A General Formulation of Simultaneous Inductive-Recursive Definitions in Type Theory

Peter Dybjer
Department of Computing Science
Chalmers University of Technology
S-412 96 Göteborg, Sweden
peterd@cs.chalmers.se

May 12, 1998

Abstract

The first example of a simultaneous inductive-recursive definition in intuitionistic type theory is Martin-Löf's universe à la Tarski. A set U_0 of codes for small sets is generated inductively at the same time as a function T_0, which maps a code to the corresponding small set, is defined by recursion on the way the elements of U_0 are generated.

In this paper we argue that there is an underlying *general* notion of simultaneous inductive-recursive definition which is implicit in Martin-Löf's intuitionistic type theory. We extend previously given schematic formulations of inductive definitions in type theory to encompass a general notion of simultaneous induction-recursion. This enables us to give a unified treatment of several interesting constructions including various universe constructions by Palmgren, Griffor, Rathjen, and Setzer and a constructive version of Aczel's Frege structures. Consistency of a restricted version of the extension is shown by constructing a realisability model in the style of Allen.

1 Introduction

Martin-Löf type theory is a foundational framework in which induction is the principal notion. It is, to quote Martin-Löf [27, p73], "intended to be a full scale system for formalising intuitionistic mathematics as developed, for example, in the book by Bishop [10]".

It is also a typed functional programming language not unlike ML [33] or Miranda [9]. A "set" in the theory is defined inductively by listing its constructors with their types in much the same way as one defines a recursive datatype in ML or Miranda. But whereas ML and Miranda are based on the simply typed λ-calculus, Martin-Löf type theory also has dependent types. The other key difference is that only well-founded elements of datatypes (sets) and terminating programs (total functions) may be constructed. To ensure well-foundedness datatype definitions have to satisfy a kind of "strict positivity" criterion. Moreover, to ensure termination recursive function definitions are restricted to "structural" recursion, that is, recursion on the way the elements of the domain of definition are inductively generated.

For simple types we can use the following notion of strict positivity. Let

$$\mathit{intro} : \alpha_1 \to \cdots \to \alpha_n \to P$$

($n \geq 0$) be a constructor for the datatype P. Then α_i either does not contain any occurrences of P or has the form $\xi_1 \to \cdots \to \xi_m \to P$ ($m \geq 0$), where ξ_j does not have any occurrences of P. It is also clear what the appropriate notion of structural recursion is for such a recursive datatype. Note that we allow *generalised inductive definitions* since a constructor can have functional arguments ($m > 0$). Therefore, the informal semantic notions of well-founded element and terminating function depend on each other.

The introduction of dependent types dramatically increases the expressiveness of the language. In particular, we can interpret intuitionistic predicate logic by following Curry, Howard, and de Bruijn and identify propositions and sets. In addition to the ordinary non-dependent set formers $0, 1, +, \times$, and $\to$, which can be used for interpreting the logical connectives $\bot$, $\top$, $\vee$, $\wedge$, and $\supset$, we now also have Σ and Π, the disjoint union and Cartesian product of a family of sets, which can be used for interpreting the quantifiers $\exists$ and $\forall$.

However, the appropriate notion corresponding to "strict positivity" becomes more complex in the context of dependent types. Instead of formulating such a general condition for inductive definitions of sets Martin-Löf [31, 27, 28, 29] gave rules for a collection of specific set formers. However, this collection may be extended when there is a need for it provided the informal semantic principles of the theory are respected.

The possibility of formulating a general schema was however mentioned in Martin-Löf 1972 [31]:

> The type N is just the prime example of a type introduced by an *ordinary inductive definition*. However, it seems preferable to treat this special case rather than to give a necessarily much more complicated general formulation which would include $(\Sigma \in A)B(x)$, $A + B$, N_n and N as special cases. See Martin-Löf 1971 [26] for a general formulation of inductive definitions in the language of ordinary first order predicate logic.

The first such general schema was formulated by Backhouse [7] and covered the case of inductively defined sets (possibly depending on parameters). This schema was generalised to the case of inductively defined families of sets by Dybjer [19, 20]. Inductively defined families subsume inductively defined predicates, and this schema can be viewed as the type-theoretic generalisation of the natural deduction schema for inductively defined predicates in predicate logic given by Martin-Löf [26].

In this paper we introduce a further generalization of the schema in Dybjer [20]. It covers *simultaneous inductive-recursive definitions* including definitions of a variety of universes which were not accounted for by the old schema. It also gives rise to other interesting notions including a constructive version of Aczel's Frege structures.

Universes in type theory are analogous to Grothendieck universes in set theory: they are sets of "small" sets and can be used for example for the formalisation of constructive category theory. Another interesting application of universes (here in conjunction with generalised inductive definitions) is Aczel's universe of iterative sets, in which a constructive version of Zermelo-Fraenkel set theory **CZF** can be interpreted [3]. But in the standard formulations of type theory universes are needed also for the more basic purpose of defining families of sets by structural recursion. For example, the predicate Z (as used

in the type-theoretic proof that $0 \neq s(n)$ [29, 43]) with the recursion equations

$$\begin{aligned} Z(0) &= \top, \\ Z(s(n)) &= \bot \end{aligned}$$

is defined in terms of universes and the rule of N-elimination.

Martin-Löf [27] introduced an infinite tower of universes $U_0 : U_1 : U_2 : \cdots$. These were formulated "à la Russell" [29], which means that there is no syntactic distinction between a small set considered as an element of a universe $A : U_i$ and considered as a set A. In contrast, the formulation "à la Tarski" [29] maintains such a distinction: $a : U_i$ is a *code* for the set $T_i(a)$. A universe à la Tarski should therefore be understood as a pair (U_i, T_i) consisting of a set U_i of codes and a decoding function T_i.

Further universes were introduced by Palmgren [35]. Firstly, he defined a *universe operator*, that is, an operator on families of sets which when applied to a universe (U_i, T_i) returns the next universe (U_{i+1}, T_{i+1}). In this way the external sequence of universes à la Tarski $(U_0, T_0), (U_1, T_1), (U_2, T_2), \ldots$ was internalised. Secondly, Palmgren introduced a *super-universe* closed under the universe operator as well as under all the usual set formers in type theory. Recently, even larger universes and universe operators have been proposed by Rathjen, Griffor, and Palmgren [39], Palmgren [36], and Setzer [42].

Martin-Löf type theory with generalised inductive definitions and universes have great proof-theoretic strength, see Griffor and Rathjen [23], Setzer [41], and Rathjen, Griffor, and Palmgren [39].

We conclude this introduction with a few words about the notation. We employ the "logical framework" formulation of Martin-Löf type theory [30, 34]. The core of this theory is a typed $\lambda\beta\eta$-calculus with dependent types. There is a base type *set*, the type of sets, and for each object $A : set$, there is the type $El(A)$ (often written just A) of the elements of A. We write $(x : \alpha)\beta$ for the type of functions which map an object $a : \alpha$ to an object $f(a) : \beta[a/x]$. If β does not depend on $x : \alpha$ we may write $(\alpha)\beta$ (rather than $\alpha \to \beta$) instead of $(x : \alpha)\beta$. Abstraction is written $(x)b$ and application $f(a)$, rather than the usual notation $\lambda x.b$ and $f\ a$ from λ-calculus. We also write $(x_1, \ldots, x_n)b = (x_1) \cdots (x_n)b$ for multiple abstraction, and $f(a_1, \ldots, a_n) = f(a_1) \cdots (a_n)$ for multiple application.

In this type system $(set)(set)set$ is for example the type of binary logical connectives; $(A : set)(B : (A)set)set$ is the type of quantifiers; and $(N)set$ is the type of unary predicates on N and also the type of N-indexed families of sets.

The formulation of the schema for simultaneous induction-recursion in section 3 will use two auxiliary notions. Firstly, we will write $(a :: \alpha)\beta$ as an abbreviation of $(a_1 : \alpha_1) \cdots (a_n : \alpha_n)\beta$ and call α a sequence of types. (This abbreviation could be avoided by adding Σ-*types* to the λ-calculus with dependent types.) Secondly, we will say that a type is *small* if it contains no occurrences of *set*. (Small types are called s-types in [19].) Small types are almost like sets. But the logical framework formulation we use maintains a distinction between the set $\Pi(A, B)$ and the small type $(x : A)B(x)$. Since we want our schema to cover the rules for Π as well we need to refer to small types when formulating the requirements on the rules.

Plan of the paper. In section 2 we introduce simultaneous induction-recursion by two examples: the first universe à la Tarski and the fresh-lists (lists where all elements are distinct). In section 3 we give the general schema for simultaneous inductive-recursive definitions in type theory. We also show how to recover the first universe à la Tarski and the fresh-lists by instantiating the schema. In section 4 we discuss further universe constructions which are instances of the schema. In section 5 we show how the construction of Frege structures in type theory is yet another instance of the schema. In section 6 we build a classical Frege structure model of a restricted version of the schema using monotone inductive definitions. In section 7 we conclude.

2 Two examples of simultaneous induction-recursion

The prototypical example of simultaneous induction-recursion is Martin-Löf's definition of the first universe à la Tarski [29]. It consists of the simultaneous *inductive* definition of the set U_0 of codes for small sets and the *recursive* definition of the decoding function $T_0 : (U_0)set$. U_0 has one introduction rule (and one equality rule) for each set former which is reflected in the universe. We here give two examples:

Π-formation is reflected by the following rule of U_0-introduction:

$$\pi_0 : (u : U_0)(u' : (x : T_0(u))U_0)U_0,$$

and we have the following equality rule:

$$T_0(\pi_0(u, u')) = \Pi(T_0(u), (x)T_0(u'(x))).$$

Eq-formation is reflected by the following rule of U_0-introduction:

$$eq_0 : (u : U_0)(b, b' : T_0(u))U_0,$$

and we have the following equality rule:

$$T_0(eq_0(u, b, b')) = Eq(T_0(u), b, b').$$

This definition does not fit the schema for strictly positive inductive definitions in Dybjer [19], since T_0 appears (even negatively) in the introduction rules for U_0. In spite of this one can justify that it is a *predicative* definition in Martin-Löf's sense [27]. Allen [5] has suggested calling this kind of definition *half-positive*.

For example, the rules for π_0 stipulate the following way of constructing new elements of U_0. At a certain stage we may have constructed an element u. Since T_0 is defined by U_0-recursion, we can compute the set $T_0(u)$. Hence we can construct a function u' with domain $T_0(u)$ and range (the presently constructed elements of) U_0. Hence we can construct an element $\pi_0(u, u')$. Moreover, the computation of $T_0(\pi_0(u, u'))$ will terminate since we already know that the computation of $T_0(u)$ and of $T_0(u'(x))$ for $x : T_0(u)$ will terminate and that the $T_0(u'(x))$'s will only be called a finite number of times.

A more down-to-earth example (due to Catarina Coquand) is the following inductive definition of the set of lists *Dlist* with distinct elements and the simultaneous definition of the freshness relation *Fresh*.

Let A be the set from which the elements of the list are drawn, and let $a\#b$ mean that a and b are different elements of A, for some difference relation $\# : (A)(A)set$. We have the following formation rules

$$\begin{array}{rcl} Dlist & : & set, \\ Fresh & : & (Dlist)(A)set \end{array}$$

and introduction rules

$$\begin{array}{rcl} nil & : & Dlist, \\ cons & : & (b : A)(u : Dlist)(b' : Fresh(u, b))Dlist. \end{array}$$

Fresh has the equality rules

$$\begin{array}{rcl} Fresh(nil, a) & = & \top, \\ Fresh(cons(b, u, b'), a) & = & b\#a \wedge Fresh(u, a). \end{array}$$

Note that *cons* has an extra argument b', which is a proof that the new element b is fresh with respect to u. Hence the introduction rules for *Dlist* refer to *Fresh*. But as for the case of the first universe we shall argue that it is a good predicative definition.

For example, the rules for *cons* stipulate the following way of constructing new elements of *Dlist*. At a certain stage we may have constructed $u : Dlist$. Since *Fresh* is defined by *Dlist*-recursion, we already know what it means for an element $b : A$ to be fresh with respect to u, that is, we know what a proof $b' : Fresh(u, b)$ is. Hence it makes sense to construct an element $cons(b, u, b')$. Moreover, we can define $Fresh(cons(b, u, b'))$ in terms of the already constructed proposition $Fresh(u)$.

There are of course a number of alternative ways to define *Dlist* and *Fresh* using ordinary inductive definitions, but the inductive-recursive one seems natural and may be preferred for some purposes.

In both examples we simultaneously build a function and its domain of definition. This intuition can be captured using a classical notion of monotone inductive definition and thus yield a consistency proof. This is the idea behind the realisability model in section 5.

A set former in Martin-Löf type theory is specified by its formation and introduction rules. Moreover, in the standard formulation [28, 29] it has an elimination rule which expresses a general principle of definition of a function by structural recursion. A particular function defined by structural recursion (for example the addition function on natural numbers) is then obtained by instantiating the elimination rule in question. Its recursion equations are then derived by instantiating the equality rules. But it is also possible to formulate definition by structural recursion by an external schema (so that for example the addition function is obtained as an instance) as in Martin-Löf [27], Coquand [14], and Dybjer [19].

When considering simultaneous inductive-recursive definitions it is essential to adopt the latter approach (using an external schema). The reason is that the elimination rule expresses only definition by structural recursion on a previously (and not simultaneously) defined set. For example, the definition of T_0 is an instance of a recursive schema. It cannot be formulated in terms of U_0-elimination, because already the formulation of U_0-elimination refers to T_0.

Moreover, we want the schema for simultaneous induction-recursion to specialise to universe constructions. It is therefore essential that we do not require that the value of a function defined by recursion necessarily is an element of a set (as in the traditional elimination rules). Instead the value can be an object of an arbitrary type (as in the "large" elimination rules [44, 45]. For example, the value may be a set (an object of the type *set*), so we have *recursively defined families of sets.*

Note that we have reversed the priority of the following concepts as compared to the standard formulations of Martin-Löf type theory [28, 29, 34]:

- *Elimination and equality rules* are special instances of the *recursive schemata*, whereas in the standard formulation the recursive schemata are derived from the elimination and equality rules.
- *Universes* are special kinds of simultaneous inductive-recursive definitions employing *set-valued recursion*, whereas in the standard formulation set-valued recursion is obtained from the elimination and equality rules in conjunction with universes.

3 Formalising the notion of a simultaneous inductive-recursive definition

We use the schematic style of Martin-Löf's intuitionistic theory of iterated inductive definitions in predicate logic [26] for presenting the theory of simultaneous inductive-recursive definitions in type theory. As already mentioned the present schema is a generalisation of the schema in Dybjer [19, 20]. The main difference is that the extended schema for an introduction rule may refer to a function defined simultaneously by recursion. To highlight the similarity between the extended schema and the old schema we use the same notation here as in Dybjer [20].

The present description could form the basis for an implementation in the same way as the old schema [20] is the basis of Giménez' [22] implementation of inductive definitions in a proof editor for Martin-Löf type theory, and as Coquand and Paulin's formulation of inductive types in the calculus of constructions [16, 38] is the basis for the Coq-system [18].

To illustrate the schema, we show how the rules for the first universe and the fresh-lists can be derived by instantiation. Later sections contain further examples. It might be helpful to study these examples before studying the general formulation given in this section.

I present the case with one inductive and one recursive definition. Clearly, the schema can be generalised to the case with several simultaneous inductive and several (possibly zero) recursive definitions, but we will not spell out the details.

To begin with (in 3.1-3.4) we assume that there are no parameters. In 3.5 it is shown how to extend the schema to simultaneous inductive-recursive definitions with parameters.

A definition is always relative to a theory containing the rules for previously defined concepts. Thus the requirements on the different parts of the definitions ($\alpha, \psi, \beta, \xi, p, q$ below) are always judgements with respect to that theory.

3.1 Formation rules

Schema. Let α be a sequence of small types. A simultaneous inductive-recursive definition of an α-

indexed family of sets P and an α-indexed family of functions f defined by P-recursion has formation rules of the form

$$\begin{aligned} P &: (a :: \alpha)set, \\ f &: (a :: \alpha)(c : P(a))\psi[a]. \end{aligned}$$

Here we also require that $\psi[a]$ is a type under the assumptions $a :: \alpha$.

Examples. The formation rules for the first universe

$$\begin{aligned} U_0 &: set, \\ T_0 &: (c : U_0)set \end{aligned}$$

are obtained by letting P be U_0, f be T_0, α be the empty sequence, and $\psi[c]$ be set.

The formation rules for fresh-lists

$$\begin{aligned} Dlist &: set, \\ Fresh &: (c : Dlist)(a : A)set \end{aligned}$$

are obtained by letting P be $Dlist$, f be $Fresh$, α be empty and $\psi[c]$ be $(a : A)set$ (the type of predicates on A).

For an example where α is a non-empty sequence, see section 5 on Frege structures, where the properties of internal propositionality and internal truth are defined.

3.2 Introduction rules

Schema. A premise of an introduction rule is either *non-recursive* or *recursive*.

- A non-recursive premise has the form
$$b : \beta,$$
where β is a small type depending on the previous premises of the rule (see below).

- A recursive premise has the form
$$u : (x :: \xi)P(p[x]),$$
where ξ is a sequence of small types, and $p[x] :: \alpha$ assuming $x :: \xi$ and the previous premises of the rule (see below). If ξ is empty the premise is called *ordinary* and otherwise *generalised* (as in ordinary and generalised induction).

The type of the conclusion of the introduction rule has the form

$$P(q),$$

where $q :: \alpha$ depending on the premises of the rule (see below).

We shall now spell out the typing criteria for β in the schema above. (The criteria for ξ, p, and q are analogous.) These criteria may seem complex at a first glance, but are nothing but what results from spelling out the obvious possible dependencies that can occur.

We write $\beta = \beta[\ldots, b', \ldots, u', \ldots]$, etc., to explicitly indicate the dependence on typical previous non-recursive $b' : \beta'$ and recursive $u' : (x' :: \xi')P(p'[x'])$ premises. (Non-recursive and recursive premises may appear in any order). The dependence on a previous recursive premise can occur only through an application of the simultaneously defined function f. Formally, this means that we require that

$$\beta[\ldots, b', \ldots, u', \ldots] = \hat{\beta}[\ldots, b', \ldots, (x')f(p'[x'], u'(x')), \ldots],$$

where $\hat{\beta}[\ldots, b', \ldots, v', \ldots]$ is a small type in the context $(\ldots, b' : \beta', \ldots, v' : (x' :: \xi')\psi[p'[x']], \ldots)$. Note that this context is obtained from the context of β by replacing each recursive premise of the form $u' : (x' :: \xi')P(p'[x'])$ by $v' : (x' :: \xi')\psi[p'[x']]$.

In the sequel we will write

$$intro : \cdots (b : \beta) \cdots (u : (x :: \xi)P(p[x])) \cdots P(q)$$

for the general form of an introduction rule. It indicates that a typical constructor *intro* may have non-recursive premises (arguments) $b : \beta$ and recursive premises $u : (x :: \xi)P(p[x])$. There may be zero or more premises of either kind and they may appear in arbitrary order.

If we remove the possibility that β, ξ, p, and q depend on previous *recursive* premises, then we essentially recover the schema in Dybjer [20], because then f cannot appear in the introduction rules for P. Moreover, since a non-recursive premise then cannot depend on a recursive one, we can without loss of generality assume that all non-recursive premises precede the recursive premises.

Examples. The introduction rule for U_0 which reflects Π-formation

$$\pi_0 : (u : U_0)(u' : (x : T_0(u))U_0)U_0$$

is obtained in the following way. The first premise $u : U_0$ is recursive and ordinary, that is, ξ in the schema is empty. The second premise $u' : (x : T_0(u))$ is recursive and generalised and depends on the first recursive premise. It is obtained from the schema by letting $\xi[u] = \hat{\xi}[T_0(u)] = T_0(u)$, that is, $\hat{\xi}[v] = v$.

The introduction rule reflecting Eq-formation is

$$eq_0 : (u : U_0)(b, b' : T_0(u))U_0.$$

The first premise $u : U_0$ is recursive and ordinary. The second and and third premises $b, b' : T_0(u)$ are non-recursive and depend on the first recursive premise. They are obtained by letting $\beta[u] = \hat{\beta}[T_0(u)] = T_0(u)$, that is, $\hat{\beta}[v] = v$.

The second introduction rule for *Dlist* is

$$cons \quad : \quad (b : A)(u : Dlist)(b' : Fresh(u, b))Dlist.$$

It has a first non-recursive premise with $\beta = A$. The second premise is recursive and ordinary (ξ is empty). The third premise is non-recursive with $\beta[b, u] = \hat{\beta}[b, Fresh(u)] = Fresh(u, b)$ so $\hat{\beta}[b, v] = v(b)$.

3.3 Equality rules for the simultaneously defined function

Schema. Let *intro* be a constructor and let as above $b : \beta$ and $u : (x :: \xi)P(p[x])$ be typical non-recursive and recursive premises respectively of the corresponding introduction rule. The form of the equality rule for f and *intro* is:

$$f(q, intro(\ldots, b, \ldots, u, \ldots)) = e(\ldots, b, \ldots, (x)f(p[x], u(x)), \ldots) : \psi[q]$$

in the context

$$(\ldots, b : \beta, \ldots, u : (x :: \xi)P(p[x]), \ldots)$$

where

$$e(\ldots, b, \ldots, v, \ldots) : \psi[q]$$

in the context

$$(\ldots, b : \beta, \ldots, v : (x :: \xi)\psi[p[x]], \ldots).$$

Examples. The equality rules for T_0

$$\begin{aligned} T_0(\pi_0(u, u')) &= \Pi(T_0(u), (x)T_0(u'(x))), \\ T_0(eq_0(u, b, b')) &= Eq(T_0(u), b, b') \end{aligned}$$

are obtained by letting $e(v, v') = \Pi(v, v')$ and $e(v, b, b') = Eq(v, b, b')$ respectively.

The equality rules for *Fresh*

$$\begin{aligned} Fresh(nil) &= (a)\top, \\ Fresh(cons(b, u, b')) &= (a)(b\#a \wedge Fresh(u, a)) \end{aligned}$$

are obtained by letting $e = (a)\top$ and $e(b, v, b') = (a)(b\#a \wedge v(a))$ respectively.

3.4 A generalisation of universe elimination

Schema. Universe elimination (as described in Nordström, Petersson, and Smith [34]) expresses definition by U_0-recursion *after* U_0 and T_0 are defined. Here we express the corresponding notion schematically. We also show that ordinary universe elimination is a special case of this schema.

In general, after the simultaneous inductive-recursive definition of P and f has been completed, we may define a new function

$$f' : (a :: \alpha)(c : P(a))\psi'[a, c],$$

by P-recursion. Here we require that $\psi'[a, c]$ is a type in the context $(a :: \alpha, c : P(a))$.

The equality rule has the form

$$f'(q, intro(\ldots, b, \ldots, u, \ldots)) = e'(\ldots, b, \ldots, u, (x)f'(p[x], u(x)), \ldots)$$

in the context

$$(\ldots, b : \beta, \ldots, u : (x :: \xi)P(p[x]), \ldots)$$

where

$$e'(\ldots, b, \ldots, u, v, \ldots) : \psi'[q, intro(\ldots, b, \ldots, u, \ldots)]$$

in the context

$$(\ldots, b : \beta, \ldots, u : (x :: \xi)P(p[x]), v : (x :: \xi)\psi'[p[x], u(x)], \ldots)$$

Note that the criteria are identical for a simultaneously defined function f and a function f' defined afterwards, except that the target type ψ' of f' (in contrast to ψ of f) may depend on c as well as on a, and that the RHS of a recursion equation e' for f' (in contrast the RHS of a recursion equation e for f) may depend on u as well as on v. This is simply because these new dependencies can occur only after P has been defined.

Examples. Ordinary U_0-elimination

$$\begin{array}{lll} U_0rec & : & (C : (U_0)set) \\ & & (e : (u : U_0)(v : C(u))(u' : (x : T_0(u))U_0)(v' : (x : T_0(u))C(u'(x)))C(\pi_0(u, u'))) \\ & & (e' : (u : U_0)(v : C(u))(b, b' : T(u))C(eq_0(u, b, b'))) \\ & & \vdots \\ & & (c : U_0)C(c) \end{array}$$

with

$$\begin{array}{rcl} U_0rec(C, e, e', \pi_0(u, u')) & = & e(u, U_0rec(C, e, e', u), u', (x)U_0rec(C, e, e', u'(x))), \\ U_0rec(C, e, e', eq_0(u, b, b')) & = & e'(u, U_0rec(C, e, e', u), b, b'), \\ & \vdots & \end{array}$$

is obtained from the general schema by letting $C, e, e', \ldots$ be parameters (see section 3.5) and $\psi'[c] = C(c)$.

Another example is the function

$$length : (c : Dlist)N$$

which computes the length of a *Dlist* by *Dlist*-recursion. The typing rule of *length* is obtained by letting $\psi'[c] = N$ in the general schema. It has the equality rules

$$\begin{array}{rcl} length(nil) & = & 0, \\ length(cons(b, u, b')) & = & s(length(u)). \end{array}$$

These are obtained by letting $e' = 0$ and $e'(b, u, v, b') = s(v)$ respectively in the general schema.

3.5 Parameters

The simultaneous inductive-recursive definition of *Dlist* and *Fresh* depends on the parameters $A : set$ and $\# : (A)(A)set$. If we make these dependencies explicit in the formation rules for *Dlist* and *Fresh* they become

$$\begin{aligned} Dlist &: (A : set)(\# : (A)(A)set)set, \\ Fresh &: (A : set)(\# : (A)(A)set)(c : Dlist(A, \#))(a : A)set. \end{aligned}$$

We also need to modify the introduction rules accordingly:

$$\begin{aligned} nil &: (A : set)(\# : (A)(A)set)Dlist(A, \#), \\ cons &: (A : set)(\# : (A)(A)set)(b : A)(u : Dlist(A, \#))(b' : Fresh(A, \#, u, b))Dlist(A, \#). \end{aligned}$$

We can extend the schema to account for parameters in general. The rule is simple: there can be zero or more parameters and they can have arbitrary types. The modified schema for formation rules is therefore

$$\begin{aligned} P &: (A :: \sigma)(a :: \alpha[A])set, \\ f &: (A :: \sigma)(a :: \alpha[A])(c : P(A, a))\psi[A, a]. \end{aligned}$$

where σ is an arbitrary sequence of types, where $\alpha[A]$ is a sequence of small types under the assumptions $A :: \sigma$ and $\psi[A, a]$ is a type under the assumptions $A :: \sigma, a :: \alpha$. The modified schema for an introduction rule is

$$intro : (A :: \sigma) \cdots (b : \beta[A]) \cdots (u : (x :: \xi[A])P(A, p[A, x])) \cdots P(A, q)$$

where σ is the same as for the formation rules, and where the requirements on β, ξ, p, and q depend on $A :: \sigma$ as well.

A recursive definition of a function f' (defined by recursion on P after the simultaneous inductive-recursive definition of P and f) may also depend on parameters. This was used above when showing that ordinary universe elimination is an instance of the schema in section 2.4. Again, there may be zero or more parameters, and these parameters may have arbitrary types. Note however that the parameters of f' need not coincide with the parameters for the inductive-recursive definition of P and f.

4 Universe hierarchies and super-universes

In this section we review the universe hierarchies and super-universes of Palmgren [35]. Palmgren presented these constructions in a one-off fashion with informal motivations for the rules. Here we use our notion of a simultaneous inductive-recursive definition to give a unified formal treatment of all these constructions: as for the rules for U_0 we can recover them by instantiating the appropriate part of the general schema above.

4.1 External universe hierarchies

The *second universe* has formation rules

$$\begin{aligned} U_1 &: set, \\ T_1 &: (U_1)set, \end{aligned}$$

and analogous rules for the constructors π_1 and eq_1 to those for π_0 and eq_0. There is also an introduction and equality rule reflecting U_0-formation:

$$\begin{aligned} u_{01} &: U_1, \\ T_1(u_{01}) &= U_0. \end{aligned}$$

We also wish to reflect T_0 as a function into the second universe:

$$t_{01} : (U_0)U_1,$$

and therefore we let t_{01} be a constructor for U_1. (Palmgren also mentions the possibility of defining t_{01} by recursion on U_0.) The equality rule for T_1 is:

$$T_1(t_{01}(b)) = T_0(b).$$

We can continue in an analogous way and define U_2 and T_2, U_3 and T_3, etc. and thus get an external universe hierarchy.

4.2 An internal universe hierarchy and a super-universe

The construction of U_{n+1} and T_{n+1} from U_n and T_n can be internalised. We give a simultaneous inductive-recursive definition of the set formers

$$\begin{aligned} Nextu &: (U : set)(T : (U)set)set, \\ Nextt &: (U : set)(T : (U)set)(Nextu(U,T))set, \end{aligned}$$

and let $U_{n+1} = Nextu(U_n, T_n)$ and $T_{n+1} = Nextt(U_n, T_n)$.

U and T are parameters of this definition. For simplicity, we suppress these parameters in the rules for *Nextu* and *Nextt* and write

$$\begin{aligned} Nextu &: set, \\ Nextt &: (Nextu)set. \end{aligned}$$

Introduction and equality rules correspond to those of the first universe, but we also need to reflect the code set U and the decoding function T:

$$* : Nextu,$$

$$Nextt(*) = U,$$

$$t : (b : U)Nextu,$$

$$Nextt(t(b)) = T(b).$$

A *super-universe* is obtained by also reflecting the next-universe construction inside a set U_∞.

The formation rules are

$$\begin{aligned} U_\infty &: set, \\ T_\infty &: (U_\infty)set. \end{aligned}$$

Introduction and equality rules correspond to those for the first universe, but we also need to reflect the first universe U_0 and the next-universe construction *Nextu*

$$u_0 : U_\infty,$$

$$T_\infty(u_0) = U_0,$$

$$nextu : (u : U_\infty)(u' : (T_\infty(u))U_\infty)U_\infty,$$

$$T_\infty(nextu(u, u')) = Nextu(T_\infty(u), (x)T_\infty(u'(x))).$$

It is straightforward to check that also these rules follow the general schema.

4.3 A parameterised super-universe

The construction of the super-universe U_∞ can be generalised. Instead of starting with the next-universe operator, we can start with an arbitrary operator given by a pair

$$\begin{aligned} F &: (U : set)(T : (U)set)set, \\ G &: (U : set)(T : (U)set)(F(U,T))set, \end{aligned}$$

which maps a family of sets (U, T) into another family of sets $(F(U,T), (x)G(U,T,x))$. Then we can construct a super-universe (U_P, T_P) closed under this operator by a simultaneous inductive-recursive definition. The definition of (U_P, T_P) is analogous to the definiton of (U_∞, T_∞); the only difference is that we replace $Nextu$ by the parameter F. If we make the dependence on the parameters F and G explicit in the types of U_P and T_P we get

$$\begin{aligned} U_P &: (F : (U : set)(T : (U)set)set) \\ &\quad (G : (U : set)(T : (U)set)(F(U,T))set) \\ &\quad set, \end{aligned}$$

$$\begin{aligned} T_P &: (F : (U : set)(T : (U)set)set) \\ &\quad (G : (U : set)(T : (U)set)(F(U,T))set) \\ &\quad (c : U_P(F,G)) \\ &\quad set. \end{aligned}$$

I learned about this generalization and its connection to Mahlo cardinals in set theory from Anton Setzer. For a complete account and more discussion, see Setzer [42]. (The reader should note that there are some differences between Setzer's formalisation and the one suggested here.)

Several more examples of simultaneous inductive-recursive definitions of large universes and universe operators can be found in the recent papers by Palmgren [36] and Rathjen, Griffor, and Palmgren [39].

5 Frege structures

The notion of a *Frege structure* was introduced by Peter Aczel [4]. One purpose was to provide an appropriate setting for λ-calculus (or abstract realisability) interpretations of Martin-Löf type theory. Another was to provide a model for a foundational framework where the notions of "proposition" and "truth" are primitive.

We shall here show how to construct a Frege structure in Martin-Löf type theory by using a simultaneous inductive-recursive definition. Thus we show a way to reduce this foundational framework to the foundational framework of type theory. This type-theoretic construction can be contrasted to Aczel's construction of Frege structures in classical set theory.

The notion of a Frege structure is an enrichment of the notion of a *λ-structure*. This is essentially the same as a *λ-model* in Barendregt [8] but is expressed in terms of an *explicitly closed family* $\mathcal{F}$. Here $\mathcal{F}_0$ is the set of *objects* of the Frege structure and $\mathcal{F}_n$ is a set of n-ary functions on $\mathcal{F}_0$, which is used for interpreting expressions with at most n free variables. A Frege structure comes with projection functions $\pi^i_n : \mathcal{F}_n$ and $\mathcal{F}$-functionals $\lambda_n : \mathcal{F}_{n+1} \to \mathcal{F}_n$ and $App_n : \mathcal{F}_n \times \mathcal{F}_n \to \mathcal{F}_n$ satisfying appropriate equations.

The objects of a Frege structure are used both for encoding propositions and other mathematical objects. Given a λ-structure we can encode the logical constants: a binary connective can be encoded as a binary function on $\mathcal{F}_0$, and a quantifier can be encoded as a function from $\mathcal{F}_1$ to $\mathcal{F}_0$.

A Frege structure is a λ-structure with encodings of logical constants together with a set of objects called *propositions* and a subset of these called *truths*. These collections have to satisfy *logical schemata*, such as [4, page 37]:

> Implication: If a is a proposition and the object b is a proposition provided that a true, then $a \supset b$ is a proposition, such that $a \supset b$ is true iff a is true implies b is true.

Universal quantification: If f is a propositional function in $\mathcal{F}_1$, then $\forall x f(x)$ is a proposition, such that $\forall x f(x)$ is true iff $f(a)$ is true for all objects a.

Note that $a \supset b$ is a non-standard notion of implication, since b is required to be a proposition only when a is true.

A λ-structure with an encoding of the logical constants can be obtained by a standard construction in the λ-calculus [4]. To get a Frege structure we need to construct the collections of propositions and truths. The basic idea is to view the logical schemata as inductively defining these collections. But a direct interpretation as a positive inductive definition is not possible, since the notion of truth appears negatively in the logical schema for implication. Instead, Aczel [4] showed that one can interpret the logical schemata as an operator on pairs of sets of objects, which is monotone with respect to the following "conservative extension" ordering:

$$\langle \mathcal{P}, \mathcal{T} \rangle \leq \langle \mathcal{P}', \mathcal{T}' \rangle \textit{ iff } \mathcal{P} \subseteq \mathcal{P}' \textit{ and } \mathcal{T} = \mathcal{T}' \cap \mathcal{P}.$$

Intuitively, when constructing new propositions, the notion of truth on old propositions should remain the same. Since this ordering is complete it follows by a standard argument that the operator has a least fixed point when working in classical set theory.

However, Aczel also made the following remark [4, page 55] which has provided motivation for the present work:

> Nevertheless I believe this result to be constructively valid. A rigorous elaboration of this point would require an explicit discussion of the role of inductive definitions in constructive mathematics. It will have to suffice here if I simply assert that the logical schemata form the clauses of an inductive definition that generate the propositions and simultaneously give conditions for their truth.

The standard construction of a λ-structure with set of objects $\mathcal{F}_0$ and an explicit equivalence relation on objects $\sim_0$, together with an encoding of the logical constants can be carried out in type theory (see for example Hedberg [24] for a formal development of constructive domain theory inside type theory).

It remains to turn the logical schemata into a simultaneous inductive definition of the property $\mathcal{P} : (a : \mathcal{F}_0)set$ of propositionality and a recursive definition of the truth of a proposition $\mathcal{T} : (a : \mathcal{F}_0)(c : \mathcal{P}(a))set$. Note that these formation rules are obtained by instantiating the schema with $P = \mathcal{P}, f = \mathcal{T}$, α the singleton sequence $\mathcal{F}_0$, and $\psi[a] = set$. It is essential that $\mathcal{T}$ has a second argument, the proof that an object is a proposition, since it is defined by recursion on that proof.

We also show the introduction rules corresponding to the logical schemata for implication and universal quantification. We first extract from the λ-structure a function $ev : (\mathcal{F}_1)(\mathcal{F}_0)\mathcal{F}_0$ which applies a unary function to an object, and codes $\supset$: $(\mathcal{F}_0)(\mathcal{F}_0)\mathcal{F}_0$ and $\forall : (\mathcal{F}_1)\mathcal{F}_0$ for implication and universal quantification. The rules are:

$$\supset\textit{-intro} : (a : \mathcal{F}_0)(p : \mathcal{P}(a))(b : \mathcal{F}_0)(q : (\mathcal{T}(a,p))\mathcal{P}(b))(c : \mathcal{F}_0)(z : c{\sim_0}(a \supset b))\mathcal{P}(c),$$

$$\mathcal{T}(c, \supset\textit{-intro}(a,p,b,q,c,z)) = \Pi t : \mathcal{T}(a,p).\mathcal{T}(b,q(t)),$$

$$\forall\textit{-intro} : (f : \mathcal{F}_1)(p : (x : \mathcal{F}_0)\mathcal{P}(ev(f,x)))(c : \mathcal{F}_0)(z : c{\sim_0}\forall(f))\mathcal{P}(c),$$

$$\mathcal{T}(c, \forall\textit{-intro}(f,p,c,z)) = \Pi x : \mathcal{F}_0.\mathcal{T}(ev(f,x), p(x)).$$

Note that these definitions ensure that $\sim_0$ is preserved by $\mathcal{P}$ and $\mathcal{T}$. This can be shown formally by using the analogue of universe elimination.

The main point is that the logical schemata directly can be interpreted as a basic kind of definition, specifically, as a simultaneous inductive-recursive definition which is an instance of our schema. We also note the similarity between this understanding and the intuitive reading of Aczel's construction as a conservative extension ordering: since $\mathcal{T}$ is a function, the truth of a proposition cannot change once it is defined.

We therefore argue that we have shown a way to make Aczel's claim, that his result is constructively valid, precise. Essential is our use of explicit proof-objects and the fundamental difference between an inductive and a recursive definition in type theory.

6 Realisability model

To prove the consistency we shall construct a realisability model. We first show how to interpret an arbitrary instance of the schema in 3.1-3.4. This interpretation also applies to definitions with parameters as specified in 3.5 provided these parameters satisfies a particular positivity criterion, see section 6.4.4.

We follow Aczel [2, 4] and interpret types as collections of objects of a Frege structure. But instead of first building collections of propositions and truths as in Aczel's work, we shall directly construct the collections of sets $\mathcal{S}et$ and elements $\mathcal{E}l[A]$ (for each $A \in \mathcal{S}et$) in a manner similar to Allen [5]. These collections will be inductively defined using Aczel's rule sets [1].

Like Aczel, we use 'object' to refer to an element of a λ-structure and 'collection' to refer to a subset of the elements of a λ-structure. Furthermore, since substitution in the λ-calculus is interpreted as set-theoretic application in the λ-structure, we will use the notation $f[x]$ for set-theoretic application in general.

We begin by reviewing Aczel's rule sets in 6.1. Then we review the interpretation of the logical framework in 6.2. The interpretation of sets is overviewed in 6.3. In 6.4 we then give the interpretation of the general schema for inductive-recursive definitions. Finally, we give an example theory in 6.5 and show how to instantiate the schematic interpretation in 6.4 to give an interpretation of this example theory. It is probably better to study the example interpretation in 6.5 before reading the interpretation of the general schema in 6.4.

6.1 Rule sets

We use Aczel's [1] set-theoretic notion of rule set for defining these collections. It is defined as follows.

A *rule* on a base set V in Aczel's sense is a pair of sets $\langle u, v\rangle$, often written

$$\frac{u}{v},$$

such that $u \subseteq V$ and $v \in V$.

Let Φ be a set of rules on V. A set w is Φ-*closed* if

$$\frac{u}{v} \in \Phi \wedge u \subseteq w \supset v \in w.$$

There is a least Φ-closed set

$$\mathcal{I}(\Phi) = \bigcap\{w \subseteq V | w \ \Phi\text{-}closed\},$$

the set inductively defined by Φ.

Each rule set Φ on V generates a monotone operator

$$\phi(X) = \{v \in V | \exists \frac{u}{v} \in \Phi . u \subseteq X\}$$

on $\mathcal{P}ow(V)$, such that $\mathcal{I}(\Phi)$ is the least fixed point of ϕ.

We use rule sets rather than monotone operators here, since they allow a more direct encoding of type-theoretic inductive definitions expressed in terms of introduction rules.

6.2 Interpretation of the logical framework

We first briefly review the interpretation of the logical framework in a λ-structure satisfying the β- and the η-rule.

The open terms of the logical framework are interpreted as follows. Abstraction and application are interpreted in terms of the $\mathcal{F}$-functionals λ_n and App_n in the λ-structure. Variables are interpreted as projections π^i_n. For reasons of presentation we shall in the sequel use the same notation for a term and its interpretation in the λ-structure: $(x)b$ for $[\![(x)b]\!]$, $c(a)$ for $[\![c(a)]\!]$ and x for $[\![x]\!]$.

Using this notation we interpret the judgements of type theory relative to an environment ρ as follows:

$$\begin{array}{lll} [\![\alpha \ type]\!]_\rho & \text{is} & [\![\alpha]\!]_\rho \subseteq \mathcal{F}_0, \\ [\![\alpha = \beta]\!]_\rho & \text{is} & [\![\alpha]\!]_\rho = [\![\beta]\!]_\rho, \\ [\![a : \alpha]\!]_\rho & \text{is} & a_\rho \in [\![\alpha]\!]_\rho, \\ [\![a = b : \alpha]\!]_\rho & \text{is} & a = b. \end{array}$$

$\mathcal{F}_0$ is the set of objects of the λ-structure (as in the section on Frege structures) and a_ρ is the result of applying the denotation $a \in \mathcal{F}_n$ to the sequence of n objects given by ρ. If the judgement in question is made under the assumptions (in the context)

$$x_1 : \alpha_1, \ldots, x_n : \alpha_n$$

then ρ assigns objects to variables as follows:

$$x_1 = a_1 \in [\![\alpha_1]\!]_{\rho_0}, \ldots, x_n = a_n \in [\![\alpha_n]\!]_{x_1=a_1,\ldots,x_{n-1}=a_{n-1}},$$

where ρ_0 is the empty environment.

The interpretation of a function type is

$$[\![(x : \alpha)\beta]\!]_\rho = \{c \in \mathcal{F}_0 | \forall a \in [\![\alpha]\!]_\rho . c(a) \in [\![\beta]\!]_{\rho, x=a}\}.$$

One can verify that all rules of the logical framework hold under this interpretation.

6.3 Overview of the interpretation of inductive and recursive definitions

We shall now show how to interpret sets and functions introduced by (possibly simultaneous) inductive and recursive definitions.

We call a specific sequence of definitions a *theory*. Recall from section 3 that the correctness of a given simultaneous inductive-recursive definition is always given relative to the sequence of prior definitions. Also recall the following:

- An inductive definition is given by a sequence of typings of constants for a set constructor and its element constructors.
- A recursive definition is given by a typing of a function constant together with its recursion equations.
- A simultaneous inductive-recursive definition is given by a sequence of typings of constants for a set constructor, a function constant, and the element constructors together with the recursion equations for the function constant.

The interpretation of an arbitrary theory following the schema has two parts.

1. Each *constant* of the theory is interpreted as an object of the λ-structure in such a way that all *recursion equations* (equality rules) are satisfied. (Given an interpretation of the constants we can construct the interpretation of an arbitrary term and it is thus clear what it means that an equation is satisfied in the model.)
2. The *base types set* and $El(A)$ are then interpreted as inductively defined collections $\mathcal{S}et$ and $\mathcal{E}l[A]$.

   $\mathcal{S}et$ and $\mathcal{E}l[A]$ are built up in stages. Assuming that the theory contains m *inductive* definitions (counting also the inductive parts of the inductive-recursive definitions), we construct a sequence of interpretations

   $$\emptyset = \mathcal{E}l_0 \subseteq \cdots \subseteq \mathcal{E}l_m = \mathcal{E}l \subseteq \mathcal{F}_0 \times \mathcal{P}ow(\mathcal{F}_0)$$

   Each $\mathcal{E}l_i$ is defined inductively. It will be the case that each $\mathcal{E}l_i$ is a functional relation, and we define $\mathcal{S}et_i$ as the domain of $\mathcal{E}l_i$. Thus we get a sequence

   $$\emptyset = \mathcal{S}et_0 \subseteq \cdots \subseteq \mathcal{S}et_m = \mathcal{S}et \subseteq \mathcal{F}_0$$

   Therefore each type α which is definable at stage i in the theory can be interpreted as a collection $[\![\alpha]\!]_i$ by interpreting each occurrence of Set as $\mathcal{S}et_i$ and each occurrence of $El(A)$ as $\mathcal{E}l_i[[\![A]\!]_i]$.

These two parts correspond to the following parts in Aczel [4].

1. The formation of an *independent family of $\mathcal{F}$-functionals* for the logical constants, and the use of fixed points to interpret recursion equations.
2. The construction of the collections of *propositions* and *truths*.

6.4 Interpretation of the general schema for simultaneous inductive-recursive definitions

6.4.1 Interpretation of constants

We assume that no definition has *parameters*. The interpretation of these are discussed in 6.4.4.

First we form *the list of set constructors* with associated arities, and for each inductively defined set we form its *list of element constructors* with associated arities. For each of these lists we interpret the constructors as objects of the λ-structure in such a way that there is another object *case* satisfying the recursion equations for case analysis. These have the form

$$case(d_1, \ldots, d_n, intro_i(a_1, \ldots, a_p)) = d_i(a_1, \ldots, a_p),$$

where $intro_i$ by abuse of notation is an object interpreting a constructor $intro_i$ of arity p on the list in question. It follows that (the interpretation of) constructors are injective,

$$intro_i(a_1, \ldots, a_p) = intro_i(a'_1, \ldots, a'_p) \supset a_1 = a'_1 \wedge \cdots \wedge a_p = a'_p,$$

and non-overlapping,

$$i \neq i' \supset intro_i(a_1, \ldots, a_p) \neq intro_{i'}(a'_1, \ldots, a'_{p'}).$$

We call this interpretation an *independent family of objects* for a list of constructors.

Independent families of objects can always be found, but the definability of *case* forces us to use "recursive" rather than "iterative" encodings, see the discussion by Parigot [37] and Altenkirch [6].

Secondly, for each recursively defined function constant, we construct an element satisfying the recursion equations in question. This can be done in a standard way using fixed points and case analysis. This construction is also used by Aczel [4].

6.4.2 Interpretation of base types

Assume that we have constructed $\emptyset = \mathcal{E}l_0 \subseteq \cdots \subseteq \mathcal{E}l_i$ and that the $i+1$-st definition is a simultaneous inductive definition of a family of sets

$$P : (a :: \alpha)set,$$

and recursive function

$$f : (a :: \alpha)(c : P(a))\psi[a].$$

A typical constructor is

$$intro : \cdots (b : \beta) \cdots (u : (x :: \xi)P(p[x])) \cdots P(q)$$

and a typical recursion equation (equality rule) is

$$f(q, intro(\ldots, b, \ldots, u, \ldots)) = e(\ldots, b, \ldots, (x)f(p[x], u(x)), \ldots).$$

(The case of a simple inductive definition or of a simple recursive definition are easily obtained as degenerate versions.)

$\mathcal{E}l_{i+1}$ is now defined by a rule set obtained by adding the rules

$$\{\frac{\emptyset}{\langle P(a), \mathcal{P}[a]\rangle} | a \in [\![\alpha]\!]_i\}$$

to the rule set for $\mathcal{E}l_i$. Here $\mathcal{P}$ is the interpretation of P as a family of collections indexed by $a \in [\![\alpha]\!]_i$ defined by

$$\mathcal{P}[a] = \{c \in \mathcal{F}_0 | a \mathcal{R}_P c\},$$

where $\mathcal{R}_P \subseteq [\![\alpha]\!]_i \times \mathcal{F}_0$ is an auxiliary relation between indices and objects. This relation is inductively defined by a rule set which is the union of the rule sets corresponding to each of the constructors. For example, to *intro* we associate the rule set

$$\{\frac{\cdots \cup \{\langle p[x], u(x)\rangle | x \in [\![\xi]\!]_i\} \cup \cdots}{\langle q, intro(\ldots, b, \ldots, u, \ldots)\rangle} | \ldots, b \in [\![\beta]\!]_i, \ldots, \forall x \in [\![\xi]\!]_i . u(x) \in \mathcal{F}_0, \ldots\}$$

on $[\![\alpha]\!]_i \times \mathcal{F}_0$. This rule set is well-defined, since $[\![\alpha]\!]_i, \ldots, [\![\beta]\!]_i, \ldots, [\![\xi]\!]_i$ are all well-defined. (We have not spelled out their dependencies however.)

6.4.3 Verification of the rules

The main point is that $\mathcal{E}l[A] = \mathcal{E}l_i[A]$ for $A \in \mathcal{S}et_i$, and hence for any small type α in the previous theory (with i inductively defined families of sets), we have $[\![\alpha]\!] = [\![\alpha]\!]_i$.

As before, the formation rule is interpreted as

$$a \in [\![\alpha]\!] \supset P(a) \in \mathcal{S}et$$

which can be shown to hold by inspecting the rule set for $\mathcal{E}l$.

The typical introduction rule is interpreted as

$$\cdots \supset b \in [\![\beta]\!]_i \supset \cdots \supset (\forall x \in [\![\xi]\!]_i.(p[x])\mathcal{R}_P(u(x))) \supset \cdots \supset q\mathcal{R}_P(intro(\ldots, b, \ldots, u, \ldots)),$$

which follows directly by inspecting the rule set for $\mathcal{R}_P$.

The typing rule for f is interpreted as

$$a \in [\![\alpha]\!] \supset a\mathcal{R}_P c \supset f(a, c) \in [\![\psi[a]]\!]$$

This is proved by induction on $\mathcal{R}_P$ using the recursion equations for f. For example, we need to prove that

$$q \in [\![\alpha]\!] \supset q\mathcal{R}_P(intro(\ldots, b, \ldots, u, \ldots)) \supset f(q, intro(\ldots, b, \ldots, u, \ldots)) \in [\![\psi[q]]\!]$$

from the induction hypotheses

$$p[x] \in [\![\alpha]\!] \supset \cdots \supset (\forall x \in [\![\xi]\!]_i.p[x]\mathcal{R}_P u(x)) \supset \cdots \supset f(p[x], u(x)) \in [\![\psi[p[x]]]\!]$$

But this follows directly by using the recursion equations for f.

Finally, the interpretation of the constants in 6.4.1 was specifically constructed to ensure that the equality rules for f are satisfied.

6.4.4 The interpretation of parameters

So far we have discussed the interpretation of set formers without parameters. Recall from subsection 3.5 that a set former can be parameterised with respect to an arbitrary sequence of types. It is easy to extend the interpretation to account for parameters as long as these parameters contain no occurrences of *set* in the α-part of a $(x : \alpha)\beta$. Because then we see that $\mathcal{E}l$ can be given by a monotone inductive definition using rule sets.

For an example of the interpretation of a set former with parameters, the reader is referred to the interpretation of Π in subsection 6.5.

When the preliminary version of this paper was written I was not aware of any interesting examples with such negative occurrences. But recently I learned that there indeed are several examples of such definitions of sets which are interesting as type-theoretic analogues of large cardinals. An example is the parameterised super-universe in 4.3.

It is therefore an interesting open problem to construct a classical set-theoretic model of the entire schema for simultaneous induction-recursion. The belief that the entire schema is consistent relies at present on an informal semantic analysis of the rules which shows that only well-founded elements and terminating functions can be constructed under the schema.

6.4.5 Consistency

The interpretation of $\bot$ is the empty set. Hence we have shown that any theory obtainable by successive instantiations of the schema (satisfying the restriction on parameters) is consistent relative to classical set theory.

One can also argue from an informal semantical analysis that there can be no element $a : \bot$. Because when a is evaluated it would terminate with a value of the form $intro(a_1, \ldots, a_n)$ where *intro* is a constructor for $\bot$. But $\bot$ has no constructor. This is called "simple-minded consistency" in Martin-Löf [29].

6.5 Interpretation of an example theory

We now show how to interpret an example theory. This theory is chosen to illustrate different aspects of the general situation and consists of the following parts: (i) the inductive definition of the set N of natural numbers; (ii) the recursive definition of the addition function *add*; (iii) the inductive definition of the family $N'(n)$ of finite sets indexed by the number n of elements of the set; (iv) the inductive definition of the cartesian product $\Pi(A, B)$ of a family of sets; and (v) the simultaneous inductive-recursive definition of a universe reflecting N, N', and Π.

The reader should check that the interpretation given here is indeed obtainable from the schematic interpretation given in 6.4.

6.5.1 The example theory

We list the rules of the theory. Note that we write $El(A)$ rather than just A (as before) for the type of elements of the set A. This is because the interpretation of El is the crucial part of the realisability model.

$$
\begin{array}{rcl}
N & : & set, \\
0 & : & El(N), \\
s & : & (u : El(N))El(N), \\
 & & \\
add & : & (El(N))(El(N))El(N), \\
add(m, 0) & = & m, \\
add(m, s(u)) & = & s(add(m, u)), \\
 & & \\
N' & : & (El(N))set, \\
0' & : & (b : El(N))El(N'(s(b))), \\
s' & : & (b : El(N))(u : El(N'(b)))El(N'(s(b))), \\
 & & \\
\Pi & : & (A : set)(B : (El(A))set)set, \\
\lambda & : & (A : set)(B : (El(A))set)(b : (x : El(A))El(B(x)))El(\Pi(A, B)), \\
 & & \\
U_0 & : & set, \\
T_0 & : & (El(U_0))set, \\
n_0 & : & El(U_0), \\
n_0' & : & (b : El(N))El(U_0), \\
\pi_0 & : & (u : El(U_0))(u' : (x : El(T_0(u)))El(U_0))El(U_0), \\
T_0(n_0) & = & N, \\
T_0(n_0'(b)) & = & N'(b), \\
T_0(\pi_0(u, u')) & = & \Pi(T_0(u), (x)T_0(u'(x))).
\end{array}
$$

6.5.2 Interpretation of the constants

The *list of set constructors* with arities is $N - 0, N' - 1, \Pi - 2, U_0 - 0$, and we construct an independent family of objects interpreting them.

The *lists of element constructors* with arities are

- $0 - 0, s - 1$ for N.
- $0' - 1, s' - 2$ for N'.

- $\lambda - 1$ for Π. (The first two arguments of λ are parameters, which are dropped under the interpretation: $[\![\lambda(A,B,b)]\!] = \lambda_n[[\![b]\!]]$.)
- $n_0 - 0, n_0' - 1, \pi_0 - 2$ for U_0,

and for each of these we construct an independent family of objects interpreting them.

The recursively defined functions are *add* and T_0. By definition of independent family of objects there are elements $case_N$ and $case_{U_0}$, such that

$$\begin{aligned} case_N(d_1,d_2,0) &= d_1, \\ case_N(d_1,d_2,s(u)) &= d_2(u), \\ case_{U_0}(d_1,d_2,d_3,n_0) &= d_1, \\ case_{U_0}(d_1,d_2,d_3,n_0'(b)) &= d_2(b), \\ case_{U_0}(d_1,d_2,d_3,\pi_0(u,u')) &= d_3(u,u'). \end{aligned}$$

Hence we can interpret *add* and T by solving the fixed point equations

$$\begin{aligned} add(m,n) &= case_N(m,(x)s(add(m,x)),n), \\ T_0(c) &= case_{U_0}(N,(b)N'(b),(u,u')\Pi(T_0(u),(x)T_0(u'(x))),c). \end{aligned}$$

This completes the interpretation of the constants.

6.5.3 Interpretation of the base types

We construct a sequence of functional relations

$$\emptyset = \mathcal{E}l_0 \subseteq \mathcal{E}l_1 \subseteq \cdots \subseteq \mathcal{E}l_4 = \mathcal{E}l \subseteq \mathcal{F}_0 \times \mathcal{P}ow(\mathcal{F}_0)$$

since there are 4 inductively defined sets.

Natural numbers. $\mathcal{E}l_1$ is defined by the singleton rule set

$$\{\frac{\emptyset}{\langle N, \mathcal{N}\rangle}\},$$

where $\mathcal{N}$ is the collection inductively defined by the rule set

$$\{\frac{\emptyset}{0}\} \cup \{\frac{\{a\}}{s(a)} | a \in \mathcal{F}_0\}$$

on $\mathcal{F}_0$.

The family of finite sets. $\mathcal{E}l_2$ is obtained by adding the rules

$$\{\frac{\emptyset}{\langle N'(n), \mathcal{N}'[n]\rangle} | n \in \mathcal{N}\}$$

to the rule set for $\mathcal{E}l_1$. Here

$$\mathcal{N}'[n] = \{i \in \mathcal{F}_0 | n \mathcal{R}_{N'} i\}$$

where $\mathcal{R}_{N'} \subseteq \mathcal{N} \times \mathcal{F}_0$ is an auxiliary relation between indices and objects inductively defined by the rule set

$$\{\frac{\emptyset}{\langle s(n), 0'(n)\rangle} | n \in \mathcal{N}\} \cup \{\frac{\{\langle n,i\rangle\}}{\langle s(n), s'(n,i)\rangle} | n \in \mathcal{N} \wedge i \in \mathcal{F}_0\}$$

on $\mathcal{N} \times \mathcal{F}_0$.

Cartesian product of a family of sets. $\mathcal{E}l_3$ is obtained by adding

$$\{\frac{\{\langle A, \mathcal{A}\rangle\} \cup \{\langle B(x), \mathcal{B}[x]\rangle | x \in \mathcal{A}\}}{\langle \Pi(A,B), \prod[\mathcal{A},\mathcal{B}]\rangle} | A, B \in \mathcal{F}_0, \mathcal{A} \in \mathcal{P}ow(\mathcal{F}_0), \mathcal{B} \in \mathcal{A} \to \mathcal{P}ow(\mathcal{F}_0)\}$$

to the rule set for $\mathcal{E}l_2$. Here $\prod[\mathcal{A},\mathcal{B}] \subseteq \mathcal{F}_0$ for $\mathcal{A} \subseteq \mathcal{F}_0$ and $\mathcal{B}[x] \subseteq \mathcal{F}_0$ for $x \in \mathcal{A}$ is the collection inductively defined by the rule set

$$\{\frac{\emptyset}{\lambda(b)}|\forall x \in \mathcal{A}.b(x) \in \mathcal{B}[x]\}$$

on $\mathcal{F}_0$.

The first universe. $\mathcal{E}l_4$ is inductively defined by the rule set obtained by adding

$$\{\frac{\emptyset}{\langle U_0, \mathcal{U}_0\rangle}\}$$

to the rule set for $\mathcal{E}l_3$. Here $\mathcal{U}_0$ is the collection inductively defined by the rule set

$$\{\frac{\emptyset}{n_0}\} \cup \{\frac{\emptyset}{n_0'(n)}|n \in \mathcal{N}\} \cup \{\frac{\{a\} \cup \{b(x)|x \in \mathcal{E}l_3[T_0(a)]\}}{\pi_0(a,b)}|a,b \in \mathcal{F}_0\}$$

on $\mathcal{F}_0$. (Note that this definition refers to $\mathcal{E}l_3$ and that it follows that $T_0(a) \in \mathcal{S}et_3$ for all $a \in U_0$.)

6.5.4 Verification of the rules

General rules. First we verify the rule *set type*, that is,

$$\mathcal{S}et \subseteq \mathcal{F}_0$$

which is immediate. Then both rules $El(A)$ *type* if A : *set* and $El(A) = El(A')$ if $A = A'$: *set* hold, since it is easy to prove that $\mathcal{E}l$ is functional:

$$\mathcal{E}l \in \mathcal{S}et \to \mathcal{P}ow(\mathcal{F}_0).$$

This proof relies on the interpretation of set constructors as an independent family of objects.

Natural numbers. N-formation is interpreted as

$$N \in \mathcal{S}et,$$

which can be shown by inspecting the rule set for $\mathcal{E}l$.

Since $\mathcal{E}l[N] = \mathcal{N}$, the rules of N-introduction are interpreted as

$$0 \in \mathcal{N}$$

and

$$a \in \mathcal{N} \supset s(a) \in \mathcal{N}.$$

Both follow directly by inspecting the rule set defining $\mathcal{N}$.

Addition. The typing rule for *add* is interpreted as

$$m \in \mathcal{N} \supset n \in \mathcal{N} \supset add(m,n) \in \mathcal{N}.$$

This can be shown by induction on $n \in \mathcal{N}$ using the recursion equations for *add*.

The equality rules for *add* follow directly from the recursion equations. Moreover, we can check that the typings in the equality rules are satisfied.

The family of finite sets. N'-formation is interpreted as

$$n \in \mathcal{N} \supset N'(n) \in \mathcal{S}et,$$

which can be shown by inspecting the rule set for $\mathcal{E}l$.

Since $i \in \mathcal{E}l[N'(n)]$ iff $n\mathcal{R}_{N'}i$, the rules of N'-introduction are interpreted as

$$n \in \mathcal{N} \supset (s(n))\mathcal{R}_{N'}(0'(n))$$

and

$$n \in \mathcal{N} \supset n\mathcal{R}_{N'}\, i \supset (s(n))\mathcal{R}_{N'}(s'(n,i)).$$

Both follow directly by inspecting the rule set defining $\mathcal{R}_{N'}$.

Cartesian product of a family of sets. Π-formation is interpreted as

$$A \in \mathcal{S}et \supset (\forall x \in \mathcal{E}l[A].B(x) \in \mathcal{S}et) \supset \Pi(A,B) \in \mathcal{S}et.$$

which can be shown by inspecting the rule set for $\mathcal{E}l$.

Π-introduction is interpreted as

$$(\forall x \in \mathcal{A}.b(x) \in \mathcal{B}[x]) \supset \lambda(b) \in \prod[\mathcal{A},\mathcal{B}]$$

which can be shown to hold by inspecting the rule set for $\prod[\mathcal{A},\mathcal{B}]$.

The first universe. U_0-formation is interpreted as

$$U_0 \in \mathcal{S}et,$$

which can be shown by inspecting the rule set for $\mathcal{E}l$.

Since $\mathcal{E}l[U_0] = \mathcal{U}_0$, T_0-typing is interpreted as

$$a \in \mathcal{U}_0 \supset T_0(a) \in \mathcal{S}et.$$

This is proved by proving the stronger property that $a \in \mathcal{U}_0 \supset T_0(a) \in \mathcal{S}et_3$ by induction on $\mathcal{U}_0$. For example, we need to show that $\pi_0(a,b) \in \mathcal{U}_0 \supset T_0(\pi_0(a,b)) \in \mathcal{S}et_3$ from $a \in \mathcal{U}_0 \supset T_0(a) \in \mathcal{S}et_3$ and $b(x) \in \mathcal{U}_0 \supset T_0(b(x)) \in \mathcal{S}et_3$ for all $x \in \mathcal{E}l_3[T_0(a)]$. But this follows from the facts that $T_0(\pi_0(a,b)) = \Pi(T_0(a),(x)T_0(b(x)))$ and that $\mathcal{S}et_3$ is closed under Π.

The rules of U_0-introduction are interpreted as

$$n_0 \in \mathcal{U}_0,$$

$$n \in \mathcal{N} \supset N'(n) \in \mathcal{U}_0,$$

$$a \in \mathcal{U}_0 \supset (\forall x \in \mathcal{E}l[T_0(a)].b(x) \in \mathcal{U}_0) \supset \pi_0(a,b) \in \mathcal{U}_0.$$

They follow directly by inspecting the rule set defining $\mathcal{U}_0$ using that $\mathcal{E}l[T_0(a)] = \mathcal{E}l_3[T_0(a)]$.

The equality rules for T_0 follow directly, since $T_0 \in \mathcal{F}_0$ was constructed to satisfy the corresponding untyped equalities. Moreover, we easily check that the typings in the equality rules are satisfied.

7 Concluding remarks

The formulation of simultaneous inductive-recursive definitions is obtained by a minor syntactic modification of the schema in Dybjer [20]. This adds evidence to the fundamental nature of the schematic natural deduction formulation of inductive definitions in type theory.

The idea to consider this generalisation was inspired by Nax Mendler's paper [32] on the category-theoretic semantics of universes in type theory. Our analysis improves fundamentally on Mendler's, since the category-theoretic machinery can be applied only if the rules for U_0 and T_0 already have been represented as an endofunctor on a category of families of sets. This representation is not itself analysed and also loses the U_0-recursive nature of T_0. It is not clear how to use category-theoretic ideas for obtaining a formal system for simultaneous induction-recursion.

The idea to enrich Frege structures with proof objects can also be found in Sato [40]. However, Sato works in a type-free constructive theory and not in type theory. Working in the same framework as Sato, Kameyama [25] has developed an approach to half-positive inductive definitions. His aims are similar to ours: to formulate a general notion which subsumes the construction of Frege structures and enables the interpretation of Martin-Löf type theory. In his type-free context the distinction between inductively and recursively defined sets (that our approach is based on) does not exist. Instead he considers simultaneous inductive definitions which give rise to operators which are monotone in the sense of an ordering which generalises Aczel's ordering for Frege structures described above.

As further examples of simultaneous induction-recursion, we would like to mention in particular the computability predicates used by Martin-Löf [31, 27] and C. Coquand [12] for proving normalisation of type theory, and the logical relations introduced by T. Coquand [13] for proving soundness and completeness of an algorithm for testing conversion in type theory. These constructions are similar in nature to the collections of propositions and truths in a Frege structure, and can be given a classical explanation in an analogous way. By defining constructions of these kinds by simultaneous induction-recursion we have paved the way for "internal type theory" [21], that is, to locally reflect the metatheory of type theory in itself. In particular, we hope to extend the technique of reduction-free normalization [15, 11, 17] developed for the simply typed case to dependent types.

References

[1] P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland, 1977.

[2] P. Aczel. The strength of Martin-Löf's type theory with one universe. In S. Miettinen and J. Väänanen, editors, *Proceedings of the Symposium on Mathematical Logic (Oulu 1974)*, pages 1–32, 1977. Report No 2 of Dept. Philosophy, University of Helsinki.

[3] P. Aczel. The type theoretic interpretation of constructive set theory. In A. MacIntyre, L. Pacholski, and J. Paris, editors, *Logic Colloquium '77*, pages 55–66. North-Holland, 1978.

[4] P. Aczel. *Frege Structures and the Notions of Proposition, Truth, and Set*, pages 31–59. North-Holland, 1980.

[5] S. Allen. *A Non-Type-Theoretic Semantics for Type-Theoretic Language.* PhD thesis, Department of Computer Science, Cornell University, 1987.

[6] T. Altenkirch. *Constructions, Inductive Types and Strong Normalization.* PhD thesis, The University of Edinburgh, Department of Computer Science, November 1993.

[7] R. Backhouse. On the meaning and construction of the rules in Martin-Löf's theory of types. In *Proceedings of the Workshop on General Logic, Edinburgh, February 1987.* Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1988. ECS-LFCS-88-52.

[8] H. P. Barendregt. *The Lambda Calculus.* North-Holland, 1984. Revised edition.

[9] R. Bird and P. Wadler. *Introduction to Functional Programming.* Prentice Hall, 1988.

[10] E. Bishop. *Foundations of Constructive Analysis.* McGraw-Hill, 1967.

[11] C. Coquand. From semantics to rules: a machine assisted analysis. In E. Börger, Y. Gurevich, and K. Meinke, editors, *Proceedings of CSL '93, LNCS 832*, 1993.

[12] C. Coquand. A realizability interpretation of Martin-Löf's type theory. In G. Sambin and J. Smith, editors, *Twenty-Five Years of Constructive Type Theory.* Oxford University Press, 1998. To appear.

[13] T. Coquand. An algorithm for testing conversion in type theory. In *Logical Frameworks*, pages 255–279. Cambridge University Press, 1991.

[14] T. Coquand. Pattern matching with dependent types. In *Proceedings of The 1992 Workshop on Types for Proofs and Programs*, June 1992.

[15] T. Coquand and P. Dybjer. Intuitionistic model constructions and normalization proofs. *Mathematical Structures in Computer Science*, 7:75–94, 1997.

[16] T. Coquand and C. Paulin. Inductively defined types, preliminary version. In *LNCS 417, COLOG '88, International Conference on Computer Logic.* Springer-Verlag, 1990.

[17] D. Čubrić, P. Dybjer, and P. Scott. Normalization and the Yoneda embedding. *Mathematical Structures in Computer Science*, 1998. To appear.

[18] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Paulin, and B. Werner. The Coq proof assistant version 5.6, user's guide. Technical report, INRIA Rocquencourt - CNRS ENS Lyon, 1991.

[19] P. Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In *Logical Frameworks*, pages 280–306. Cambridge University Press, 1991.

[20] P. Dybjer. Inductive families. *Formal Aspects of Computing*, pages 440–465, 1994.

[21] P. Dybjer. Internal type theory. In *TYPES '95, Types for Proofs and Programs*, number 1158 in Lecture Notes in Computer Science, pages 120–134. Springer, 1996.

[22] E. Giménez. A command for inductive sets in ILF. Master Thesis, Universidad de la República, Montevideo, 1992.

[23] E. Griffor and M. Rathjen. The strength of some Martin-Löf type theories. *Archive of Mathematical Logic*, 33:337–385, 1994.

[24] M. Hedberg. *Type Theory and the External Logic of Programs.* PhD thesis, Chalmers University of Technology and University of Göteborg, 1994.

[25] Y. Kameyama. A type-free theory of half-monotone inductive definitions. *International Journal of Foundations of Computer Science*, 6(3):203–234, 1995.

[26] P. Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 179–216. North-Holland, 1971.

[27] P. Martin-Löf. An intuitionistic theory of types: Predicative part. In *Logic Colloquium '73*, pages 73–118. North-Holland, 1975.

[28] P. Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science, VI, 1979*, pages 153–175. North-Holland, 1982.

[29] P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, 1984.

[30] P. Martin-Löf. Amendment to intuitionistic type theory. Notes from a lecture given in Göteborg, March 1986.

[31] P. Martin-Löf. An intuitionistic theory of types. In G. Sambin and J. Smith, editors, *Twenty-Five Years of Constructive Type Theory.* Oxford University Press, 1998. To appear. Reprinted version of an unpublished report from 1972.

[32] P. F. Mendler. Predicative type universes and primitive recursion. In *Proceedings Sixth Annual Synposium on Logic in Computer Science.* IEEE Computer Society Press, 1991.

[33] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML.* MIT Press, 1990.

[34] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: an Introduction.* Oxford University Press, 1990.

[35] E. Palmgren. *On Fixed Point Operators, Inductive Definitions and Universes in Martin-Löf's Type Theory.* PhD thesis, Uppsala University, 1991.

[36] E. Palmgren. On universes in type theory. In G. Sambin and J. Smith, editors, *Twenty-Five Years of Constructive Type Theory.* Oxford University Press, 1998. Preprint. To appear.

[37] M. Parigot. Programming with proofs: a second order type theory. In H. Ganzinger, editor, *ESOP'88, 2nd European Symposium on Programming, Nancy, LNCS 300*, pages 145–159, March 1988.

[38] C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *Proceedings Typed λ-Calculus and Applications*, pages 328–245. Springer-Verlag, LNCS, March 1993.

[39] M. Rathjen, E. R. Griffor, and E. Palmgren. Inaccessibility in constructive set theory and type theory. *Annals of Pure and Applied Logic*, 1998. To appear.

[40] M. Sato. Adding proof objects and inductive definition mechanism to Frege structures. In T. Ito and A. Meyer, editors, *Proc. International Conference on Theoretical Aspects of Computer Science*, number 526 in LNCS, pages 53–87. Springer Verlag, 1991.

[41] A. Setzer. *Proof theoretical strength of Martin-Löf Type Theory with W-type and one universe.* PhD thesis, Fakultät für Mathematik der Ludwig-Maximilians-Universität München, 1993.

[42] A. Setzer. Extending Martin-Löf Type Theory by one Mahlo-universe. Preprint, 1996.

[43] J. Smith. The independence of Peano's fourth axiom from Martin-Löf's type theory without universes. *Journal of Symbolic Logic*, 49(3), 1988.

[44] J. Smith. Propositional functions and families of types. *Notre Dame Journal of Formal Logic*, 30(3):442–458, 1989.

[45] B. Werner. A normalization proof for an impredicative type system with large elimination over integers. In B. Nordström, K. Petersson, and G. Plotkin, editors, *Proceedings of the 1992 Workshop on Proofs and Programs*, pages 361–377. Department of Computer Sciences, Chalmers University of Technology, June 1992.