Indexed Induction-Recursion

Peter Dybjer*and Anton Setzer†

September 24, 2001

Abstract

We give two finite axiomatizations of indexed inductive-recursive definitions in intuitionistic type theory. They extend our previous finite axiomatizations of inductive-recursive definitions of sets to *indexed families* of sets and encompass virtually all definitions of sets which have been used in intuitionistic type theory. The more restricted of the two axiomatization arises naturally by considering indexed inductive-recursive definitions as initial algebras in slice categories, whereas the other admits a more general and convenient form of an introduction rule.

The class of indexed inductive-recursive definitions properly contains the class of indexed inductive definitions (so called "inductive families"). Such definitions are ubiquitous when using intuitionistic type theory for formalizing mathematics and program correctness. A side effect of the present paper is to get compact finite axiomatizations of indexed inductive definitions in intuitionistic type theory as special cases.

Proper indexed inductive-recursive definitions (those which do not correspond to indexed inductive definitions) are useful in intuitionistic metamathematics, and as an example we formalize Tait-style computability predicates for dependent types. We also show that Palmgren's proof-theoretically strong construction of higher-order universes is an example of a proper indexed inductive-recursive definition. A third interesting example is provided by Bove and Capretta's definition of termination predicates for functions defined by nested recursion.

Our axiomatizations form a powerful foundation for generic programming with dependent types by introducing a type of codes for indexed inductive-recursive definitions and making it possible to define generic functions by recursion on this type.

Keywords: Dependent type theory, Martin-Löf Type Theory, inductive definitions, inductive-recursive definitions, inductive families, initial algebras, normalization proofs, generic programming.

*Department of Mathematics and Computing Science, Chalmers University of Technology. Email: peterd@cs.chalmers.se, Fax: +46 31 165655.

†Supported by Nuffield Foundation, grant No. NAL/00303/G; Department of Computer Science, University of Wales Swansea, Singleton Park, Swansea SA2 8PP, UK. Email: a.g.setzer@swan.ac.uk, Fax: +44 1792 295651.

1 Introduction

In [8], the first author introduced the concept of inductive-recursive definitions, an extension of ordinary inductive definitions. By inductive-recursive definitions one can inductively define a set U while simultaneousely recursively defining a function $T : U \to D$, where D is an arbitrary type.

The prime example of an inductive-recursive definition is a universe, that is, a set U of codes for sets together with a decoding function $T : U \to \text{set}$, which assigns to a code in U the set it denotes. As an example consider the constructor $\widehat{\Sigma}$ introducing codes for the Σ-type. We have $\widehat{\Sigma} : (a : U, b : T(a) \to U) \to U$ (the reader not familiar with this notation, which will be explained in Sec. 2, might temporarily substitute $\Pi a : U.\Pi b : (T(a) \to U).U$ for the type of $\widehat{\Sigma}$). Note that the second argument b of $\widehat{\Sigma}(a, b)$ refers to $T(a)$. This makes sense since before $\widehat{\Sigma}(a, b)$ is introduced, $a : U$ has to be established, and when $a : U$ is established, one can recursively define $T(a)$. We then define recursively $T(\widehat{\Sigma}(a, b)) = \Sigma(T(a), (x)T(b(x)))$ (or in more traditional notation $T(\widehat{\Sigma}(a, b)) = \Sigma x : T(a).T(b(x))$). This is possible, since $T(a)$ and $T(b(x))$ are known before we introduce $\widehat{\Sigma}(a, b)$. Here we see the novelty of inductive-recursive definitions relative to inductive definitions: an introduction rule for U may refer to the function T which is simultaneously defined. Note that T even can appear negatively in the type of a constructor, as in the type of $\widehat{\Sigma}$. However, the constructor refers only strictly positively to elements of the inductively defined set U.

In the special case where $T : U \to \mathbf{1}$, where $\mathbf{1}$ is the type with only one element, so that T does not contain any information, inductive-recursive definitions specialize to inductive definitions. For a detailed explanation of the concept of inductive-recursive definitions, the reader is advised to read the aforementioned article [8].

In [8] the case of *indexed* inductive-recursive definitions (**IIR**) is also considered. In an IIR one defines a *family* of sets $U(i)$, indexed over $i : I$, inductively, while simultaneously recursively defining a family of functions $T(i) : U(i) \to D[i]$ for some collection of types $D[i]$ $(i : I)$. (We have to write $D[i]$, the result of substituting i for a fixed variable in D here, since $D[i]$ is a type and we cannot write $D : I \to \text{type}$.) Again constructors of $U(i)$ can refer to $T(j)$ applied to elements of $U(j)$.

In [9] and [10] we presented closed axiomatizations of the theory of non-indexed inductive-recursive definitions. It is the objective of this article to extend this to indexed inductive-recursive definitions. We will see that the resulting rules are not much more complicated than the rules for the non-indexed case. We will look at two alternatives: In **restricted IIR** (introduced by T. Coquand for use in the Half and Agda [4] systems) we can determine for each index i the set of arguments of the constructors introducing elements of the set U_i. In **unrestricted IIR** [8], for a constructor C with arguments $\vec{a}$, the index i s.t. $C(\vec{a}) : \mathrm{U}_i$ depends on the arguments $\vec{a}$ of C.

Indexed inductive definitions **(IID)** subsume inductively defined relations, such as the identity relation understood as the least reflexive relation. The identity relation was indeed the only example of an IID in the early versions of Martin-Löf type theory [13]. Theoretically, this is not very limiting since one can define many other families of sets (predicates) using the identity together with

the other set formers of type theory. But from a more practical point of view it became desirable to extend Martin-Löf type theory with a general notion of inductive definition of an indexed family of sets (often called "inductive family" for short) [6, 7]. Such a general mechanism is also a key part of the Calculus of Inductive Constructions [21], the impredicative type theory underlying the Coq system. Indexed inductive definitions are ubiquitous in formalizations carried out in the different proof systems for Martin-Löf type theory and the Calculus of Inductive Constructions.

No proper IIR were part of the original versions of Martin-Löf type theory. However, when proving normalization for an early version of this theory, Martin-Löf made use of an informal construction on the metalevel which has an implicit indexed inductive-recursive character.

Plan. In Section 2 we introduce the logical framework of dependent types which is the basis for our theories of IIR. This section also explains the notation used in the paper. In Section 3 we begin by reviewing a few examples of IID, such as finitely branching trees, the even and odd numbers, the accessible part of a relation, the identity set, and context free grammars. Then we give some examples of proper IIR: Martin-Löf's computability predicates for dependent types, Palmgren's higher order universes, and Bove and Capretta's analysis of the termination of nested recursive definitions of functional programs. We also introduce the restricted form of an inductive-recursive definition which has an easier syntax and is easier to implement. In Section 4 we first explain how the restricted version of indexed inductive-recursive definitions arises from the existence of initial algebras of strictly positive endofunctors on slice categories over the category of indexed families of types. Then we show how to formalize the general notion of an indexed inductive-recursive definition and give a uniform theory for both the general and the restricted form by giving a type of codes for IIR and then derive the formation, introduction, elimination and equality rules for a particular code. In Section 5 we show how to instantiate the general theory to some of the examples given in Section 3.

2 The Logical Framework

Before giving the rules for IIR we need to introduce the basic Logical Framework of dependent types. This is essentially Martin-Löf's Logical Framework [18] extended with rules for the types $\mathbf{0}$, $\mathbf{1}$, and $\mathbf{2}$. A more detailed presentation can be found in [10].

The Logcial Framework has the following forms of judgements: Γ context, and $A : \text{type}$, $A = B : \text{type}$, $a : A$, $a = b : A$, depending on contexts Γ (written as $\Gamma \Rightarrow A : \text{type}$, etc.). We have $\text{set} : \text{type}$ and if $A : \text{set}$, then $A : \text{type}$. The collection of types is closed under the formation of dependent function types written as $(x : A) \to B$ (which is often written as $\Pi x : A.B$ in the literature – we prefer to reserve $\Pi x : A.B$ for the inductive-recursively defined set with constructor $\lambda : ((x : A) \to B) \to (\Pi x : A.B)$). The elements of $(x : A) \to B$ are denoted by $(x : A)a$ (abstraction of x in a; this is often denoted by $\lambda x : A.a$ in the literature) and application is written in the form $a(b)$. We have $\beta-$ and η-rules. Types are also closed under the formation of dependent products written as $(x : A) \times B$ (often denoted by $\Sigma x : A.B$ which is here reserved for the

inductively defined set with introduction rule $\mathrm{p} : ((x : A) \times B) \to (\Sigma x : A.B)$). The elements of $(x : A) \times B$ are denoted by $\langle a, b \rangle$, the projection functions by π_0 and π_1 and again we have β and η-rule (surjective pairing). There is the type $\mathbf{1}$, with unique element $\star : \mathbf{1}$ and η-rule expressing that, if $a : \mathbf{1}$, then $a = \star : \mathbf{1}$. Further we have the empty type $\mathbf{0}$ with elimination rule Efq (ex falsum quodlibet).

Moreover, we include in our logical framework the type $\mathbf{2}$ with two elements $\star_0 : \mathbf{2}$ and $\star_1 : \mathbf{2}$, ordinary elimination rule $\mathrm{C}_2 : (a : \mathbf{2}, A[\star_0], A[\star_1]) \to A(a)$ (where $x : \mathbf{2} \Rightarrow A[x] : \text{type}$) and the strong elimination rule

$$\frac{a : \mathbf{2} \qquad A : \text{type} \qquad B : \text{type}}{\mathrm{C}_2^{\text{type}}(a, A, B) : \text{type}}$$

with equality rules

$$\mathrm{C}_2^{\text{type}}(\star_0, A, B) = A \ , \qquad \mathrm{C}_2^{\text{type}}(\star_1, A, B) = B \ .$$

It is necessary to have the strong elimination rule, since we want to inductively define indexed families of sets $U : I \to \text{set}$ and functions $T : (i : I) \to U(i) \to D[i]$ where $D[i]$ depends non-trivially on i, as in the definition of Palmgren's higher-order universe (where for instance $D[0] = \text{set}$, $D[1] = \text{Fam(set)} \to \text{Fam(set)}$, see 3.2 for more explanation).

We will also add a level between set and type, which we call stype for small types: stype : type. (The reason for the need for stype is discussed in [8].) If $a : \text{set}$ then $a : \text{stype}$. Moreover, stype is also closed under dependent function types, dependent products and includes $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$. However, set itself will not be in stype. The logical framework does not have any rules for introducing elements of set, they will be introduced by IIR later and set will therefore consist exactly of the sets introduced by IIR.

We also use some abbreviations, such as omitting the type in an abstraction, that is, writing $(x)a$ instead of $(x : A)a$, and writing repeated application as $a(b_1, \ldots, b_n)$ instead of $a(b_1) \cdots (b_n)$ and repeated abstraction as $(x_1 : A_1, \ldots, x_n : A_n)a$ instead of $(x_1 : A_1) \cdots (x_n : A_n)a$.

In the following we will sometimes refer to a type depending on a variable x. We want to use the notation $D[t]$ for $D[x := t]$ for some fixed variable x and D for $(x)D[x]$. Note that we can't simply introduce $D : I \to \text{type}$, since this goes beyond the logical framework. Instead we introduce the notion of an abstracted expression, which is an expression together with one or several designated free variables. For an abstracted expression E, $E[t_1, \ldots, t_n]$ means the substitution of the variables by $t_1, \ldots, t_n$. If we take for D above an abstracted expression of the form $(x)E$, then $D[t]$ denotes $D[x := t]$ and we can write D as parameter for $(x)E$. More formally:

Definition 1 *(a) An n-times abstracted expression is an expression $(x_1, \ldots, x_n)E$ where $x_1, \ldots, x_n$ are distinct variables and E an expression of the language of type theory. An abstracted expression is a 1-times abstracted expression.*

(b) $((x_1, \ldots, x_n)E)[t_1, \ldots, t_n] := E[x_1 := t_1, \ldots, x_n := t_n]$.

(c) Whenever we write $s[a_1, \ldots, a_n]$, s is to be understood as an n-times abstracted expression.

(d) If $U : A \to B$, we identify U with the abstracted expression $(a)U(a)$.

3 Some Examples

3.1 Indexed Inductive Definitions

Trees and forests. Many IID occur as the simultaneous inductive definition of finitely many sets, each of which has a different name. One example is the set of well-founded trees Tree with finite branching degrees, which is defined together with the set Forest of finite lists of such trees. The constructors are:

$$\begin{aligned} \text{tree} &: \text{Forest} \to \text{Tree} \ , \\ \text{nil} &: \text{Forest} \ , \\ \text{cons} &: \text{Tree} \to \text{Forest} \to \text{Forest} \ . \end{aligned}$$

If we replace Tree by $\text{Tree}'(\star_0)$ and Forest by $\text{Tree}'(\star_1)$, where $x : \mathbf{2} \Rightarrow \text{Tree}'(x) : \text{set}$, we obtain an IID with index set $\mathbf{2}$. Since for every index we know the set of constructors introducing elements of it in advance, we have an example of restricted IID.

The even number predicate. Another simple example of an IID is the predicate $\text{Even} : \text{N} \to \text{set}$. This is inductively generated by the two rules

$$\begin{aligned} \text{C}_0 &: \text{Even}(0) \ , \\ \text{C}_1 &: (n : \text{N}) \to \text{Even}(n) \to \text{Even}(\text{S}(\text{S}(n))) \ . \end{aligned}$$

In this form we have unrestricted IID, since in this form the constructors introducing an element of $\text{Even}(n)$ is not given in advance.

The accessible part of a relation. Let I be a set and $<: I \to I \to \text{set}$ be a binary relation on it. We define the accessible part (the largest well-founded initial segment) of $<$ as a predicate $\text{Acc} : I \to \text{set}$ by a generalized indexed inductive definition with one introduction rule:

$$\text{acc} : (i : I) \to ((x : I) \to (x < i) \to \text{Acc}(x)) \to \text{Acc}(i) \ .$$

Note that acc introduces elements of $\text{Acc}(i)$ while referring to possibly infinitely many elements of the sets $\text{Acc}(x)$ (for each $x : I$ and each proof of $x < i$). Acc is an example of a restricted IID.

The identity relation. The only example of an IID in Martin-Löf's original formulation of type theory, which is not an ordinary inductive-recursive definition is the identity type ("identity type" is only used for historic reason – a more appropriate name would be "identity set"). Assume $A : \text{set}$. The identity on A is given as the least reflexive relation on $A \times A$, and is the intensional equality type on A. We have as formation rule $a : A, b : A \Rightarrow \text{I}(A, a, b) : \text{set}$. The introduction rule expresses that it is reflexive:

$$\text{r} : (a : A) \to \text{I}(A, a, a) \ .$$

The elimination rule inverts the introduction rule. Assume we have a type which is a reflexive relation on $A \times A$ and a subrelation of the identity type on A. So assume $a : A, b : A, p : \text{I}(A, a, b) \Rightarrow C[a, b, p] : \text{type}$ and for every $a : A$ we have

$s[a] : C[a, a, \mathrm{r}(a)]$ (so s is the step-function corresponding to the constructor r). Then for every $a, b : A$ and $p : \mathrm{I}(A, a, b)$ we have $\mathrm{J}(a, b, p, (x)s[x]) : C[a, b, p]$. The equality rule now uses the step function in case an element is introduced by a constructor: $\mathrm{J}(a, a, \mathrm{r}(a), (x)s[x]) = s[a]$.

If we write $\mathrm{I}'_A(\langle a, b\rangle)$ instead of $\mathrm{I}(A, a, b)$, we obtain for every A : set an unrestricted IID I'_A with index set $A \times A$.

Context free grammars. IID occur very frequently in applications in computer science. For example a context free grammars over a finite alphabet Σ and a finite set of nonterminals NT can be seen as an $\mathrm{NT} \times \Sigma^*$-indexed inductive definition L, where each production corresponds to an introduction rule.

As an example consider the context free grammar with $\Sigma = \{a, b\}$, $\mathrm{NT} = \{A, B\}$ and productions $A \longrightarrow a$, $A \longrightarrow BB$, $B \longrightarrow AA$, $B \longrightarrow b$. This corresponds to an inductive definition of an indexed family L, where $L(A, \alpha)$ is the set of derivation trees of the string α from the start symbol A. So α is in the language generated by the grammar with start symbol A iff $L(A, \alpha)$ is inhabited. L has one constructor for each production: $C_0 : L(A, a)$, $C_1 : L(B, \alpha) \to L(B, \beta) \to L(A, \alpha\beta)$, $C_2 : L(A, \alpha) \to L(A, \beta) \to L(B, \alpha\beta)$, $C_3 : L(B, b)$.

Alternatively, we can inductively define an NT-indexed set D of "abstract syntax trees" for the grammar, and then recursively define the string $d(A, p)$ ("concrete syntax") corresponding to the abstract syntax tree $p : D(A)$. In the example given before we get $C_0 : D(A)$, $C_1 : D(B) \to D(B) \to D(A)$, $C_2 : D(A) \to D(A) \to D(B)$, $C_3 : D(B)$. Further $d(A, C_0) = a$, $d(A, C_1(p, q)) = d(B, p) * d(B, q)$, $d(B, C_2(p, q)) = d(A, p) * d(B, q)$, $d(B, C_3) = b$.

More examples. There are many more examples (eg. the set of formulas derivable in a formal system, computation rules of the operational semantics of a programming language) of similar nature. If Formula is a set of formulas of the formal system, then to be a theorem is given by a Formula-indexed inductive definition Theorem : Formula $\to$ set, where the axioms and inference rules correspond to introduction rules. An element $d : \mathrm{Theorem}(\phi)$ is a notation for a derivation (or proof tree) with conclusion ϕ. Yet more examples are provided by the computation rules in the definition of the *operational semantics* of a programming language.

Proofs by induction on the structure of an indexed inductive definition of these kinds are often called proofs by "rule induction". Thus the general form of rule induction is captured by the elimination rule for unrestricted IIR which we will give later.

3.2 Indexed Inductive-Recursive Definitions

Martin-Löf's computability predicates for dependent types. We shall now turn to proper IIR. As a first example we shall formalize the Tait-style computability predicates for dependent types introduced by Martin-Löf [17]. This example was crucial for the historical development of IIR, since it may be viewed as an early occurrence of the informal notion of an IIR. In [17] Martin-Löf presents an early version of his intuitionistic type theory and proves a normalization theorem using such Tait-style computability predicates. He works in an informal intuitionistic metalanguage but gives no explicit justification for the

meaningfulness of these computability predicates. (Later Aczel [1] has shown how to model a similar construction in classical set theory.) Since the meta-language is informal the inductive-recursive nature of this definition is implicit. One of the objectives of the current work is indeed to formalize an extension of Martin-Löf type theory where the inductive-recursive nature of this and other definitions is formalized. In this way we hope to help clarify the reason why it is an acceptable notion from the point of view of intuitionistic meaning explanations in the sense of Martin-Löf [14, 16, 15].

First recall that for the case of the simply typed lambda calculus the Tait-computability predicates ϕ_A are predicates on terms of type A which are defined by *recursion* on the structure of A. We read $\phi_A(a)$ as "a is a computable term of type A". To match Martin-Löf's definition [17] we consider here a version where the clause for function types is

- If $\phi_B(b[a])$ for all closed terms a such that $\phi_A(a)$ then $\phi_{A\to B}(\lambda x.b[x])$.

(Here $b[x]$ denotes an expression with a possible occurrence of the free variable x and $b[a]$ denotes the expression which is obtained by substituting x by a in $b[x]$.)

How can we generalize this to dependent types? First we must assume that we have introduced the syntax of expressions for dependent types including Π-types, with lambda abstraction and application. Now we cannot define ϕ_A for all (type) expressions A but only for those which are "computable types". The definition of ϕ_A has several clauses, such as the following one for Π [17, p. 161]:

> 4.1.1.2. Suppose that ϕ_A has been defined and that $\phi_{B[a]}$ has been defined for all closed terms a of type A such that $\phi_A(a)$. We then define $\phi_{\Pi x:A.B[x]}$ by the following three clauses.
>
> 4.1.1.2.1. If $\lambda x.b[x]$ is a closed term of type $\Pi x : A.B[x]$ and $\phi_{B[a]}(b[a])$ for all closed terms a of type A such that $\phi_A(a)$, then $\phi_{\Pi x:A.B[x]}(\lambda x.b[x])$.
>
> 4.1.1.2.2. ...
>
> 4.1.1.2.3. ...

(We omit the cases 4.1.1.2.2 and 4.1.1.2.3, which express closure under reduction, since they are not relevant for the present discussion. Note also that the complete definition of the computability predicate also has one case for each of the other type formers of type theory.)

We also note that Martin-Löf does not use the term "A is a computable type" but only states "that ϕ_A has been defined". We can understand Martin-Löf's definition as an indexed inductive-recursive definition by introducing a predicate Φ on expressions, where $\Phi(A)$ stands for "ϕ_A is defined" or "A is a *computable type*". Moreover, we add a second argument to ϕ so that $\phi_A(p, a)$ means that "a is a computable term of the computable type A, where p is a proof that A is computable. Now we observe that we define Φ inductively while we simultaneously recursively define ϕ.

It would be possible to formalize Martin-Löf's definition verbatim, but for simplicity we shall follow a slightly different version due to C. Coquand [5]. Assume that we have inductively defined the set Exp of expressions and have an operation Apl : Exp $\to$ Exp $\to$ Exp for the application of one expression to another. Apl is a constructor of Exp, and there will be additional reduction

rules for expressions, like reduction of β-redexes. We will write in the following $A\ b$ for $\mathrm{Apl}(A, b)$.

Now define an Exp-indexed IIR

$$\begin{array}{lcl} \Psi & : & \mathrm{Exp} \to \mathrm{set}\ , \\ \psi & : & (A : \mathrm{Exp}) \to \Psi(A) \to \mathrm{Exp} \to \mathrm{set}\ . \end{array}$$

So the index set is Exp, Ψ plays the rôle of U, ψ the rôle of $T : (a : U) \to D[a]$, where $D[a] = \mathrm{Exp} \to \mathrm{set}$ for $a : \mathrm{Exp}$. Note that ψ depends negatively on Ψ, so this is not a simultaneous inductive definition. The introduction rule for Ψ (corresponding to Martin-Löf's 4.1.1.2) is:

$$\begin{array}{lcl} \pi & : & (A : \mathrm{Exp}) \to (p : \Psi(A)) \to (B : \mathrm{Exp}) \to \\ & & (q : (a : \mathrm{Exp}) \to \psi(A, p, a) \to \Psi(B\,a)) \to \\ & & \Psi(\Pi(A, B))\ . \end{array}$$

Note that q refers to $\psi(A, p, a)$ which is short notation for $\psi(A, p)(a)$, where $\psi(A, p)$ is the result of the recursively defined function for the second argument p. The corresponding equality rule is

$$\psi(\Pi(A, B), \pi(A, p, B, q), b) = \forall a : \mathrm{Exp}.\forall x : \psi(A, p, a).\psi(B\,a, q(a, x), b\,a)\ .$$

Again, the reader should be aware that we have presented only one crucial case of the complete IIR in [5]. For instance there are clauses corresponding to closure under reductions.

Palmgren's higher-order universes [20]. This construction generalizes Palmgren's super universe [19], that is, a universe which for any family of sets in it contains a universe containing this family.

It is outside the scope of this paper to give a full explanation of higher-order universes, and the interested reader is referred to Palmgren [20]. They are included here as an example of a proof-theoretically strong construction which is subsumed by our theory of IIR: they are conjectured to reach (without elimination rules into arbitrary types) the strength of Kripke-Platek set theory with one recursive Mahlo ordinal. They also provide an example where we have decoding functions $\mathrm{T}_k : \mathrm{U}_k \to D[k]$ with $D[k] = \mathrm{OP}^k(\mathrm{set})$ which depend non-trivially on k and, for $k > 0$, is a higher type which goes beyond set.

The higher-order universes of level n is a family of universes $\mathrm{U}_k, \mathrm{T}_k$, indexed by $k < n$, where U_k is a set of codes for operators on families of sets of level k, and $\mathrm{T}_k : \mathrm{U}_k \to \mathrm{OP}^k(\mathrm{set})$ is the decoding function. The set $\mathrm{OP}^k(\mathrm{set})$ of such operators is defined by introducing more generally for $A : \mathrm{type}$

$$\begin{array}{lcl} \mathrm{Fam}(A) & := & (X : \mathrm{set}) \times (X \to A)\ , \qquad \mathrm{Op}(A) := \mathrm{Fam}(A) \to \mathrm{Fam}(A)\ , \\ \mathrm{Op}^n(A) & := & \underbrace{\mathrm{Op}(\cdots(\mathrm{Op}}_{n\ \text{times}}(A))\cdots)\ , \end{array}$$

Let

$$n : \mathrm{N}, \qquad A_k : \mathrm{set}\ , \qquad B_k : A_k \to \mathrm{Op}^k(\mathrm{set}) \quad (k = 0, \ldots, n)\ .$$

In the following all sets and constructors are parameterized with respect to n, A_k, B_k, but for simplicity we omit those parameters.

U_0, T_0 have the standard closure properties of a universe. Additionally we have for $k = 0, \ldots, n$ the two constructors (with decodings)

$$\begin{array}{ll} \widehat{\mathrm{A}}_k : \mathrm{U}_0 \ , & \mathrm{T}_0(\widehat{\mathrm{A}}_k) = A_k \ , \\ \widehat{\mathrm{B}}_k : A_k \to \mathrm{U}_k \ , & \mathrm{T}_k(\widehat{\mathrm{B}}_k(a)) = B_k(a) \ . \end{array}$$

Furthermore, under the additional assumptions $i \in \{0, \ldots, n-1\}$, $f : \mathrm{U}_{i+1}$, we have two more constructors (with decodings)

$$\begin{array}{l} \mathrm{ap}_i^0(f) : (u : \mathrm{U}_0, \mathrm{T}_0(u) \to \mathrm{U}_i) \to \mathrm{U}_0 \ , \\ \quad \mathrm{T}_0(\mathrm{ap}_i^0(f, u, v)) = \pi_0(\mathrm{T}_{i+1}(f)(\langle \mathrm{T}_0(u), \mathrm{T}_i \circ v \rangle)) \ , \\ \mathrm{ap}_i^1(f) : (u : \mathrm{U}_0, v : \mathrm{T}_0(u) \to \mathrm{U}_i, \pi_0(\mathrm{T}_{i+1}(f)(\langle \mathrm{T}_0(u), \mathrm{T}_i \circ v \rangle))) \to \mathrm{U}_i \ , \\ \quad \mathrm{T}_i(\mathrm{ap}_i^1(f, u, v, a)) = \pi_1(\mathrm{T}_{i+1}(f)(\langle \mathrm{T}_0(u), \mathrm{T}_i \circ v \rangle))(a) \ . \end{array}$$

If we let $I := \{0, \ldots, n\}$ (which more formally should be replaced by N_{n+1}), then we observe that we have introduced $\mathrm{U} : I \to \mathrm{set}$ together with $\mathrm{T} : (i : I, \mathrm{U}(i)) \to D_i$ where $D_i = \mathrm{Op}^i(\mathrm{set})$. For each $i : I$ we can determine a collection of constructors, each of which refers strictly positively to U and (positively and negatively) to T applied to the arguments of U referred to.

Bove and Capretta's analysis of the termination of nested recursive definitions of functional programs. In a recent paper [3], Bove and Capretta use indexed inductive-recursive definitions in their analysis of the termination of functions defined by nested general recursion. Given such a function f the idea is to simultaneously define a predicate $D(x)$ expressing that $f(x)$ terminates, and a function $f'(x, p)$ which returns the same value as $f(x)$ but has as second argument a proof $p : D(x)$ that $f(x)$ terminates.

Assume for instance the rewrite rules $f(0) \longrightarrow f(f(1))$, $f(1) \longrightarrow 2$, $f(2) \longrightarrow f(1)$ on the domain $\{0, 1, 2\}$. We now inductive-recursively define the termination predicate D for for f. We get one constructor for each rewrite rule: $C_0 : (p : D(1), q : D(f'(1, p))) \to D(0)$, $C_1 : D(1)$, $C_2 : (p : D(1)) \to D(2)$. Furthermore, the equality rules for f' are $f'(0, C_0(p, q)) = f'(f'(1, p), q)$, $f'(1, C_1) = 2$, $f'(2, C_2(p)) = f'(1, p)$. This is a proper IIR, since in the type of C_0 the second argument depends on $f'(1, p)$, where p is the first argument.

3.3 Restricted Indexed Inductive-Recursive Definitions

There are reasons for focussing attention on IIR where we can determine for every index i the set of constructors introducing elements of U_i. More precisely this means that if C is a constructor of the IIR, and we write its type in uncurried form, then its type is of the form $((i : I) \times A(i)) \to \mathrm{U}_i$, so the first argument determines the index i. An example which satisfies this restriction is the accessible part of a relation.

We can also determine the set of constructors for U_i, where a constructor has type $B \to \mathrm{U}_i$ and i does not depend on B, provided the equality on I is decidable. In this case the type of C can be replaced by $(i' : I) \to \mathrm{C}_2^{\mathrm{type}}(i =_{\mathrm{dec}} i', B, \mathbf{0}) \to \mathrm{U}_{i'}$, where $=_{\mathrm{dec}}$ is the decidable equality on I, with result in $\mathbf{2}$). In this way for example finitely branching trees and forests can be seen to be captured by restricted IIR.

In the implementation of the Half proof system of type theory Thierry Coquand enforced this restriction, and it was kept in the Agda system [4], the successor of Half. One reason is that both the introduction and elimination rules can be specified more simply: for introducing an IIR, the constructors C_i of U_j are just listed in the form data$\{C_1(\vec{a}_1) \mid \cdots \mid C_n(\vec{a}_n)\}$, and one doesn't have to include the index j in the arguments of $\vec{a}_i$ of $C_i(\vec{a}_i)$ – this simplifies the syntax of the system substantially. If one wants to define $g(x)$ for $x : U_i$, one can write it in the form

$$\begin{array}{rcl} \text{case } x \text{ of } \{C_1(\vec{a}_1) & \rightarrow & f_1(\vec{a}_n); \\ \cdots & & \\ C_n(\vec{a}_n) & \rightarrow & f_n(\vec{a}); \} \end{array}$$

For defining unrestricted IIR, a more complicated syntax has to be used, especially the elimination rules have to make use of a more general form of pattern matching. See for instance the proof assistant Alf ([12], [2]), where unrestricted IIR can be implemented.

Another reason for this restriction is that it is easier to construct mathematical models: below we will see that restricted IIR can be modelled as initial algebras in an I-indexed slice category. Furthermore, domain-theoretic models of restricted IIR can be given more easily. That complications arise when modelling unrestricted IIR is one of the reasons why many believe that a fully satisfactory understanding of the identity type has not yet been achieved.

It is often possible to replace general IIR by restricted IIR, especially if we have a decidable equality on the index set. For example, we can define a function by recursion which tests equality of natural numbers. Using this equality we can write the introduction rules for the even number predicate in an alternative way: Even(n) holds iff $n = 0$ or there exists m such that $n = \mathrm{S}(\mathrm{S}(m))$ and Even(m). That is, we have two constructors: $\mathrm{C}_0(n) : (n = 0) \rightarrow \mathrm{Even}(n)$, and $\mathrm{C}_1(n) : (m : \mathrm{N}) \times (n = \mathrm{S}(\mathrm{S}(m))) \times (\mathrm{Even}(m)) \rightarrow (\mathrm{Even}(n))$. (It is also worth mentioning here that alternatively, we can directly define the even numbers by primitive recursion on the natural numbers: Even(0) is true and Even(S(n)) is the negation of Even(n), using the two element universe $\mathbf{2}$ with decoding $T : \mathbf{2} \rightarrow \mathrm{set}$.)

However, it is not always possible to transform a definition into the restricted form. The prime example is that of the identity relation. In [11] we however show that if we have extensional equality, we can simulate general IIR by restricted IIR. The definitions are quite complicated though, and the resulting programs may be computationally inefficient.

4 Formalizing the Theory of Indexed Inductive-Recursive Definitions

4.1 The Category of Indexed Families of Types

As for the non-indexed case, we shall derive a formalization of IIR by modeling them as initial algebra constructions in slice categories. Let R be a set of rules for the language of type theory (where each rule is given by a finite set of judgements as premisses and one judgement as conclusion) which includes the

logical framework used in this article and an equality. For such R let $\mathrm{TT}(R)$ be the resulting type theory. The category $\mathbf{Type}(R)$ is the category, the objects of which are A s.t. $\mathrm{TT}(R)$ proves $A : \mathrm{type}$, and the morphisms from A to B of which are terms f s.t. $\mathrm{TT}(R)$ proves $f : A \to B$. Objects A, A' such that $\mathrm{TT}(R)$ proves $A = A' : \mathrm{type}$ and functions f, f' s.t. $\mathrm{TT}(R)$ proves $f = f' : B \to C$ are identified. In order to model I-indexed inductive-recursive definitions, where I is an arbitrary stype, we will use the category $\mathbf{Fam}(R, I)$ of I-indexed families of types. An object of $\mathbf{Fam}(R, I)$ is an I-indexed family of types, that is, an abstracted expression A for which we can prove $i : I \Rightarrow A[i] : \mathrm{type}$ in $\mathrm{TT}(R)$. An arrow from A to B is a I-indexed function, that is, an abstracted expression f for which we can prove (in $\mathrm{TT}(R)$) $i : I \Rightarrow f[i] : A[i] \to B[i]$. Again we identify A, A' s.t. we can prove $i : I \Rightarrow A[i] = A'[i] : \mathrm{type}$ and f, f' s.t. we can can prove $i : I \Rightarrow f[i] = f'[i] : B[i] \to C[i]$. We will usually omit the argument R in $\mathbf{Fam}(R, I)$.

If $\mathbf{C}$ is a category, D an object of it, then $\mathbf{C}/D$ is the slice category with objects pairs (A, f) where A is an object of $\mathbf{C}$ and f an arrow $A \to D$, and as morphisms from (A, f) to (B, g) morphisms $h : A \to B$ s.t. $g \circ h = f$. Note that we write pairs with round brackets on this level. This is different from the notation $\langle a, b\rangle$ for the pair of a and b in the logical framework.

General assumption 4.1.1 *In the following we assume in all rules* $I : \mathrm{stype}$, $i : I \Rightarrow D[i] : \mathrm{type}$ *(*D *an abstracted expression).*

4.2 Coding Several Constructors by One

We can code several constructors of an IIR into one as follows: let J be a finite index set for all constructors and A_j be the type of the jth constructor. Then replace all constructors by one constructor of type $(j : J) \to A_j$, which is definable using case distinction on $\mathbf{2}$. In case of restricted IIR we can obtain one constructor in restricted form with type $(i : I, j : J) \to A_{ij} \to C_i$, if the type of the jth constructor is $i : I \to A_{ij} \to C_i$.

In this way it will suffice to consider only IIR with one constructor in the sequel.

4.3 Restricted IIR as Initial Algebras in Slice Categories

Assume we have a restricted IIR with one constructor

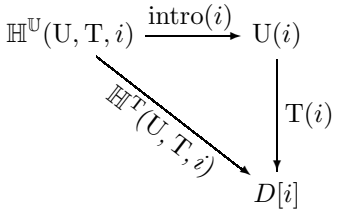
$$\mathrm{intro} : (i : I) \to \mathbb{H}^{\mathrm{U}}(\mathrm{U}, \mathrm{T}, i) \to \mathrm{U}(i)$$

(with all arguments of intro except i in uncurried form), which introduces $\mathrm{U} : I \to \mathrm{set}$ and $\mathrm{T} : (i : I, \mathrm{U}(i)) \to D[i]$. Here $\mathbb{H}^{\mathrm{U}} : (U : I \to \mathrm{set}, T : (i : I, U(i)) \to D[i], I) \to \mathrm{stype}$ with no free occurrences of U, T and i. Let further the equality rule for T be

$$\mathrm{T}(i, \mathrm{intro}(i, a)) = \mathbb{H}^{\mathrm{T}}(\mathrm{U}, \mathrm{T}, i, a) \ ,$$

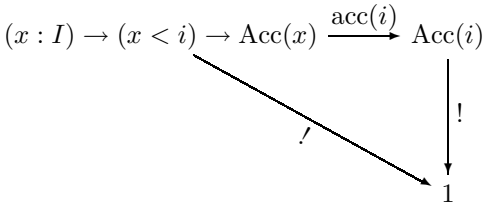
where $\mathbb{H}^{\mathrm{T}} : (U : I \to \mathrm{set}, T : (i : I, U(i)) \to D[i], i : I, \mathbb{H}^{\mathrm{U}}(U, T, i)) \to D[i]$ with no free occurrences of U, T, i or a. Now one observes that the introduction rule and equality rule are captured as an I-indexed family of algebras for $\mathbb{H}$ in $\mathbf{Fam}(I)/D$, where $\mathbb{H}(U, T) = (\mathbb{H}^{\mathrm{U}}(U, T), \mathbb{H}^{\mathrm{T}}(U, T))$:[1]

[1] To be pedantic: one has to replace $\mathbb{H}(U, T)$, $\mathbb{H}^{\mathrm{U}}(U, T)$, $\mathbb{H}^{\mathrm{T}}(U, T)$ by uncurried variants $\mathbb{H}'((U, T))$, $\mathbb{H}'^{\mathrm{U}}((U, T))$, $\mathbb{H}'^{\mathrm{T}}((U, T))$



Note that the situation generalizes the situation for non-indexed induction-recursion [10], where the rules for U and T are captured as an algebra of an appropriate endofunctor $\mathbb{F}$ on the slice category **Type**$/D$.

If $D[i] = \mathbf{1}$ and therefore $\mathbb{H}(U,T,i)$ does not depend on T, we have the important special case of a restricted indexed inductive definition. We show the example of the accessible part of a relation:

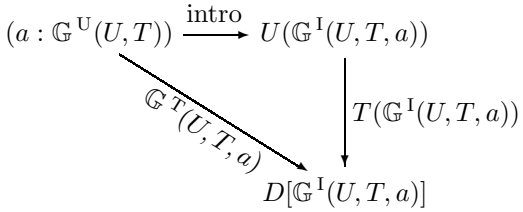


i.e. $\mathbb{H}^{\mathrm{U}}(U,T,i) = (x : I) \to (x < i) \to U(x)$.

4.4 A Diagram for General IIR

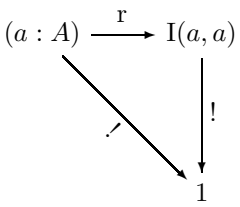
If a particular IIR does not have the restricted form then we do not prima facie know the set of constructors for a particular expression. On the other hand, given an argument to the constructor, we can determine the index of the constructed element.

To capture this situation the categorical representation must be changed. Depending on $U : I \to \text{set}$ and $T : (i : I, U(i)) \to D[i]$ we have an stype $\mathbb{G}^{\mathrm{U}}(U,T)$ of the arguments of the constructor intro, and, depending on $a : \mathbb{G}^{\mathrm{U}}(U,T)$, we have $\mathbb{G}^{\mathrm{I}}(U,T,a) : I$, which provides the i s.t. intro$(a) : \mathrm{U}_i$, and $\mathbb{G}^{\mathrm{T}}(U,T,a) : D[\mathbb{G}^{\mathrm{I}}(U,T,a)]$, which determines the value of $\mathrm{T}(i, \mathrm{intro}(a))$. The diagram is:



As a first simple instance we look at the identity relation on A. It has index set $I := A \times A$, $D[i] = \mathbf{1}$, and $\mathbb{G}^{\mathrm{U}}(U,T) = A$, $\mathbb{G}^{\mathrm{I}}(U,T,a) = \langle a,a \rangle$, and

$\mathrm{U}(\langle a, a\rangle) = \mathrm{I}_A(a, a)$:



As a second illustration we show how to obtain the rules for computability predicates for dependent types. (As in Section 3 we only give a definition containing one case, but the complete definition [5] can be obtained by expanding the definition corresponding to the additional constructors).

$$\begin{array}{rcl}
\mathbb{G}^{\,\mathrm{U}}(\Psi, \psi) & = & (A : \mathrm{Exp}) \times (p : \Psi(A)) \times \\
 & & (B : \mathrm{Exp}) \times ((a : \mathrm{Exp}) \to \psi(A, p, a) \to \psi(B\ a))\ , \\
\mathbb{G}^{\,\mathrm{I}}(\Psi, \psi, \langle A, p, B, q\rangle) & = & \Pi(A, B)\ , \\
\mathbb{G}^{\,\mathrm{T}}(\Psi, \psi, \langle A, p, B, q\rangle) & = & (b)\forall a : \mathrm{Exp}.\forall x : \psi(A, p, a).\psi(B\ a, q(a, x), b\ a)\ .
\end{array}$$

Note that in the general case we no longer have an endofunctor $\mathbf{Fam}(I)/D \to \mathbf{Fam}(I)/D$.

4.5 A Uniform Theory for Restricted and General IIR

If we look at the functors arising in general IIR we observe that we have obtained one stype $\mathbb{G}^{\mathrm{U}}(U, T)$ and two functions

$$\begin{array}{rcl}
\mathbb{G}^{\mathrm{I}}(U, T) & : & \mathbb{G}^{\mathrm{U}}(U, T) \to I\ , \\
\mathbb{G}^{\mathrm{T}}(U, T) & : & (a : \mathbb{G}^{\mathrm{U}}(U, T)) \to D[\mathbb{G}^{\mathrm{I}}(U, T, a)]\ .
\end{array}$$

where (U, T) is an element of $\mathbf{Fam}(I)/D$, $\mathbb{G}^{\mathrm{IT}}(U, T) := \langle \mathbb{G}^{\mathrm{I}}(U, T), \mathbb{G}^{\mathrm{T}}(U, T)\rangle$ an element of $E := (i : I) \times D[i]$, so $(\mathbb{G}^{\mathrm{U}}(U, T), \mathbb{G}^{\mathrm{IT}}(U, T))$ is an element of $\mathbf{Type}/E$. This can be expanded to a functor

$$\mathbb{G} : \mathbf{Fam}(I)/D \to \mathbf{Type}/E\ .$$

In restricted IIR we have for $i : I$

$$\begin{array}{rcl}
\mathbb{H}^{\mathrm{U}}(U, T, i) & : & \mathrm{stype}\ , \\
\mathbb{H}^{\mathrm{T}}(U, T, i) & : & (a : \mathbb{H}^{\mathrm{U}}(U, T, i)) \to D[i]\ ,
\end{array}$$

which can be combined to an endofunctor (with obvious arrow part and extension to U s.t. $i : I \Rightarrow U[i] : \mathrm{type}$)

$$\mathbb{H} : \mathbf{Fam}(I)/D \to \mathbf{Fam}(I)/D\ .$$

Consider the functor $\pi_i : \mathbf{Fam}(I)/D \to \mathbf{Type}/D[i]$ with object part $\pi_i(U, T) := (U[i], T[i])$ and obvious arrow part.

Every element $(U, T) : \mathbf{Fam}(I)/D$ is uniquely determined by its projections $\pi_i(U, T)$. One also notes that for every sequence of functors $\mathbb{H}_i : \mathbf{Fam}(I)/D \to$

$\mathbf{Type}/D[i]$ there exists a unique functor $\mathbb{H} : \mathbf{Fam}(I)/D \to \mathbf{Fam}(I)/D$ s.t. $\pi_i \circ \mathbb{H} = \mathbb{H}_i$. $\mathbb{H}$ and $\mathbb{G}$ will be strictly positive functors in much the same way as $\mathbb{F}$ in [10] . But since $\mathbb{H}$ is determined by $\pi_i \circ \mathbb{H}$ and both $\pi_i \circ \mathbb{H}$ and $\mathbb{G}$ are functors $\mathbf{Fam}(I)/D \longrightarrow \mathbf{Type}/E$, (where $E = (i : I) \times D[i]$ in the general case and $E = D[i]$ in the restricted case), it is more economical to more generally introduce the notion of a strictly positive functor

$$\mathbb{F} : \mathbf{Fam}(I)/D \longrightarrow \mathbf{Type}/E$$

for an arbitrary type E. From this we can derive the functors $\mathbb{G}$ and $\mathbb{H}$. As in [10], we define for E : type the type of indices for strictly positive functors

$$\frac{E : \text{type}}{\text{OP}_{I,D,E} : \text{type}} \ ,$$

together with, for $\gamma : \text{OP}_{I,D,E}$ (we omit I, D, E, if the parameter γ is given)

$$\begin{aligned} \mathbb{F}^{\text{U}}_{\gamma} &: (U : I \to \text{set}, T : (i : I, U(i)) \to D[i]) \to \text{stype} \ , \\ \mathbb{F}^{\text{T}}_{\gamma} &: (U : I \to \text{set}, T : (i : I, U(i)) \to D[i], a : \mathbb{F}^{\text{U}}_{\gamma}(U, T)) \to E \ . \end{aligned}$$

(It is straightforward to define arrow parts of these functor and the extension to arguments (U, T) s.t. $i : I \Rightarrow U[i] : \text{type}$.)

We construct elements of $\text{OP}_{I,D,E}$ in a similar way as in the non-indexed case:

- Base Case: This corresponds to having IIR with no arguments of the constructor and one only has to determine the result of E:

$$\begin{aligned} \iota &: E \to \text{OP}_{I,D,E} \ , \\ \mathbb{F}^{\text{U}}_{\iota(e)}(U, T) &= \mathbf{1} \ , \\ \mathbb{F}^{\text{T}}_{\iota(e)}(U, T, \star) &= e \ . \end{aligned}$$

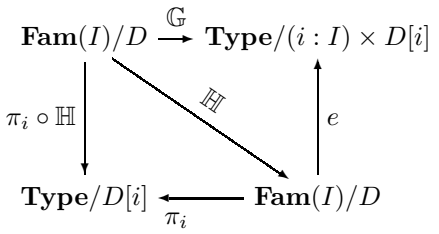
- Nondependent union of functors: This corresponds to the situation where the constructor has an argument A and (depending on $a : A$) further arguments coded as $\gamma(a)$.

$$\begin{aligned} \sigma &: (A : \text{stype}, \gamma : A \to \text{OP}_{I,D,E}) \to \text{OP}_{I,D,E} \ , \\ \mathbb{F}^{\text{U}}_{\sigma(A,\gamma)}(U, T) &= (a : A) \times \mathbb{F}^{\text{U}}_{\gamma(a)}(U, T) \ , \\ \mathbb{F}^{\text{T}}_{\sigma(A,\gamma)}(U, T, \langle a, b \rangle) &= \mathbb{F}^{\text{T}}_{\gamma(a)}(U, T, b) \ . \end{aligned}$$

- Dependent union of functors: This corresponds to the situation where the constructor has one inductive argument indexed over A. For each element $a : A$ we have to determine an index $i(a) : I$, which indicates the set $U(i(a))$, the inductive argument is chosen from. The result of T for this argument is an element of $(a : A) \to D[i(a)]$ and the other arguments depend on this function. Let $T \circ [i, f] := (x)T(i(x), f(x))$.

$$\begin{aligned} \delta &: (A : \text{stype}, i : A \to I, \gamma : ((a : A) \to D[i(a)]) \to \text{OP}_{I,D,E}) \to \text{OP}_{I,D,E} \ , \\ \mathbb{F}^{\text{U}}_{\delta(A,i,\gamma)}(U, T) &= (f : (a : A) \to U(i(a))) \times \mathbb{F}^{\text{U}}_{\gamma(T \circ [i,f])}(U, T) \ , \\ \mathbb{F}^{\text{T}}_{\delta(A,i,\gamma)}(U, T, \langle f, b \rangle) &= \mathbb{F}^{\text{T}}_{\gamma(T \circ [i,f])}(U, T, b) \ . \end{aligned}$$

We finish this subsection by drawing a diagram which shows the relationship between the functors $\mathbb{G}$ and $\mathbb{H}$ for general and restricted IIR:



where $e(U,T) = ((i:I) \times U(i), (a)\langle \pi_0(a), T(\pi_0(a), \pi_1(a))\rangle)$. So the functors in the restricted case are those which factor through an endofunctor $\mathbb{H}$, which itself is determined by the "fibers" $\pi_i \circ \mathbb{H}$.

The functors for the restricted case have codes of the form

$$\sigma(I,(i)\cdots\delta(A,f,(a)\cdots\delta(B,g,(b)\cdots\sigma(C,(c)\cdots\delta(D,(d)\cdots\iota(\langle i,e\rangle)))))) \ ,$$

ie. the codes always start with $\sigma(I,(i)\ldots$ and the innermost parts are of the form $\iota(\langle i,e\rangle)$ where i is value of the first non-inductive argument. (Note that the order of the constructors in an element of OP might depend on arguments preceding them). That a functor $\mathbb{G}$ has such a code corresponds exactly to the fact that it is obtained by the above diagram from strictly positive functors $\pi_i \circ \mathbb{H}$.

4.6 Formation and Introduction Rules for Restricted IIR

Restricted IIR (indicated by a superscript r) are given by strictly positive endofunctors $\mathbb{H}$ in the category $\mathbf{Fam}(I)/D$, which can be given by their (strictly positive) projections $\pi_i \circ \mathbb{H} : \mathbf{Fam}(I)/D \to \mathbf{Type}/D[i]$. So the set of codes for these functors is given as a family of codes for $\pi_i \circ \mathbb{H}$, and the type of codes is given as

$$\mathrm{OP}^{\mathrm{r}}_{I,D} : \mathrm{type} \ , \qquad \mathrm{OP}^{\mathrm{r}}_{I,D} = (i:I) \to \mathrm{OP}_{I,D,D[i]} \ .$$

Assume now $\gamma : \mathrm{OP}^{\mathrm{r}}_{I,D}$, $U : I \to \mathrm{set}$, $T : (i : I, U(i)) \to D[i]$, $i : I$. The object part of $\mathbb{H}$ (restricted to $U : I \to \mathrm{set}$) is defined as

$$\begin{aligned} \mathbb{H}^{\mathrm{U}}_{\gamma}(U,T,i) &:= \mathbb{F}^{\mathrm{U}}_{\gamma(i)}(U,T) : \mathrm{stype} \\ \mathbb{H}^{\mathrm{T}}_{\gamma}(U,T,i,a) &:= \mathbb{F}^{\mathrm{T}}_{\gamma(i)}(U,T,a) : D[i] \end{aligned}$$

for $a : \mathbb{H}^{\mathrm{U}}_{\gamma}(U,T,i)$.

We have the following formation rules for $\mathrm{U}^{\mathrm{r}}_{\gamma}$ and $\mathrm{T}^{\mathrm{r}}_{\gamma}$:

$$\mathrm{U}^{\mathrm{r}}_{\gamma}(i) : \mathrm{set} \ , \qquad \mathrm{T}^{\mathrm{r}}_{\gamma}(i) : \mathrm{U}^{\mathrm{r}}_{\gamma}(i) \to D[i] \ .$$

$\mathrm{U}^{\mathrm{r}}_{\gamma}(i)$ has constructor

$$\mathrm{intro}^{\mathrm{r}}_{\gamma}(i) : \mathbb{H}^{\mathrm{U}}_{\gamma}(\mathrm{U}^{\mathrm{r}}_{\gamma}, \mathrm{T}^{\mathrm{r}}_{\gamma}, i) \to \mathrm{U}^{\mathrm{r}}_{\gamma}(i) \ ,$$

and the equality rule for $\mathrm{T}^{\mathrm{r}}_{\gamma}(i)$ is:

$$\mathrm{T}^{\mathrm{r}}_{\gamma}(i, \mathrm{intro}^{\mathrm{r}}_{\gamma}(i,a)) = \mathbb{H}^{\mathrm{T}}_{\gamma}(\mathrm{U}^{\mathrm{r}}_{\gamma}, \mathrm{T}^{\mathrm{r}}_{\gamma}, i, a) \ .$$

4.7 Formation and Introduction Rules for General IIR

In general IIR (as indicated by superscript g) we have to consider strictly positive functors in $\mathbb{G} : \mathbf{Fam}(I)/D \to \mathbf{Type}/(i : I) \times D[i]$. The type of codes is given as

$$\mathrm{OP}^{\mathrm{g}}_{I,D} : \mathrm{type} \ , \qquad \mathrm{OP}^{\mathrm{g}}_{I,D} = \mathrm{OP}_{I,D,(i:I)\times D[i]} \ .$$

Assume now $\gamma : \mathrm{OP}^{\mathrm{g}}_{I,D}$, $U : I \to \mathrm{set}$, $T : (i : I, U(i)) \to D[i]$. The components of $\mathbb{G}$ needed in the following are

$$\begin{array}{rcl} \mathbb{G}^{\,\mathrm{U}}_{\gamma}(U,T) & := & \mathbb{F}^{\,\mathrm{U}}_{\gamma}(U,T) : \mathrm{stype} \\ \mathbb{G}^{\,\mathrm{I}}_{\gamma}(U,T,a) & := & \pi_0(\mathbb{F}^{\,\mathrm{T}}_{\gamma}(U,T,a)) : I \\ \mathbb{G}^{\,\mathrm{T}}_{\gamma}(U,T,a) & := & \pi_1(\mathbb{F}^{\,\mathrm{T}}_{\gamma}(U,T,a)) : D[\mathbb{G}^{\,\mathrm{I}}_{\gamma}(U,T,a)] \end{array}$$

for $a : \mathbb{G}^{\,\mathrm{U}}_{\gamma}(U,T)$.

We have essentially the same formation rules for $\mathrm{U}^{\mathrm{g}}_{\gamma}$ and $\mathrm{T}^{\mathrm{g}}_{\gamma}$

$$\mathrm{U}^{\mathrm{g}}_{\gamma} : I \to \mathrm{set} \ , \qquad \mathrm{T}^{\mathrm{g}}_{\gamma} : (i : I, \mathrm{U}^{\mathrm{g}}_{\gamma}(i)) \to D[i] \ .$$

There is one constructor for all $\mathrm{U}^{\mathrm{g}}_{\gamma}(i)$. Depending on its arguments $\mathbb{G}^{\,\mathrm{I}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a)$ determines the index it belongs to. So the introduction rule is:

$$\mathrm{intro}^{\mathrm{g}}_{\gamma} : (a : \mathbb{G}^{\,\mathrm{U}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma})) \to \mathrm{U}^{\mathrm{g}}_{\gamma}(\mathbb{G}^{\,\mathrm{I}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a)) \ .$$

The equality rule for $\mathrm{T}^{\mathrm{g}}_{\gamma}$ is:

$$\mathrm{T}^{\mathrm{g}}(\mathbb{G}^{\,\mathrm{I}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a), \mathrm{intro}^{\mathrm{g}}_{\gamma}(a)) = \mathbb{G}^{\,\mathrm{T}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a) \ .$$

4.8 Elimination Rules for IIR

We now define the induction principle both for the restricted and the general case. We define first more generally $\mathbb{F}^{\,\mathrm{IH}}_{\gamma}$ and $\mathbb{F}^{\,\mathrm{map}}_{\gamma}$ for $\gamma : \mathrm{OP}_{I,D,E}$. Assume F is a twice abstracted expression and

$$\begin{array}{l} \gamma : \mathrm{OP}_{I,D,E} \ , \quad U : I \to \mathrm{set} \ , \quad T : (i : I, U(i)) \to D[i] \ , \\ i : I, u : U(i) \Rightarrow F[i,u] : \mathrm{type} \ . \end{array}$$

The rules for $\mathbb{F}^{\,\mathrm{IH}}$ and $\mathbb{F}^{\,\mathrm{map}}$ are as follows:

$$\frac{a : \mathbb{F}^{\,\mathrm{U}}_{\gamma}(U,T)}{\mathbb{F}^{\,\mathrm{IH}}_{\gamma}(U,T,F,a) : \mathrm{type}} \qquad \frac{h : (i : I, a : U(i)) \to F[i,u]}{\mathbb{F}^{\,\mathrm{map}}_{\gamma}(U,T,F,h) : (a : \mathbb{F}^{\,\mathrm{U}}_{\gamma}(U,T)) \to \mathbb{F}^{\,\mathrm{IH}}_{\gamma}(U,T,F,a)}$$

$$\mathbb{F}^{\,\mathrm{IH}}_{\iota(e)}(U,T,F,\star) = \mathbf{1} \ ,$$

$$\mathbb{F}^{\,\mathrm{map}}_{\iota(e)}(U,T,F,h,\star) = \star \ ,$$

$$\mathbb{F}^{\,\mathrm{IH}}_{\sigma(A,\gamma)}(U,T,F,\langle a,b\rangle) = \mathbb{F}^{\,\mathrm{IH}}_{\gamma(a)}(U,T,F,b) \ ,$$

$$\mathbb{F}^{\,\mathrm{map}}_{\sigma(A,\gamma)}(U,T,F,h,\langle a,b\rangle) = \mathbb{F}^{\,\mathrm{map}}_{\gamma(a)}(U,T,F,h,b) \ ,$$

$$\mathbb{F}^{\,\mathrm{IH}}_{\delta(A,i,\gamma)}(U,T,F,\langle f,b\rangle) = ((a : A) \to F[i(a), f(a)]) \times \mathbb{F}^{\,\mathrm{IH}}_{\gamma(T\circ[i,f])}(U,T,F,b) \ ,$$

$$\mathbb{F}^{\,\mathrm{map}}_{\delta(A,i,\gamma)}(U,T,F,h,\langle f,b\rangle) = \langle h \circ [i,f], \mathbb{F}^{\,\mathrm{map}}_{\gamma(T\circ[i,f])}(U,T,F,h,b)\rangle \ .$$

In the restricted case we have now under the assumptions

$$\begin{array}{l}\gamma : \mathrm{OP}^{\mathrm{r}}_{I,D} \ , \\ i : I, a : \mathrm{U}^{\mathrm{r}}_{\gamma}(i) \Rightarrow F[i,a] : \mathrm{type} \ , \\ h : (i : I, a : \mathbb{H}^{\mathrm{U}}_{\gamma}(\mathrm{U}^{\mathrm{r}}_{\gamma}, \mathrm{T}^{\mathrm{r}}_{\gamma}, i), \mathbb{F}^{\mathrm{IH}}_{\gamma(i)}(\mathrm{U}^{\mathrm{r}}_{\gamma}, \mathrm{T}^{\mathrm{r}}_{\gamma}, F, a)) \to F[i, \mathrm{intro}^{\mathrm{r}}_{\gamma}(i,a)] \ ,\end{array}$$

$$\begin{array}{l}\mathrm{R}^{\mathrm{r}}_{\gamma,F}(h) : (i : I, a : \mathrm{U}^{\mathrm{r}}_{\gamma}(i)) \to F[i,a] \ , \\ \mathrm{R}^{\mathrm{r}}_{\gamma,F}(h, i, \mathrm{intro}^{\mathrm{r}}_{\gamma}(i,a)) = h(i, a, \mathbb{F}^{\mathrm{map}}_{\gamma(i)}(\mathrm{U}^{\mathrm{r}}_{\gamma}, \mathrm{T}^{\mathrm{r}}_{\gamma}, F, \mathrm{R}^{\mathrm{r}}_{\gamma,F}(h), a)) \ .\end{array}$$

And for the general case we have under the assumptions

$$\begin{array}{l}\gamma : \mathrm{OP}^{\mathrm{g}}_{I,D} \ , \\ i : I, a : \mathrm{U}^{\mathrm{g}}_{\gamma}(i) \Rightarrow F[i,a] : \mathrm{type} \ , \\ h : (a : \mathbb{G}^{\mathrm{U}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}), \mathbb{F}^{\mathrm{IH}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, F, a)) \to F[\mathbb{G}^{\mathrm{I}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a), \mathrm{intro}^{\mathrm{g}}_{\gamma}(a)] \ ,\end{array}$$

$$\begin{array}{l}\mathrm{R}^{\mathrm{g}}_{\gamma,F}(h) : (i : I, a : \mathrm{U}^{\mathrm{g}}_{\gamma}(i)) \to F[i,a] \ , \\ \mathrm{R}^{\mathrm{g}}_{\gamma,F}(h, \mathbb{G}^{\mathrm{I}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a), \mathrm{intro}^{\mathrm{g}}_{\gamma}(a)) = h(a, \mathbb{F}^{\mathrm{map}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, F, \mathrm{R}^{\mathrm{g}}_{\gamma,F}(h), a)) \ .\end{array}$$

In the restricted case one can show that the above rules express that the family $(\mathrm{U}^{\mathrm{r}}_{\gamma}(i), \mathrm{T}^{\mathrm{r}}_{\gamma}(i))$ for $i : I$ is (the carrier of) an initial algebra for the endofunctor $\mathbb{H}$ on $\mathbf{Fam}(I)/D$. This generalizes a theorem in [10] about the connection between inductive-recursive definitions of a set U and a function $\mathrm{T} : \mathrm{U} \to D$ and initial algebras in the slice category $\mathbf{Type}/D$.

Definition 2 *(a) The basic theory of indexed inductive recursive definitions (***Bas-IIR***) consists of the rules for the logical framework as introduced in this article, the formation and introduction rules for* OP *and the defining rules for* $\mathbb{F}^{\mathrm{U}}$, $\mathbb{F}^{\mathrm{T}}$, $\mathbb{F}^{\mathrm{IH}}$ *and* $\mathbb{F}^{\mathrm{map}}$.

(b) The theory $\mathbf{IIR^r}$ *of restricted indexed inductive recursive definitions consists of* **Bas-IIR**, *the defining rules for* OP^{r}, $\mathbb{H}^{\mathrm{U}}$, $\mathbb{H}^{\mathrm{T}}$, *and the formation/introduction/elimination/equality rules for* U^{r} *and* T^{r}.

(c) The theory $\mathbf{IIR^g}$ *of general indexed inductive recursive definitions consists of* **Bas-IIR**, *the defining rules for* OP^{g}, $\mathbb{G}^{\mathrm{U}}$, $\mathbb{G}^{\mathrm{I}}$, $\mathbb{G}^{\mathrm{T}}$, *and the formation/introduction/elimination/equality rules for* U^{g} *and* T^{g}.

(d) $\mathbf{IID^r}$ *and* $\mathbf{IID^g}$ *are the restrictions of* $\mathbf{IIR^r}$ *and* $\mathbf{IIR^g}$, *where in all rules in this article* $D[i] = \mathbf{1}$. *These are the theories of restricted and general indexed inductive definitions.*

5 The Examples Revisited

We first introduce the following abbreviations:

$$\gamma +_{\mathrm{OP}} \gamma' := \sigma(\mathbf{2}, (x)\mathrm{C}_2(x, \gamma, \gamma'))$$

and

$$\gamma_1 +_{\mathrm{OP}} \cdots +_{\mathrm{OP}} \gamma_n := (\cdots((\gamma_1 +_{\mathrm{OP}} \gamma_2) +_{\mathrm{OP}} \gamma_3) +_{\mathrm{OP}} \cdots +_{\mathrm{OP}} \gamma_n)$$

if $n \geq 3$.

So if $\gamma_1, \ldots, \gamma_n$ are codes for constructors C_i then $\gamma_1 +_{\mathrm{OP}} \cdots +_{\mathrm{OP}} \gamma_n$ is a code for a constructor C. The first argument of C codes an element i of $\{1, \ldots, n\}$. The later arguments of C are the arguments of the constructor C_i.

In the following $(-)$ stands for an abstraction (x) for a variable x, which is not used later. Let $\iota^{\mathrm{g}}_\star(a) := \iota(\langle a, \star\rangle)$, $\iota^{\mathrm{r}}_\star := \iota(\star)$.

- The trees and forests have code $\gamma : \mathrm{OP}^{\mathrm{r}}_{\mathbf{2},(-)\mathbf{1}}$ $(= \mathbf{2} \to \mathrm{OP}_{\mathbf{2},(-)\mathbf{1},\mathbf{1}})$, where
  $\gamma(\star_0) \;=\; \delta(\mathbf{1}, (-)\star_1, (-)\iota^{\mathrm{r}}_\star)$,
  $\gamma(\star_1) \;=\; \iota^{\mathrm{r}}_\star +_{\mathrm{OP}} \delta(\mathbf{1}, (-)\star_0, (-)\delta(\mathbf{1}, (-)\star_1, (-)\iota^{\mathrm{r}}_\star))$.
  Then $\mathrm{Tree} = \mathrm{U}^{\mathrm{r}}_{\mathbf{2},(-)\mathbf{1},\gamma}(\star_0)$, $\mathrm{Forest} = \mathrm{U}^{\mathrm{r}}_{\mathbf{2},(-)\mathbf{1},\gamma}(\star_1)$.

- The even number predicate has code
  $\iota^{\mathrm{g}}_\star(0) +_{\mathrm{OP}} \sigma(\mathrm{N}, (n)\delta(\mathbf{1}, (-)n, (-)\iota^{\mathrm{g}}_\star(\mathrm{S}(\mathrm{S}(n))))) : \mathrm{OP}^{\mathrm{g}}_{\mathrm{N},(-)\mathbf{1}}$
  $(= \mathrm{OP}_{\mathrm{N},(-)\mathbf{1},\mathrm{N}\times\mathbf{1}})$.

- The accessible part of a relation has code
  $(i)\delta((x : I) \times (x < i), (z)\pi_0(z), (-)\iota^{\mathrm{r}}_\star) : \mathrm{OP}^{\mathrm{r}}_{I,(-)\mathbf{1}}$
  $(= (i : I) \to \mathrm{OP}_{I,(-)\mathbf{1},\mathbf{1}})$.
  As a general IIR it has code
  $\sigma(I, (i)\delta((x : I) \times (x < i), (z)\pi_0(z), (-)\iota^{\mathrm{g}}_\star(i))) : \mathrm{OP}^{\mathrm{g}}_{I,(-)\mathbf{1}}$
  $(= \mathrm{OP}_{I,(-)\mathbf{1},I\times\mathbf{1}})$.

- The identity set has code
  $\sigma(A, (a)\iota^{\mathrm{g}}_\star(\langle a, a\rangle)) : \mathrm{OP}^{\mathrm{g}}_{A\times A,(-)\mathbf{1}}$ $(= \mathrm{OP}_{A\times A,(-)\mathbf{1},(A\times A)\times\mathbf{1}})$.

- For the Tait-style computability predicates for dependent types we have $I = \mathrm{Exp}$, $D[i] = \mathrm{Exp} \to \mathrm{set}$. The rules given in Subsection 3.2 are incomplete, additional constructors have to be added by using $+_{\mathrm{OP}}$ (the current definition actually defines the empty set). The code for the constructor given in Subsection 3.2 is

$$\begin{array}{l}
\sigma(\mathrm{Exp}, (A) \\
\quad \delta(\mathbf{1}, (-)A, (\psi_A) \\
\quad\quad \sigma(\mathrm{Exp}, (B) \\
\quad\quad\quad \delta((a : \mathrm{Exp}) \times \psi_A(\star, a), (y)(B\,\pi_0(y)), (\psi_B) \\
\quad\quad\quad\quad \iota(\langle \Pi(A, B),\ (b)\forall a : \mathrm{Exp}.\forall x : \psi_A(\star, a).\psi_B(\langle a, x\rangle, (b\,a))\rangle)))))) \\
: \mathrm{OP}^{\mathrm{g}}_{I,(i)D[i]} \quad (= \mathrm{OP}_{I,(i)D[i],(i:I)\times D[i]})\ .
\end{array}$$

- E. Palmgren's higher order universe has code
  $\gamma : \mathrm{OP}^{\mathrm{r}}_{\{0,\ldots,n\},(l)\mathrm{Op}^l(\mathrm{set})}$
  $(= (k : \{0, \ldots, n\}) \to \mathrm{OP}_{\{0,\ldots,n\},(l)\mathrm{Op}^l(\mathrm{set}),\mathrm{Op}^k(\mathrm{set})})$,
  where

$$\begin{array}{rcl}
\gamma(0) & := & \gamma_{\mathrm{Univ}} +_{\mathrm{OP}} \gamma_{\widehat{\mathrm{A}}_0} +_{\mathrm{OP}} \cdots +_{\mathrm{OP}} \gamma_{\widehat{\mathrm{A}}_n} +_{\mathrm{OP}} \gamma_{\widehat{\mathrm{B}}_0} +_{\mathrm{OP}} \\
 & & \gamma_{\mathrm{ap}^0_0} +_{\mathrm{OP}} \gamma_{\mathrm{ap}^0_1} +_{\mathrm{OP}} \cdots +_{\mathrm{OP}} \gamma_{\mathrm{ap}^0_n} +_{\mathrm{OP}} \gamma_{\mathrm{ap}^1_0} \ , \\
\gamma(i) & := & \gamma_{\widehat{\mathrm{B}}_i} +_{\mathrm{OP}} \gamma_{\mathrm{ap}^1_i} \qquad (i = 1, \ldots, n-1) \ , \\
\gamma(n) & := & \gamma_{\widehat{\mathrm{B}}_n} \ .
\end{array}$$

and

$$
\begin{array}{lcl}
\gamma_{\mathrm{Univ}} & \text{is a code for all standard universe constructors,} & \\
\gamma_{\widehat{\mathrm{A}}_k} & := & \iota(A_k)\ , \\
\gamma_{\widehat{\mathrm{B}}_k} & := & \sigma(A_k, (a)\iota(B_k(a)))\ , \\
\gamma_{\mathrm{ap}_i^0} & := & \delta(\mathbf{1}, (-)(i+1), (T_f) \\
 & & \ \delta(\mathbf{1}, (-)0, (T_u) \\
 & & \ \ \delta(T_u(\star), (-)i, (T_v) \\
 & & \ \ \ \iota(\pi_0(T_f(\star, \langle T_u(\star), T_v\rangle)))))))\ , \\
\gamma_{\mathrm{ap}_i^1} & := & \delta(\mathbf{1}, (-)(i+1), (T_f) \\
 & & \ \delta(\mathbf{1}, (-)0, (T_u) \\
 & & \ \ \delta(T_u(\star), (-)i, (T_v) \\
 & & \ \ \ \sigma(\pi_0(T_f(\star, \langle T_u(\star), T_v\rangle))), (a) \\
 & & \ \ \ \ \iota(\pi_1(T_f(\star), \langle T_u(\star), T_v\rangle)(a))))))\ .
\end{array}
$$

(To be formally correct we would have to work with N_{n+1} instead of $\{0, \ldots, n\}$).

6 Further Results

There is an extended version [11] of this article which contains some additional topics.

In particular we show the consistency of our theories by constructing a model in classical set theory where type-theoretic function spaces are interpreted as full set-theoretic function spaces, and some axioms for large cardinals are used for interpreting the type of sets closed under IIR and also to interpret the logical framework. This construction is a generalization of the model for non-indexed induction-recursion given in [9].

Furthermore, we show that the restricted and general form of an IIR are intertranslatable under certain conditions. We also show how to simulate certain forms of IIR by non-indexed inductive-recursive definitions (IR).

References

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

[2] T. Altenkirch, V. Gaspes, B. Nordström, and B. von Sydow. A user's guide to ALF. http://www.www.cs.chalmers.se/ComputingScience/Research/Logic/alf/guide.html, 1996.

[3] A. Bove and V. Capretta. Nested general recursion and partiality in type theory. To appear in the Proceedings of TPHOL 2001.

[4] C. Coquand. Agda. 2000. http://www.cs.chalmers.se/ catarina/agda/.

[5] 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.

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

[7] P. Dybjer. Inductive families. *Formal Aspects of Computing*, 6:440–465, 1994.

[8] P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2), June 2000.

[9] P. Dybjer and A. Setzer. A finite axiomatization of inductive-recursive definitions. In J.-Y. Girard, editor, *Typed Lambda Calculi and Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 129–146. Springer, April 1999.

[10] P. Dybjer and A. Setzer. Induction-recursion and initial algebras. Submitted for publication, April 2000.

[11] P. Dybjer and A. Setzer. Indexed induction-recursion. To appear as a Report of Institut Mittag-Leffler, 2001.

[12] L. Magnusson and B. Nordström. The ALF proof editor and its proof engine. In *Types for proofs and programs*, volume 806 of *Lecture Notes in Computer Science*, pages 213–317. Springer, 1994.

[13] P. Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, pages 73–118. North-Holland, 1975.

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

[15] P. Martin-Löf. On the meaning of the logical constants and the justification of the logical laws, 1983. Notes from a series of lectures given in Siena.

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

[17] 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. Reprinted version of an unpublished report from 1972.

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

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

[20] 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.

[21] 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.