systems for the intuitionistic predicate μ-, ν-logic. In M. Haveraaen and O. Owe, editors, *Selected Papers of the 8th Nordic Workshop on Programming Theory (NWPT '96), Oslo, Norway, December 1996*, volume 248 of *Research Reports, Department of Informatics, University of Oslo*, pages 237–246, May 1997.

[vdPS95] Jaco van de Pol and Helmut Schwichtenberg. Strict functionals for termination proofs. In Mariangola Dezani-Ciancaglini and Gordon Plotkin, editors, *Typed Lambda Calculi and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 350–364. Springer Verlag, 1995.