Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types

Dissertation
an der Fakultät für Mathematik und Informatik
der Ludwig-Maximilians-Universität München

vorgelegt von
Ralph Matthes

eingereicht am 20. Mai 1998

1. Berichterstatter: Prof. Dr. H. Schwichtenberg

2. Berichterstatter: Prof. Dr. W. Buchholz

Tag der mündlichen Prüfung: 2. Februar 1999

Chapter 1

Introduction

We study systems of typed terms extending system F [Gir72, Rey74] with notions of β-reduction for every type construction. We focus on types of the form $\mu\alpha\rho$ whose intended interpretation is least pre-fixed-points of monotone operators. There are always two associated β-rules: The rule modelling iteration and the rule which models primitive recursion on the data structure. It is well known that data types such as the natural numbers may be represented in system F. This works very well for the definition of functions by iteration. But even the defined predecessor on natural numbers inside system F does not behave very well: The predecessor of the successor of n can be reduced to n only for numerals n and this process needs $O(n)$ steps. I do not know whether there is a proof that no reduction preserving embedding of primitive recursion on positive inductive types into system F is possible. But nobody seems to know such an embedding. Therefore, I decided to compare extensions of F allowing for primitive recursion on inductive types.

1.1 Motivation

This work is in some sense a chapter of the theory of complete lattices. But without the magnifying glasses of proof theory it would be quite short a chapter and moreover it would not contain any new result—new relative to Tarski's fixed-point theorem [Tar55][1].
First we use the magnifier of intuitionism to explore whether the classical results may also be proved constructively or whether they may be turned into statements ("constructivizations") permitting a constructive proof. Along this line we may define the term systems which are typed lambda calculi and also the encodings between them.
But this thesis is dominated by the question of the behaviour of term rewrite rules[2] of the systems and of their relation between systems. And this question may be turned into a problem of proof theory by using the Curry-Howard isomorphism [How80] in its most trivial reading[3]: The types of the extensions of system F are the formulas of an intuitionistic second-order propositional logic and the typed terms are simply proof terms in a natural deduction formulation of this intuitionistic logic. A term proves its type and its free typed term variables are the assumptions used in the proof. The term rewrite rules become proof transformations and they are justified if they do not change the type (formula) of the term (proof) and have as free term variables (assumptions) at most those which were present in the original term (proof).
We continue by adopting the proof-theoretic language. The analysis of proofs is greatly simplified

[1] We read: "Most of the results contained in this paper were obtained in 1939."
[2] We only consider β-rules for all the type constructs.
[3] Many subtleties concerning Curry-Howard isomorphisms may be found in [Geu93].

if the proof transformation system is normalizing and confluent because then one may only consider proofs in normal form. The comparison of the head forms of proofs (e. g. Lemma 2.24) and the set of normal forms (e. g. the inductively defined set NF on p. 18) shows what is gained by normalization. Consistency is a typical problem which is solved by studying the normal forms.

Since [Tai75] it has been common to establish strong normalization if possible. Although already normalization furnishes consistency and hence needs means of proof going beyond the logical system (which is an extension of pure second-order propositional logic) it is a challenge also to prove strong normalization. There are numerous published variations on that proof. Nevertheless I present another proof of strong normalization for system F which in my view separates the difficulties even more—most prominently the absence of reduct analysis in any reasoning with saturated sets which are a variant of the reducibility candidates (see e. g. [GLT89]).

Up to now I only spoke about system F. Although I present it in great detail it is only the basis on which to build the extensions by inductive types. Proof theorists might lose their interest in this work because system F is already fully impredicative and an extension by inductive types seems to give only "sugar" to the system. This would in fact be true if we only studied iteration on these inductive types. But we also study (full) primitive recursion. This time it is an algorithmic motivation: Functions defined by primitive recursion may be more efficient than functions defined by iteration. I alluded to it at the very beginning of this chapter by giving the example of the predecessor on the naturals (which in that encoding works iteratively).

This algorithmic motivation is intimately connected to a proof-theoretic question: What is the appropriate logical system allowing to extract efficient programs out of existence proofs[4]? Extraction of typed lambda terms[5] (viewed as programs) from intuitionistic proofs may be done via the modified realizability interpretation (due to G. Kreisel; see the citations in [Ber95]). I mention two examples where modified realizability is used for the extraction of programs out of normalization proofs: In [Ber93] a normalization algorithm is extracted from a proof of strong normalization of simply typed lambda calculus following Tait's computability predicate method [Tai67]. [vdP96b] builds on [Ber93] and extracts bounds on reduction lengths for Gödel's T from a variant of the normalization proof by the Tait method.

In [Ber95] a modified realizability interpretation is given for a system of interleaving positive inductive types with iteration only. The main insight I got from it was the motivation of the reduction rules for positive inductive types from it: The soundness of modified realizability only holds up to some equational theory on the terms and the attempt to prove soundness shows which equational axioms have to be postulated. All these efforts are in vain if the resulting equality theory is not effective but fortunately the axioms may be turned into term rewrite rules which form a strongly normalizing and confluent term rewrite system having hence decidable intensional equality. Hence, we may indeed interpret the extracted terms as programs.

Although I believe that a modified realizability interpretation could be given for the systems presented in this thesis I decided to be very informal about program extraction. But one should see the study of the term systems as the first part of a project aiming at the constructive interpretation of monotone inductive definitions. And it is a big part because the proof of soundness of the realizability interpretation is not so much different from the proof of strong normalization. Most of the difference is the extension from propositional logic (the term system) to full predicate logic. And this extension is quite easy: As long as there are no dependent types in the framework one can only use the objects in a uniform way in the proofs.

[4] More precisely: Proofs of Π^0_2-formulas.

[5] In [Par92] (system TTR) untyped lambda terms are extracted and only the reduction of untyped lambda calculus exhibits the algorithmic content. Correctness is expressed by typing rules which also include positive recursive types. In contrast to that I add reduction rules for every new type construct.

Perhaps the second part would become big if not only soundness (which guarantees that the extracted program meets the specification expressed in the existential statement) were the goal. A project going beyond the soundness theorem for modified realizability would be to formalize the normalization proofs and not only extract embeddings between systems out of them but also have a meta-theorem assuring that these encodings are always respecting β-reduction, hence that the extracted encodings are indeed embeddings (cf. section 9.3).
What are monotone inductive types? They are motivated by Tarski's theorem from which I only use the following: Let $(U, \leq)$ be a complete lattice and $\Phi : U \to U$ be monotone. Then Φ has a least fixed point[6] $\mathsf{lfp}(\Phi)$ which is given by

$$\mathsf{lfp}(\Phi) = \bigwedge\{M \in U | \Phi(M) \leq M\}.$$

The minimality of $\mathsf{lfp}(\Phi)$ gives a principle of induction:

$$(\mathsf{lfp}\text{-E}) \quad \Phi(M) \leq M \Rightarrow \mathsf{lfp}(\Phi) \leq M.$$

This will motivate iteration. As a derived rule we get

$$(\mathsf{lfp}\text{-E}^+) \quad \Phi(\mathsf{lfp}(\Phi) \wedge M) \leq M \Rightarrow \mathsf{lfp}(\Phi) \leq M.$$

I call this rule the principle of extended induction[7]. It motivates primitive recursion.
The idea is as follows: We depart from the explicit representation of $\mathsf{lfp}(\Phi)$ as an infimum (which for pedagogic reasons is studied in a system of infimum types) because its reduction behaviour is not satisfactory. Instead we directly represent $\mathsf{lfp}(\Phi)$ by a type $\mu\alpha\rho$ in a system with iteration and primitive recursion.
We may now give an answer to the question on an appropriate logical system allowing to extract "efficient" programs out of existence proofs: It has extended induction as a primitive notion.
Monotone inductive definitions also draw attention in classical proof theory. The content of [Fef82, Rat96, GRS97, Rat98, Rat99] is orthogonal to the topics of this thesis, though. Their results are mainly based on classical logic, and they are very much concerned about measuring the strength of the theories at hand which clearly forbids the use of full impredicativity as expressed by system F. In this thesis, we exclusively study intuitionistic systems extending system F. The problems solved all have to do with reduction behaviour which does not play a role in the cited papers. Due to the different positions concerning impredicativity they focus on the possibility of getting least fixed points via transfinite iteration of a monotone operator starting with the empty set (hence being an approach from below) whereas we build on Tarski's theorem guaranteeing that the impredicatively justified infimum of the pre-fixed-points (hence being an approach from above) is also a pre-fixed-point and hence that a least pre-fixed-point exists.
Even the focus on primitive recursion instead of iteration (especially in the case of non-strictly positive inductive types) itself seems to be a quest for more impredicativity because the step term in the recursive definition may refer to the inductively defined set (modelled by the inductive type) in its entirety.
How do we model monotone inductive types? We allow $\mu\alpha\rho$ to be a type for any type ρ and any type variable α but we only may form terms of this type if we know monotonicity. How do we know that $\lambda\alpha\rho$ is monotone? By having already constructed a term of type

[6] In fact we only use that we have a least pre-fixed-point. The fact that this least pre-fixed-point is also a fixed point is never used in the systems of this thesis because it does not fit with β-reduction.

[7] In mathematical practice it is quite common to make unconcious use of "extended induction" instead of the original induction principle. Take e. g. the squaring function on the reals and try to prove that it has only natural numbers as values for natural number arguments. (One may already assume that addition does not leave $\mathbb{N}$.)

$\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$. This term is called monotonicity witness and may arbitrarily use term formation rules for inductive types, hence permitting any interleaving of inductive types.
Interleaving inductive types are those where a free parameter of a subexpression of the type of the form $\mu\alpha\rho$ is bound by μ. Nesting would only mean that a type of the form $\mu\alpha\rho$ may be a subexpression of a type $\mu\alpha'\rho'$ with α' not free in $\mu\alpha\rho$. One could also call the interleaving inductive types inductive types with essential use of parameters.
Most presentations (for me important are [Lei90, Geu92]) restrict to positive inductive types, hence to a restriction already enforcing monotonicity on the level of types[8]. Mendler's type-theoretic system (with dependent types and subset types) in [Men87b] has the notion of monotone inductive types, i. e., μ-types are introduced under the assumption that the monotonicity statement (expressed via subset types) has already been derived. Surprisingly, the monotonicity proof is not used in the reduction rule. But this becomes clear when recognizing that the elimination rule does not express definition by primitive recursion. It corresponds to a definition by primitive recursion on a set inductively defined via a "positivization" of the monotone operator expressed by the monotone inductive type. And this positivization even works for non-monotone operators explaining why the monotonicity proof could be left out.
Why monotone inductive types?

- We try to prove as general results as possible.
- Monotone inductive types have a meta-theory where proofs may be done by structural recursion. In the case of positive inductive types we have the phenomenon that syntactic material pertaining to the canonical monotonicity witnesses necessary for formulating the rules of iteration and primitive recursion "pops up" during reduction. Therefore e. g. in the final soundness theorem in the proof of strong normalization one has to use quite complicated measures of induction. Compare [JO97] in the case of "abstract data type systems" where on p. 380 we find a tremendously complicated induction measure which is needed because the syntactic material is not carried around[9].
- It is nice to have an apparatus with expressive means which allow to formulate theorems which are only true after imposing some restrictions. In section 4.7.1 I present a slight variation of a very reasonable system for which normalization fails.
- And there is a nice inductive type which is monotone but not positive (due to U. Berger).

However, a large part of this thesis is concerned with showing how to embed systems of monotone inductive types into systems of non-interleaving positive inductive types while preserving reduction.[10] The main idea is to define "positivizations" of an arbitrary operator. Let us give an example: In calculus the limit inferior $\liminf$ of a sequence $(a_k)_{k\in\mathbb{N}}$ of real numbers is formed by setting

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

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 for this work it is most important that the monotonicity may be read off syntactically: The argument n occurs positively in the expression $\inf\{a_k|k \geq n\}$.

[8] Those systems nicely embed into the proposed systems of monotone inductive types.
[9] No doubt, this is more user-friendly but the user interface may be built up after having shown normalization and confluence.
[10] Let henceforth an embedding be an encoding which preserves reduction, i. e., for any reduction step in the source system one can do at least one reduction step in the target system to get from the encoding of a term to the encoding of its reduct.

In a similar fashion we get the lower and the upper monotonization of any operator. The upper monotonization gives rise to an embedding of monotone inductive types into non-interleaving strictly positive inductive types including the existential quantifier. The lower monotonization motivates embeddings into non-interleaving positive inductive types without the existential quantifier.
The embeddings work because the monotonization of a monotone operator is the operator itself, but the reflection of this fact joins the monotonicity via a monotonicity witness to the monotonicity via positivity.
Two other notions of monotonicity enter the proofs of strong normalization. They are the semantic requirement that a monotonicity witness is element of the saturated set

$$\forall\mathcal{P}\forall\mathcal{Q}.(\mathcal{P}\to\mathcal{Q})\to\Phi(\mathcal{P})\to\Phi(\mathcal{Q})$$

for Φ a family of operators on saturated sets which is a strong monotonicity requirement for Φ and finally the standard notion of monotonicity for such a family of operators. Unfortunately, I have no intuition how to reason by mere monotonicity of such operators in order to establish strong normalization. The good news is: I never had to reason in this way because always monotonizations could be found which were syntactically monotone (i. e., positive) and had properties good enough to let the proofs of strong normalization go through.

1.2 Outline of results

Most of the work is done in order to establish the following

Theorem Define four classes of systems as follows.

I Systems without primitive recursion.

- F: the basic impredicative system with $\to$ and $\forall$.
- eF: F extended by 0, 1, $\times$ and $+$.
- IT: eF extended by infimum types $i\alpha\rho$.
- eF+ex: eF extended by $\exists$.
- varEMIT–rec: the variant of the system of elimination-based (elimination rules without monotonicity witness) monotone inductive types with no primitive recursion but iteration.
- IMIT–rec: the system of introduction-based (introduction rule without monotonicity witness) monotone inductive types with no primitive recursion but iteration.
- MeIT–rec: the system in the spirit of Mendler with no primitive recursion but iteration.
- coMeIT–rec: the dual to MeIT–rec.
- Any other system of inductive types with iteration but no primitive recursion.

II The systems in IIa and in IIb to be defined next.

IIa Strictly positive systems without $\exists$.

- SPIT: the system of (interleaving) strictly positive inductive types (with iteration and primitive recursion and map terms built without primitive recursion but iteration).

- NISPIT: the system of non-interleaving strictly positive inductive types (with iteration and primitive recursion and map terms built with neither iteration nor recursion).

IIb Classes of systems with selected monotone inductive types.

- ESMIT: systems of elimination-based (unrestricted use of $\mu^{(+)}$-elimination in map terms) selected monotone inductive types.
- ESMITit: those of ESMIT without primitive recursion in the definitions of map terms.
- ESMITrec: those of ESMIT without iteration in the definitions of map terms.
- ESMIT+ex: systems like ESMIT but including $\exists$.
- ISMIT: systems of introduction-based (unrestricted use of μ-introduction in map terms) selected monotone inductive types.
- ISMIT+ex: systems like ISMIT but including $\exists$.

III 20 other systems of inductive types.

- PIT=PITit: the system of positive inductive types with iteration and primitive recursion where interleaving and non-strict positivity are allowed and the map terms do not use primitive recursion but only iteration.
- PITrec: like PITit but the map terms use primitive recursion and no iteration.
- PIT+ex: PIT extended by $\exists$.
- SPIT+ex: SPIT extended by $\exists$.
- NIPIT: the system of non-interleaving (but including non-strictly) positive inductive types with iteration and primitive recursion and map terms built with neither iteration nor recursion.
- NISPIT+ex: NISPIT extended by $\exists$.
- EMIT: the system of elimination-based (elimination rules without monotonicity witness) monotone inductive types with primitive recursion and iteration.
- EMIT–it: EMIT without iteration.
- varEMIT: a variant of EMIT with a slightly weaker requirement of monotonicity for the monotonicity witnesses.
- IMIT: the system of introduction-based (introduction rule without monotonicity witness) monotone inductive types with primitive recursion and iteration.
- IMIT–it: IMIT without iteration.
- varIMIT: a variant of IMIT with a slightly weaker requirement of monotonicity for the monotonicity witnesses.
- IMIT+ex: IMIT extended by $\exists$.
- MeIT: the system in the spirit of Mendler ("Mendler's Inductive Types") with iteration and primitive recursion.
- MeIT–it: MeIT without iteration (nearer to Mendler's [Men87a] system).
- coMeIT: the dual (the explanation via "monotonizations" of operators is dual) to MeIT.
- coMeIT–it: coMeIT without iteration.

- UVIT: "Uustalu's and Vene's Inductive Types", a slight generalization of MeIT (which serves for clarifying the embedding of MeIT into NISPIT+ex) also with iteration and primitive recursion.
- MePIT: the subsystem of NISPIT+ex which is actually needed for embedding MeIT into NISPIT+ex. It also has iteration and primitive recursion.
- coMePIT: the subsystem of NIPIT which is actually needed for embedding coMeIT into NIPIT; with iteration and primitive recursion.

Let an embedding be an encoding which preserves reduction, i. e., for any reduction step in the source system one can do at least one reduction step in the target system to get from the encoding of a term to the encoding of its reduct.
Then the following holds: There are embeddings of each system in I into any other. There are embeddings of each system in II into any of III and each system of III embeds into any other system of III. Hence, the systems in I are embedding equivalent to each other, and the same holds for the systems in III.
Every system in I, II and III is strongly normalizing and confluent.

All of the above systems are motivated and defined carefully. Examples are given and a lot of additional information is taken out of the proofs, among them naïve reduction-free normalization, consistency, the impossibility of proving the Peirce formula, an insight how much easier weak normalization is than strong normalization and even embeddings into systems of fixed-point types.

In chapter 2 the systems without inductive types are formulated and analyzed very carefully. Starting from system F with arrow types and universal types only, we subsequently add 0, 1, product types, sum types, existential types and infimum types.
Chapter 3 shows the different notions of inductive types (including strictly positive and positive and non-interleaving positive inductive types) and measures for them. Examples are considered.
Term systems for monotone inductive types are studied in chapter 4. The two possibilities of fixing a monotonicity witness to the term rules are presented. There is a non-trivial example of monotonicity which does not come from positivity. Also the systems with selected monotone inductive types having a specified monotonicity witness for every inductive type are defined and embedded into the systems with monotone inductive types by using a notion of stratified term. An elaborate discussion on the relation of iteration and recursion is included. The iterative fragment of varEMIT is embedded into the system of infimum types by simply reflecting the proof of Tarski's theorem.
In chapter 5 systems with positive inductive types are defined which means to define the canonical monotonicity witnesses whose stratification and uniformity alone guarantee strong normalization. It is presented how much easier the definitions are for subsystems like that of strictly positive inductive types or of non-interleaving positive inductive types. Very special instances are MePIT and coMePIT which may be seen as examples for the calculation of canonical monotonicity witnesses.
Chapter 6 is the heart of this thesis: The junction between monotone inductive types and non-interleaving positive inductive is introduced in form of MeIT and coMeIT the first of which is a variant of Mendler's [Men87a] system and the second a dual to it. The introductory sections to that chapter are essential for understanding why the embeddings work.
In chapter 7 we collect the embeddings established so far and prove the statements on embeddings in the theorem above.
Confluence is proved by a variant to Takahashi's [Tak95] method in chapter 8.

Chapter 9 and appendix A contain the most technical part: The proofs of strong normalization for several systems. Again much effort has been invested to make the proofs as clear and as modular as possible. The variations on the proof for EMIT are put in the appendix although they show a wealth of subtleties. A final discussion explains how the embeddings of monotone inductive types into MePIT and coMePIT could have been found.
Appendix B discusses possible extensions which came to my mind during the writing of this thesis mostly concerning fixed-point types, η-reduction, functoriality of the monotonicity witnesses and permutative conversions for inductive types. A list of open problems is given.

As central points of this thesis I consider:

- Monotone inductive types formulated[11] and proved to be strongly normalizing and confluent.
- Selected monotone inductive types formulated (as abstraction from the canonical monotonicity witnesses for positive inductive types) and embedded into monotone inductive types via an inductively defined set of stratified terms.
- Mendler's system proved to be strongly normalizing without any restriction to forming inductive types.
- Formulation of a dual (coMeIT) to Mendler's system.
- Embedding of monotone inductive types into non-interleaving positive inductive types. The main step: varEMIT embeds into coMeIT and varIMIT embeds into MeIT. Explanation why such easy embeddings exist.
- Reduction of iteration to primitive recursion even for introduction-based monotone inductive types.
- A measure of height (h) for abstracted types which allows to define the canonical monotonicity witnesses for interleaving positive inductive types.
- A technical device: The vector notation for multiple eliminations.
- The separation of reduct analysis from any reasoning on saturated sets.
- The definition of infimum types to clarify the encoding of iteration on elimination-based monotone inductive types inside system F.
- The organization of the proof of strong normalization such that always an introduction-based and an elimination-based definition of computability predicates is possible and that embeddings may be read off the proof.
- The equivalence with respect to embeddings of the 20 systems of class III in the above theorem.

[11] The very idea of considering primitive recursion in the presence of some arbitrary term of "functorial strength" already occurs in [Alt93c] together with a sketch of an elimination-based normalization proof.