Tarski's fixed-point theorem and higher-order term rewrite systems

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

August 12, 1999

Abstract

Tarski proved that monotone operators on complete lattices have a least fixed-point which is simply given by the infimum of all pre-fixed-points. This theorem is the basis of a general understanding of induction. The computational counterpart of induction is iteration. Classically, one may derive (full) primitive recursion from iteration. However, this translation is computationally not acceptable. We introduce some polymorphic lambda calculi suitable for studying those phenomena. Led by the lattice-theoretic motivation we even arrive at a surprisingly simple reduction of arbitrary monotone inductive types (the lambda calculus representation of those least fixed-points) to syntactically monotone (i. e., positive) inductive types—reduction in the sense of higher-order term rewrite systems which is simulation of computation.

1 Introduction and Overview

The aim is to give a notion of inductive types which is as general as possible and draws its motivation from Tarski's theorem. The main result will be that even with respect to reduction we do not get too far: The system of monotone inductive types collapses to that of non-interleaving positive inductive types. But this reduction heavily relies on parametric polymorphism as expressed by system F.

*I am thankful for support by the Volkswagenstiftung.

We do not aim at proving results in full detail. This has all been done in my PhD thesis [Mat98]. The focus is on giving a self-contained introduction to the field of inductive types drawing its motivation from category theory (which is also not a prerequisite) which in turn is presented as a generalisation of lattice theory. And we even do not use lattice theory except for the definition of a complete lattice and Tarski's theorem. And also knowledge of lambda calculus is not taken for granted.[1]

Examples are given which show how rich even non-interleaving positive inductive types are and even more those with interleaving.

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 lambda calculus is developped to cover those notions syntactically. We leave the problem open how to model the morphism part of the functors. Examples, the definition of system F and a first answer form section 4. In section 5 primitive recursion as opposed to iteration is introduced. Section 6 proceeds with original material from [Mat98] spelt out in a (hopefully) more transparent manner. First the reference system is given, then (partly malicious) variants are dealt with, finally the collapse is explained. In section 7 an attempt is made to relate this all to other research.

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 preorder, and we even have that $\mu\Phi$ is a fixed point of Φ.

[1] In [Mat98] neither a categorial motivation nor an explanation of lambda calculus is given. Nor the example of the squaring function illustrating the use of "extended induction".

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

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

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 already know 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 Lambda Calculus

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 generalisation of the theory of partial orders 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 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).

In case there is at most one morphism from any object to any object, this gives precisely the partial orderings (= 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.

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

- The composition of morphisms is mapped to the composition of the mapped morphisms.

In case of partial orderings 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 partial ordering.

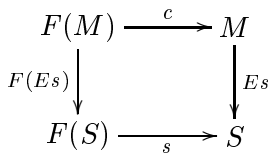

Figure 1: Initial F-Algebra

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 least pre-fixed-points 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 $\rightarrow$ allowing to form the type $\rho \rightarrow \sigma$ from types ρ and σ. The type $\rho \rightarrow \sigma$ 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 $\rho \rightarrow \sigma$ 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 which allows to **replace** $(\lambda x r)s$ 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 but may do the replacement to some subterm of the term under consideration. 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 normalisation of simply typed lambda calculus[3]. 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) normalisation implies that the least equivalence relation making a term and its replacement equivalent is decidable. It is called beta equality.

Let us model initial F-algebras syntactically [Geu92]. Fix a type ρ and some variable α which might occur in ρ. 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.

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).

[3]Without types even normalisation does not hold as may be seen from the generic example $(\lambda x.xx)(\lambda x.xx) \to (xx)[x := \lambda x.xx] \equiv (\lambda x.xx)(\lambda x.xx)$.

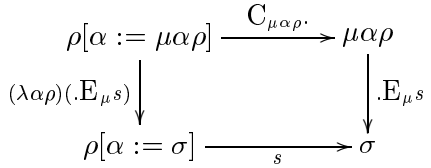

Figure 2: Inductive Type

Weak[4] 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.

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

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

($r \mapsto r'$ means that a subterm r may be replaced by r'.)

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 the term $(\lambda\alpha\rho)(.\mathrm{E}_\mu s)$ which has to have 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 $(\lambda\alpha\rho)(.\mathrm{E}_\mu s)$ 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.[5]

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:

[4] We do not require uniqueness.

[5] Clearly, 1 and sum types are the counterparts to the categorical notions of terminal object and coproduct, respectively [Geu92].

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

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

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 $0\mathrm{E}_\mu s$ reduces to s_0, and $(\mathrm{S}t)\mathrm{E}_\mu s$ and $s_{\mathrm{S}}(t\mathrm{E}_\mu s)$ reduce to the same term. An easy calculation shows that this is achieved by setting

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

This will be an instance of the general method indicated below. $\lambda\alpha.1+\alpha$ 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 an arrow) in $1+\alpha$.

Now consider the type $\mathrm{cont} := \mu\alpha.1+((\alpha \to \mathrm{nat}) \to \mathrm{nat})$ which may be seen as the type of continuations for programs whose results are natural

numbers. 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 $\mathrm{cont} \to \mathrm{nat}$ which completes in some sense the continuation it gets as an argument. It is specified as ($=$ shall denote the equivalence imposed by reduction):

- $e\mathrm{D} = 0$.
- $e(\mathrm{C}f) = fe$ 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 $\mathrm{cont} \to \mathrm{nat}$, 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 $(\lambda\alpha.1 + ((\alpha \to \mathrm{nat}) \to \mathrm{nat}))(.\mathrm{E}_\mu s)$ properly. Later we cite a result that this definition read as a term rewrite system (from left to right) is strongly normalising. 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 arrow type as the full set-theoretic function space.[6] Nevertheless, the monotonicity of $\lambda\alpha.1 + ((\alpha \to \mathrm{nat}) \to \mathrm{nat}))$ is seen quite easy: 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 arrows. Clearly, this type is not strictly positive.

A third example is given by the ρ-branching well-founded trees for ρ any type: $\mathrm{tree}(\rho) := \mu\beta.1 + (\rho \to \beta)$. ($\beta$ 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

[6] 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.

tree(α) 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 for α only positively occurring in ρ (and ρ obeys hereditarily to this restriction), then there are closed terms*

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

This is well-known [Lei90, How92]. For a careful definition yielding normal terms map see [Mat98, 5.1.1]. Now we may set for those types:

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

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

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).
- 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: $(\Lambda\alpha r)\sigma \mapsto r[\alpha := \sigma]$.

It was Girard who found a normalisation 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.[7]

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 reduction step in the source system gives rise to at least one reduction step between the translated terms in the target system.

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

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

The proof that our system of positive inductive types embeds into system F is as usual. (Note however, that most authors do not really care about the simulation of reduction 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. 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: For any terms $s_0 : \sigma$ and $s_S : \text{nat} \to \sigma \to \sigma$ there is a term $F : \text{nat} \to \sigma$ (with free variables among those of s_0 and s_S) such that $F0 = s_0$ and $F(St) = s_S t(Ft)$ (the equality relation induced by beta reduction). Of course, it is possible to represent functions defined by primitive recursion by functions defined by iteration only. However, the reduction behaviour of this encoding 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.[8]

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 reduction 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 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 morphisms π_A from $A \times B$ to A and π_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

[8] 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.

that $\pi_A \circ \langle l, r\rangle = l$ and $\pi_B \circ \langle l, r\rangle = r$. 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: $\langle r, s\rangle\mathrm{L} \mapsto r$ and $\langle r, s\rangle\mathrm{R} \mapsto s$.

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 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.

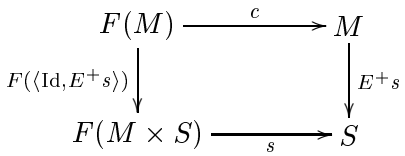

Figure 3: Initial Recursive F-Algebra

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.

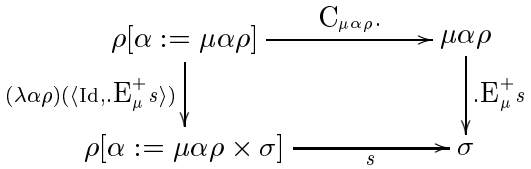

Figure 4: Primitive Recursion

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:

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

Theorem 3 *The extended system of positive inductive types with iteration*[9] *and primitive recursion is strongly normalising.*

Proof E. g. by an extension of an appropriate proof for system F.

It is preferable to include system F's universal type quantification 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 $\mathrm{map}_{\lambda\alpha\rho}$ of type $\forall\alpha\forall\beta.(\alpha\to\beta)\to\rho\to\rho[\alpha:=\beta]$. (Simply take $\Lambda\alpha\Lambda\beta.\mathrm{map}_{\lambda\alpha\rho,\alpha,\beta}$ and also consider the case of universal quantification in the proof of that lemma.) The term $\mathrm{map}_{\lambda\alpha\rho}$ obviously witnesses the monotonicity of $\lambda\alpha\rho$ as displayed by its type $\forall\alpha\forall\beta.(\alpha\to\beta)\to\rho\to\rho[\alpha:=\beta]$. And this monotonicity is derived from the syntactic requirement of positivity in the formation of $\mu\alpha\rho$. This directly leads us to the generalisation studied in the next section.

6 Monotone Inductive Types

The observation central to this article:[10]

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

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 restriction.

[9] Iteration appears in the definition of the terms map.

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

- 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 $\mathrm{map}_{\lambda\alpha\rho,\mu\alpha\rho,\tau}$ by $m(\mu\alpha\rho)\tau$ (for the types $\tau = \sigma$ and $\tau = \mu\alpha\rho \times \sigma$, respectively).

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

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 normalisation proof for system F [Mat98, 9.5.1].

6.2 Other Systems of Monotone Inductive Types

Normalisation of the previous system does not carry over to arbitrary other reasonable definition 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 both cases we get as term formation rule: If t is a term of type $\mu\alpha\rho$, m is as given 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 $\mathrm{map}_{\lambda\alpha\rho}$ with first argument $\mu\alpha\rho$ and second argument σ):

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

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

Proof Again by an intricate extension of the normalisation proof for system F in the positive case [Mat98, 9.5.4][11], and by a counterexample in the negative case.

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

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

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.

$(\mathrm{C}_{\mu\alpha\alpha}t)\mathrm{E}_{\mu}ms$ reduces 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 reduction strategy.

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.*

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

Proof An amazingly simple transformation which is clarified by some intermediate system motivated from an easy lattice-theoretic observation. It has been found by defining a "dualisation" of a system proposed by Mendler [Men87a] and an analysis of some direct proof of normalisation by means of saturated sets à la Tait [Tai75].

First the lattice-theoretic motivation: Let again $(U, \subseteq, \wedge, \vee)$ be a complete lattice of sets with set inclusion $\subseteq$. Consider an arbitrary $\Phi : U \to U$. Define the lower monotonisation $\Phi^{\subseteq}$ of Φ by

$$\Phi^{\subseteq}(S) := \bigwedge\{\Phi(S')|S \subseteq S'\}.$$

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

Obviously, $\Phi^{\subseteq}(S) \subseteq \Phi(S)$ and $\Phi^{\subseteq}$ is monotone—simply because of syntactic reasons (the argument occurs at a non-strict positive position).

Compare this with the definition of the limit inferior in calculus:

$$\liminf_{k\to\infty} a_k := \lim_{n\to\infty} \inf_{k\geq n} a_k.$$

The limit to the right always exists (it may be infinite) because the sequence $(\inf\{a_k | k \geq n\})_{n\in\mathbb{N}}$ is monotonically increasing. And this may again be read off syntactically: The argument n occurs positively in the expression $\inf\{a_k | k \geq n\}$.

Obviously, $\Phi^{\subseteq} \supseteq \Phi \Leftrightarrow \Phi$ monotone.

In order to see the connection to the lambda calculus, we exploit our assumption that the lattice consists of sets and get

$$\Phi^{\subseteq}(S) = \{x | \forall S'.S \subseteq S' \Rightarrow x \in \Phi(S')\}.$$

The function mapping Φ to $\Phi^{\subseteq}$ now corresponds to a transformation from $\lambda\alpha\rho$ to $\lambda\alpha\forall\beta.(\alpha \to \beta) \to \rho[\alpha := \beta]$ where we use an informal modified realisability interpretation (due to Kreisel, for a readable introduction see [Ber93]). In essence, the translation works by ignoring first-order quantifiers, replacing sets by types and implication and subset inclusion[12] by the arrow type.

Let us investigate, what our rules (μ-I), (μ-E) and (μ-E$^+$) applied to $\mu\Phi^{\subseteq}$ instead of $\mu\Phi$ say in terms of Φ (i. e., we unfold the definition of $\Phi^{\subseteq}$). In every even line the type read off via the informal realisability interpretation is shown.

(μ-I) Rule $(\forall S'.\mu\Phi^{\subseteq} \subseteq S' \Rightarrow x \in \Phi(S')) \Rightarrow x \in \mu\Phi^{\subseteq}$.
Type $(\forall\alpha.(\mu\alpha\rho \to \alpha) \to \rho) \to \mu\alpha\rho$.

(μ-E) Rule $\Big(\forall x\Big(\forall S'.S \subseteq S' \Rightarrow x \in \Phi(S')\Big) \Rightarrow x \in S\Big) \Rightarrow \mu\Phi^{\subseteq} \subseteq S$.
Type $\Big(\Big(\forall\alpha.(\sigma \to \alpha) \to \rho\Big) \to \sigma\Big) \to \mu\alpha\rho \to \sigma$.

(μ-E$^+$) Rule $\Big(\forall x\Big(\forall S'.(\mu\Phi^{\subseteq} \wedge S) \subseteq S' \Rightarrow x \in \Phi(S')\Big) \Rightarrow x \in S\Big) \Rightarrow \mu\Phi^{\subseteq} \subseteq S$.
Type $\Big(\Big(\forall\alpha.((\mu\alpha\rho \times \sigma) \to \alpha) \to \rho\Big) \to \sigma\Big) \to \mu\alpha\rho \to \sigma$.

We now form a system of terms in which these types are inhabited by construction for arbitrary $\mu\alpha\rho$, i. e., by modelling the term formation rules such that they reflect the above rules where we take $\mu\Phi^{\subseteq}$ as primitive and do no longer consider the concrete definition of $\Phi^{\subseteq}$:

[12] This is nothing but implication together with a first-order quantifier.

- If $t : \forall\alpha.(\mu\alpha\rho \to \alpha) \to \rho$, then $\mathrm{C}_{\mu\alpha\rho}t : \mu\alpha\rho$.
- If $r : \mu\alpha\rho$ and $s : \Big(\forall\alpha.(\sigma \to \alpha) \to \rho\Big) \to \sigma$, then $r\mathrm{E}_\mu s : \sigma$.
- If $r : \mu\alpha\rho$ and $s : \Big(\forall\alpha.((\mu\alpha\rho \times \sigma) \to \alpha) \to \rho\Big) \to \sigma$, then $r\mathrm{E}^+_\mu s : \sigma$.

How about reduction? $(\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu s$ shall have as reduct the reduct of the translation corresponding to the mapping $\mu\alpha\rho \mapsto \mu\alpha\forall\beta(\alpha \to \beta) \to \rho[\alpha := \beta]$ of the types. This yields non-interleaved positive inductive types for which we have reducts via lemma 1. We do the same for primitive recursion and arrive at

- $(C_{\mu\alpha\rho}t)\mathrm{E}_\mu s \mapsto s\Big(\Lambda\alpha\lambda z^{\sigma\to\alpha}.t\alpha(\lambda x^{\mu\alpha\rho}.z(x\mathrm{E}_\mu s))\Big)$.
- $(C_{\mu\alpha\rho}t)\mathrm{E}^+_\mu s \mapsto s\Big(\Lambda\alpha\lambda z^{\mu\alpha\rho\times\sigma\to\alpha}.t\alpha\Big(\lambda x^{\mu\alpha\rho}.z\langle x, (\lambda x^{\mu\alpha\rho}.x\mathrm{E}^+_\mu s)x\rangle\Big)\Big)$.

Now we get to the heart of the proof: If Φ is monotone, then $\Phi^\subseteq = \Phi$, hence the original rules are derivable from our new rules (trivially, because we only unfolded the definition of $\Phi^\subseteq$ when deriving the latter rules). But we get much more: We can even embed the **term rewrite system** of monotone inductive types into the new rewrite system! The reason: The derivation of the original rules from the new ones only uses Φ's monotonicity at places where we have the monotonicity witness at hand on the term side. It should not be by chance that even reduction is preserved (which may be easily checked [Mat98, 6.3.2]). The underlying principles clearly deserve further study. To sum up: In the lattice world 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 $\mu\alpha\rho \mapsto \mu\alpha\forall\beta(\alpha \to \beta) \to \rho[\alpha := \beta]$).

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 reduction. 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 normalisation proof by means of saturated sets.

Extensions of system F correspond via the Curry-Howard-isomorphism to intuitionistic second-order propositional logics in natural deduction formulation. 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 [Mat99b] (reducible to non-interleaving positive fixed-point types which have a very nice direct proof of strong normalisation) and the reduction of monotone inductive and coinductive types to non-interleaving positive fixed-point types [Mat99a].

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 $\lambda\alpha\rho$ 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.

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 Pettersson, 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.

[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/`.

[Mat99a] 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, 1999.

[Mat99b] Ralph Matthes. Monotone fixed-point types and strong normalization. In Gottlob et al. [GGS99], pages 298–312.

[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).

[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 Con-*

struction. 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.