Induction is not derivable in second order dependent type theory

Herman Geuvers*

Department of Computer Science, University of Nijmegen, The Netherlands

Abstract. This paper proves the non-derivability of induction in second order dependent type theory ($\lambda P2$). This is done by providing a model construction for $\lambda P2$, based on a saturated sets like interpretation of types as sets of terms of a weakly extensional combinatory algebra. We give counter-models in which the induction principle over natural numbers is not valid. The proof does not depend on the specific encoding for natural numbers that has been chosen (like e.g. polymorphic Church numerals), so in fact we prove that there can not be an encoding of natural numbers in $\lambda P2$ such that the induction principle is satisfied. The method extends immediately to other data types, like booleans, lists, trees, etc.
In the process of the proof we establish some general properties of the models, which we think are of independent interest. Moreover, we show that the Axiom of Choice is not derivable in $\lambda P2$.

1 Introduction

In second order dependent type theory, $\lambda P2$, we can encode all kinds of inductive data types, like the types of natural numbers, lists, trees etcetera. This is usually done via the Böhm-Berarducci encoding (see [Girard et al. 1989] for a general exposition), which yields e.g. the well-known polymorphic Church numerals as interpretation of the natural numbers. This encoding already works for non-dependent second order type theory (the well-known polymorphic λ-calculus $\lambda 2$), but dependent types give the extra advantage that we can also state the *induction principle* for the inductive data types. For example, if nat is the type of polymorphic Church numerals with zero O and successor function succ, then the induction principle is represented by the type ind defined as

$$\mathsf{ind} := \Pi P{:}\mathsf{nat}\to\star.(P\mathsf{O})\to(\Pi y{:}\mathsf{nat}.(Py)\to(P(\mathsf{succ}y)))\to\Pi x{:}\mathsf{nat}.(Px).$$

Here, $\star$ denotes the 'kind' (universe) of all types, which captures both the sets ($\mathsf{nat}:\star$) and the propositions ($\mathsf{ind}:\star$). The induction principle for nat is said to be *derivable* in $\lambda P2$ if there is a closed term of type ind.

In this paper we show that the induction principle for nat is not derivable in $\lambda P2$. As a matter of fact, we prove something stronger: the non-derivability of

* email: herman@cs.kun.nl, fax: +31 24 3652525

induction does not depend on the specific choice of the encoding of the natural numbers: given any (closed) type N with $0 : N$ and $s : N \to N$, there can be no closed term of type $\Pi P{:}N \to \star.(P0) \to (\Pi y{:}N.(Py) \to (P(sy))) \to \Pi x{:}N.(Px)$. This rules out any 'smart' encoding of the natural numbers (like the N above) for which induction would be provable in $\lambda P2$. What a 'smart encoding' could possibly look like, see the small diversion below in 1.1.

It should be pointed out here that, of course, inductive reasoning can easily represented in $\lambda P2$ by 'relativizing' all statements about nat to the inductive natural numbers. If we let $\mathsf{Ind}\, x$ say that x is an 'inductive natural number', defined in $\lambda P2$ as follows,

$$\mathsf{Ind}\, x := \Pi P{:}\mathsf{nat} \to \star.(P\mathsf{O}) \to (\Pi y{:}\mathsf{nat}.(Py) \to (P(\mathsf{succ} y))) \to (Px),$$

we can relativize $\Pi x{:}\mathsf{nat}.\varphi$ to $\Pi x{:}\mathsf{nat}.(\mathsf{Ind}\, x) \to \varphi$. Then one can reason by induction, just because all statements about nat are restricted to the inductive natural numbers. However, this does not give us an *inductive type* of natural numbers.

Our result extends immediately to other inductive data types, so induction is not derivable for any encoding of any inductive data type in $\lambda P2$. Also we show in this paper that the induction principle for one data type can not be derived from the induction principle for another data type. The results extend immediately to other systems like the Calculus of Constructions (without inductive types). In [Streicher 1991], also a non-derivability induction result is proved, using a realizability semantics, but only for one specific encoding of the natural numbers, as polymorphic Church numerals. Our proof of non-derivability uses a fairly simple model construction which originates from [Geuvers 1996] and [Stefanova and Geuvers 1996]. The model we construct has some similarities with the one used in [Berardi 1993] to justify encoding mathematics in the Calculus of Constructions. To establish our main result we construct a model in which the type that represents induction is empty.

Apart from the induction principle we also show the non-derivability of the Axiom of Choice.

1.1 Small diversion: a possible smart encoding of the naturals

One may wonder whether there are other 'smarter' encodings of the natural numbers for which induction *is* provable. In this subsection we suggest a possible different encoding of the naturals. Our final result implies that induction is also non-derivable for this representation. Let us define

$$N := \exists x{:}\mathsf{nat}.(\mathsf{Ind}\, x),$$

with $\mathsf{Ind}\, x$ saying that x is an 'inductive natural number', defined as above. Now the 'inductivity' of the natural numbers is 'built in' in their encoding. ($\exists$ is defined in the well-known second order way: $\exists x{:}\sigma.\tau := \Pi\alpha{:}\star.(\Pi x{:}\sigma.\tau \to \alpha) \to \alpha$.) By using the definable $\exists$-elim and $\exists$-intro rules, it is now easy to define $\underline{\mathsf{O}}$, $\underline{\mathsf{succ}}$

for this encoding:

$$\begin{aligned}\underline{\mathsf{O}} &:= \lambda\alpha : \star.\lambda h{:}(\Pi x{:}\mathsf{nat}.(\mathsf{Ind}x)\to\alpha).h\mathsf{O}q_{\mathsf{O}},\\ \underline{\mathsf{succ}} &:= \lambda n{:}N.nN\big(\lambda x{:}\mathsf{nat}.\lambda p{:}(\mathsf{Ind}\,x).\\ &\qquad \lambda\alpha{:}\star.\lambda h{:}(\Pi y{:}\mathsf{nat}.(\mathsf{Ind}y)\to\alpha).h(\mathsf{succ}\,x)(q_{\mathsf{succ}}xp)\big),\end{aligned}$$

where q_{O} and q_{succ} are terms such that $q_{\mathsf{O}} : (\mathsf{Ind}\,\mathsf{O})$ and $q_{\mathsf{succ}} : \Pi x{:}\mathsf{nat}.(\mathsf{Ind}\,x)\to(\mathsf{Ind}\,(\mathsf{succ}\,x))$. One may wonder whether the induction principle is derivable for the type N. It is not the case, which can intuitively be grasped from the fact that there is no 'coherence' among the possible proofs of $\mathsf{Ind}\,x$. (There are many possible proofs of $\mathsf{Ind}\,\mathsf{O}$, which are not all captured.)

2 Second order dependent type theory

The system of second order dependent type theory, $\lambda P2$, is an extension of the polymorphic λ-calculus with dependent types and it was first introduced in [Longo and Moggi 1988]. It can be seen as a subsystem of the Calculus of Constructions ([Coquand and Huet 1988], [Coquand 1990]), where the operations of forming type constructors are restricted to second order ones. (So, one can quantify over type constructors of kind $\sigma\to\star$, but one can not form type constructors of kind $(\sigma\to\star)\to\star$.) It can also be seen as an extension of the first order system λP, where quantification over type constructors has been added. For an extensive discussion on these systems and their relations, we refer to [Barendregt 1992] or [Geuvers 1993]. Here we just define the system $\lambda P2$ and give some initial motivation for it.

Definition 1. *The type system $\lambda P2$ is defined as follows. The set of* pseudo-terms*,* T*, is defined by*

$$\mathsf{T} ::= \star \mid \mathsf{Kind} \mid \mathrm{Var} \mid (\Pi\mathrm{Var}{:}\mathsf{T}.\mathsf{T}) \mid (\lambda\mathrm{Var}{:}\mathsf{T}.\mathsf{T}) \mid \mathsf{T}\mathsf{T},$$

where Var *is a countable set of variables. On* T *we have the usual notion of* β*-reduction,* $\longrightarrow_\beta$*. We adopt from the untyped λ-calculus the conventions of denoting the transitive reflexive closure of* $\longrightarrow_\beta$ *by* $\twoheadrightarrow_\beta$ *and the transitive symmetric closure of* $\twoheadrightarrow_\beta$ *by* $=_\beta$*.*

The typing of terms is done under the assumption of specific types for the free variables that occur in the term. This is done in a context*, a finite sequence of declarations* $\Gamma = v_1{:}T_1,\ldots,v_n{:}T_n$ *(the v are variables and the T are pseudo-terms). Typing judgments are written as* $\Gamma \vdash M : T$*, with Γ a context and M and T pseudo-terms.*

The deduction rules for $\lambda P2$ are as follows. (v ranges over Var*, s, s_1 and s_2 range over* $\{\star,\mathsf{Kind}\}$ *and M, N, T and U range over* T*.)*

$$(axiom)\ \vdash \star : \mathsf{Kind} \quad (var)\frac{\Gamma\vdash T : \star/\mathsf{Kind}}{\Gamma, v{:}T \vdash v : T} \quad (weak)\frac{\Gamma\vdash T : \star/\mathsf{Kind} \quad \Gamma\vdash M : U}{\Gamma, v{:}T\vdash M : U}$$

$$(\Pi)\ \frac{\Gamma \vdash T : s_1 \quad \Gamma, v{:}T \vdash U : s_2}{\Gamma \vdash \Pi v{:}T.U : s_2}\ \textit{if } (s_1, s_2) \neq (\mathsf{Kind}, \mathsf{Kind})$$

$$(\lambda)\ \frac{\Gamma, v{:}T \vdash M : U \quad \Gamma \vdash \Pi v{:}T.U : s}{\Gamma \vdash \lambda v{:}T.M : \Pi v{:}T.U}$$

$$(\mathit{app})\ \frac{\Gamma \vdash M : \Pi v{:}T.U \quad \Gamma \vdash N : T}{\Gamma \vdash MN : U[N/v]} \qquad (\mathit{conv}_\beta)\ \frac{\Gamma \vdash M : T \quad \Gamma \vdash U : s}{\Gamma \vdash M : U}\ \textit{if } T =_\beta U$$

In the rules (var) and (weak) it is always assumed that the newly declared variable is fresh, that is, it has not yet been declared in Γ. *For convenience, we split up the set* Var *into a set* $\mathrm{Var}^\star$, *the* object variables, *and* $\mathrm{Var}^{\mathsf{Kind}}$, *the* constructor variables. *Object variables will be denoted by* $x, y, z, \ldots$ *and constructor variables by* $\alpha, \beta, \ldots$. *In the rules (var) and (weak), we take the variable* v *out of* $\mathrm{Var}^\star$ *if* $s = \star$ *and out of* $\mathrm{Var}^{\mathsf{Kind}}$ *if* $s = \mathsf{Kind}$.

We call a pseudo-term M *well-typed* if there is a context Γ and another pseudo-term N such that either $\Gamma \vdash M : N$ or $\Gamma \vdash N : M$ is derivable. The well-typed terms can be split into the following disjoint subsets:

- $\{\mathsf{Kind}\}$,
- the set of *kinds*: terms A such that $\Gamma \vdash A : \mathsf{Kind}$ for some Γ; this includes $\star$. In $\lambda P2$ all kinds are of the form $\Pi x_1{:}\sigma_1 \ldots \Pi x_n{:}\sigma_n.\star$, with $\sigma_1, \ldots, \sigma_n$ *types* and $x_1, \ldots, x_n \in \mathrm{Var}^\star$.
- the set of *constructors*: terms of type a 'kind', i.e. terms P such that $\Gamma \vdash P : A$ for some kind A; this includes the *types*, terms of type $\star$.
  In $\lambda P2$ all constructors are of one of the following forms
  - $\alpha \in \mathrm{Var}^{\mathsf{Kind}}$,
  - Pt, with P a constructor and t an *object*,
  - $\lambda x{:}\sigma.P$, with σ a type, P a constructor, $x \in \mathrm{Var}^\star$,
  - $\Pi x{:}\sigma.\tau$, with σ and τ types, $x \in \mathrm{Var}^\star$,
  - $\Pi \alpha{:}A.\tau$, with A a kind, τ a type, $\alpha \in \mathrm{Var}^{\mathsf{Kind}}$.
- the *objects*: terms of type a 'type', i.e. terms M such that $\Gamma \vdash M : \sigma$ for some type σ. In $\lambda P2$ all objects are of one of the following forms
  - $x \in \mathrm{Var}^\star$,
  - qt, with q and t an objects,
  - qP, with P a constructor and q an object,
  - $\lambda x{:}\sigma.t$, with σ a type, t an object, $x \in \mathrm{Var}^\star$,
  - $\lambda \alpha{:}A.t$, with A a kind, t an object, $\alpha \in \mathrm{Var}^{\mathsf{Kind}}$.

Convention We denote kinds by $A, B, C, \ldots$, types by $\sigma, \tau, \ldots$, constructors by $P, Q, \ldots$ and objects by $t, q, \ldots$.
If v is not free in U, we denote – as usual – $\Pi v{:}T.U$ by $T \to U$. In arrow types, we let brackets associate to the right, so $T \to T \to T$ denotes $T \to (T \to T)$. In application types, we let brackets associate to the left, so MNP denotes $(MN)P$.

Data types and formulas in $\lambda P2$ The well-known encoding of inductive data types in polymorphic λ-calculus extends immediately to $\lambda P2$. For the general procedure we refer to [Girard et al. 1989]. Here we give some examples. It is also standard that these inductive data types come together with the possibility of defining functions by iteration. We do not discuss the iteration scheme, as it is outside the scope of this paper. We do give, for each data type the associated induction principle. In this paper we show that the induction principle for natural numbers is not provable in $\lambda P2$. However the same method applies immediately to other data types, like the ones given below.

1. The natural numbers can be encoded by $\mathsf{nat} := \Pi\alpha{:}\star.\alpha\to(\alpha\to\alpha)\to\alpha$, with zero and successor:

$$\begin{aligned}\mathsf{O} &:= \lambda\alpha{:}\star.\lambda x{:}\alpha.\lambda f{:}\alpha\to\alpha.x,\\ \mathsf{succ} &:= \lambda n{:}\mathsf{nat}.\lambda\alpha{:}\star.\lambda x{:}\alpha.\lambda f{:}\alpha\to\alpha.f(n\alpha x f).\end{aligned}$$

   The induction principle reads

$$\mathsf{ind}_{\mathsf{nat}} := \Pi P{:}\mathsf{nat}\to\star.(P\mathsf{O})\to(\Pi y{:}\mathsf{nat}.(Py)\to(P(\mathsf{succ}y)))\to\Pi x{:}\mathsf{nat}.(Px).$$

2. The list over a given carrier type σ can be encoded by $\mathsf{list}_\sigma := \Pi\alpha{:}\star.\alpha\to(\sigma\to\alpha\to\alpha)\to\alpha$, with empty list and 'cons' map:

$$\begin{aligned}\mathsf{nil} &:= \lambda\alpha{:}\star.\lambda x{:}\alpha.\lambda f{:}\sigma\to\alpha\to\alpha.x,\\ \mathsf{cons} &:= \lambda a{:}\sigma.\lambda l{:}\mathsf{list}_\sigma.\lambda\alpha{:}\star.\lambda x{:}\alpha.\lambda f{:}\sigma\to\alpha\to\alpha.fa(l\alpha x f).\end{aligned}$$

   As we are in $\lambda P2$, we can not define list as a type constructor $\mathsf{list} := \lambda\alpha{:}\star.\mathsf{list}_\alpha : \star\to\star$, simply because the kind $\star\to\star$ is not available in $\lambda P2$. For simplicity we write list for list_σ if the σ is clear from the context.
   The induction principle reads

$$\mathsf{ind}_{\mathsf{list}} := \Pi P{:}\mathsf{list}\to\star.(P\mathsf{nil})\to(\Pi a{:}\sigma.\Pi y{:}\mathsf{list}.(Py)\to(P(\mathsf{cons}ay)))\to\Pi x{:}\mathsf{list}.(Px).$$

3. The well-founded labeled trees of branching type τ and with labels in σ can be encoded by $\mathsf{tree}_{\tau\sigma} := \Pi\alpha{:}\star.(\sigma\to\alpha)\to(\sigma\to(\tau\to\alpha)\to\alpha)\to\alpha$, with maps leaf and join (taking a label and a 'τ-sequence' of trees and returning a tree):

$$\begin{aligned}&\mathsf{leaf} := \lambda a{:}\sigma.\lambda\alpha{:}\star.\lambda x{:}\sigma\to\alpha.\lambda f{:}\sigma\to(\tau\to\alpha)\to\alpha.xa,\\ &\mathsf{join} := \lambda a{:}\sigma.\lambda t{:}\tau\to\mathsf{tree}_{\tau\sigma}.\lambda\alpha{:}\star.\lambda x{:}\sigma\to\alpha.\lambda f{:}\sigma\to(\tau\to\alpha)\to\alpha.fa(\lambda z{:}\tau.tz\alpha x f).\end{aligned}$$

   The remark about not being able to define $\mathsf{list} : \star\to\star$ also applies to tree. We omit the indices in tree if no confusion arises. The induction principle reads

$$\begin{aligned}\mathsf{ind}_{\mathsf{tree}} := \Pi P{:}\mathsf{tree}\to\star.&(\Pi a{:}\sigma.(P(\mathsf{leaf}a)))\to\\ &(\Pi a{:}\sigma.\Pi y{:}\tau\to\mathsf{tree}.(\Pi z{:}\tau.(P(yz)))\to(P(\mathsf{join}ay)))\to\Pi x{:}\mathsf{tree}.(Px).\end{aligned}$$

There is a *formulas-as-types* embedding from constructive second order predicate logic into $\lambda P2$.

3 Model construction for $\lambda P2$

The model notion for $\lambda P2$ we give is not a general (categorical) one, but a description of a class of models, which is the same as in [Geuvers 1996]. It can be extended to a class of models for the Calculus of Constructions, which is done in [Stefanova and Geuvers 1996].

The models of $\lambda P2$ are built from *weakly extensional combinatory algebras* (weca for short). A *combinatory algebra* (ca for short) is a tuple $\mathcal{A} = \langle \mathbf{A}, \cdot, \mathbf{k}, \mathbf{s} \rangle$, with $\mathbf{A}$ a set, $\cdot$ a binary function from $\mathbf{A} \times \mathbf{A}$ to $\mathbf{A}$ (as usual denoted by infix notation), $\mathbf{k}, \mathbf{s} \in \mathbf{A}$ such that $(\mathbf{k} \cdot a) \cdot b = a$ and $((\mathbf{s} \cdot a) \cdot b) \cdot c = (a \cdot c) \cdot (b \cdot c)$. For $\mathcal{A}$ a combinatory algebra, the set of *terms over* $\mathcal{A}$, $\mathcal{T}(\mathcal{A})$, is defined by letting $\mathcal{T}(\mathcal{A})$ contain infinitely many variables $v_1, v_2, \ldots$ and distinct elements c_a for every $a \in \mathbf{A}$, and letting $\mathcal{T}(\mathcal{A})$ be closed under application (the operation $\cdot$). Given a term t and a valuation ρ, mapping variables to elements of $\mathbf{A}$, the *interpretation of t in $\mathbf{A}$ under ρ*, notation $[\![t]\!]^{\mathcal{A}}_{\rho}$, is defined in the usual way ($[\![c_a]\!]^{\mathcal{A}}_{\rho} = a$, $[\![MN]\!]^{\mathcal{A}}_{\rho} = [\![M]\!]^{\mathcal{A}}_{\rho} \cdot [\![N]\!]^{\mathcal{A}}_{\rho}$, etcetera). An important property of cas is that they are *combinatory complete*, i.e. if $t[v] \in \mathcal{T}(\mathcal{A})$ is a term with free variable v, then there is an element in $\mathbf{A}$, usually denoted by $\lambda^* v.t[v]$, such that $\forall x((\lambda^* v.t[v]) \cdot x = t[x])$ in $\mathcal{A}$. (More technically, this means that $[\![(\lambda^* v.t[v]) \cdot x]\!]^{\mathcal{A}}_{\rho} = [\![t[x]]\!]^{\mathcal{A}}_{\rho}$ for all ρ.) A ca is *weakly extensional* if $[\![t_1]\!]^{\mathcal{A}}_{\rho(x:=a)} = [\![t_2]\!]^{\mathcal{A}}_{\rho(x:=a)}$ for all $a \in \mathbf{A}$ implies that $[\![\lambda^* x.t_1]\!]^{\mathcal{A}}_{\rho} = [\![\lambda^* x.t_2]\!]^{\mathcal{A}}_{\rho}$. In other words: a ca is weakly extensional if abstraction is a *function* on the weca $\langle \mathcal{T}(\mathcal{A}), \cdot, \mathbf{k}, \mathbf{s} \rangle$, i.e. if (in $\mathcal{T}(\mathcal{A})$) $t_1 = t_2$, then $\lambda^* x.t_1 = \lambda^* x.t_2$.

The need for *weakly extensional* cas comes from the fact that we want

$$M =_{\beta} N \Rightarrow (\![M]\!)_{\rho} = (\![N]\!)_{\rho} \text{ for all } \rho,$$

where $(\![-]\!)_{\rho}$ interprets pseudo-terms as elements of $\mathbf{A}$, using a valuation ρ for the free variables. Of course, $(\![-]\!)_{\rho}$ is close to $[\![-]\!]^{\mathcal{A}}_{\rho}$, except for the fact that now we also have to interpret abstraction: under $(\![-]\!)_{\rho}$, λ is interpreted as λ^*. [1]

Example 1. 1. A standard example of a weca is $\mathbf{\Lambda}$, consisting of the classes of open λ-terms modulo β-equality. So, $\mathbf{A}$ is just Λ/β and $[M] = [N]$ iff $M =_{\beta} N$. It is easily verified that this yields a weca.

2. Given a set of constants C, we define the weca $\mathbf{\Lambda}(C)$ as the equivalence classes of open λ_C-terms (i.e. lambda-terms over the constant set C) modulo βc-equality, where the c-equality rules says

$$cN =_c c \qquad \lambda v.c =_c c$$

   for all $c \in C$ and $N \in \Lambda_C$.

3. Another example of a weca is $\mathbf{1}$, the *degenerate* weca where $\mathbf{A} = 1$, the one-element set. In this case $\mathbf{k} = \mathbf{s}$, which is usually not allowed in combinatory algebras, but note that we do allow it here.

[1] In general, for cas, $M = N \not\Rightarrow (\![M]\!)_{\rho} = (\![N]\!)_{\rho}$ (e.g. take combinatory logic and $M \equiv x$, $N \equiv Ix$). However, for wecas this implication holds.

The types of $\lambda P2$ will be interpreted as subsets of $\mathbf{A}$.

Definition 2. *A* polyset structure *over the weakly extensional combinatory algebra $\mathcal{A}$ is a collection $\mathcal{P} \subseteq \wp(\mathbf{A})$ such that*

1. $\mathbf{A} \in \mathcal{P}$,
2. *$\mathcal{P}$ is closed under arbitrary intersection $\bigcap$,*
3. *$\mathcal{P}$ is closed under* dependent products, *i.e. if $X \in \mathcal{P}$ and $F : X \to \mathcal{P}$, then $\Pi_{t \in X} F(t) \in \mathcal{P}$, where $\Pi_{t\in X} F(t)$ is defined as*
$$\{a \in \mathbf{A} \mid \forall t \in X(a \cdot t \in F(t))\}.$$

The elements of a polyset structure are called polysets. *If F is the constant function with value Y, we write $X \to Y$ instead of $\Pi_{t\in X} Y$.*

Example 2.
1. We obtain the *full polyset structure* over the weca $\mathcal{A}$ if we take $\mathcal{P} = \wp(\mathbf{A})$.
2. The *simple polyset structure* over the weca $\mathcal{A}$ is obtained by taking $\mathcal{P} = \{\emptyset, \mathbf{A}\}$. It is easily verified that this is a polyset structure.
3. Given the weca $\mathbf{\Lambda}(C)$ as defined in Example 1 (so C is a set of constants), we define the *polyset structure generated from C* by
$$\mathcal{P} := \{X \subseteq \mathbf{\Lambda}(C) \mid X = \emptyset \vee C \subseteq X\}.$$
To show that $\mathcal{P}$ is a polyset structure, the only interesting thing is to verify that $\mathcal{P}$ is closed under dependent product. So, let $X \in \mathcal{P}$ and $F : X \to \mathcal{P}$. We distinguish cases: if $X = \emptyset$, then $\Pi_{t\in X} F(t) = \mathbf{\Lambda}(C) \in \mathcal{P}$; if $F(t) = \emptyset$ for some $t \in X$, then $\Pi_{t\in X} F(t) = \emptyset \in \mathcal{P}$; in all other cases $C \subseteq \Pi_{t\in X} F(t)$, because for $c \in C$ and $t \in X$, $ct =_c c \in C \subseteq F(t)$, so $ct \in F(t)$.
4. Given the weca $\mathcal{A}$ and a set $C \subseteq \mathbf{A}$ such that $\forall a, b \in \mathbf{A}(a \cdot b \in C \Rightarrow a \in C$, we define the *power polyset structure of C* by
$$\mathcal{P} := \{X \subseteq \mathbf{A} \mid X \subseteq C \vee X = \mathbf{A}\}.$$
To check that this is a polyset structure, one only has to verify that, for $X \in \mathcal{P}$ and $F : X \to \mathcal{P}$, $\Pi_{t\in X} F(t) \in \mathcal{P}$. This follows from an easy case distinction: $\forall t \in X(F(t) = \mathbf{A})$ or $\exists t \in X(F(t) \subseteq C)$.
An interesting instance of a power polyset structure is the one arising from $C = \mathrm{HNF}$, the set of λ-terms with a head-normal-form, in the weca Λ/β.

The dependent product of a polyset structure will be used to interpret types of the form $\Pi x{:}\sigma.\tau$, where both σ and τ are types. The intersection will be used to interpret types of the form $\Pi\alpha{:}A.\sigma$, where σ is a type and A is a kind. To interpret kinds we need a *predicative structure.*

Definition 3. *For $\mathcal{P}$ a polyset structure, the* predicative structure over $\mathcal{P}$ *is the collection of sets $\mathcal{N}$ defined inductively by*

1. $\mathcal{P} \in \mathcal{N}$,

2. If $X \in \mathcal{P}$ and $\forall t \in X(F(t) \in \mathcal{N}$, then $\prod_{t\in X} F(t) \in \mathcal{N}$.

If F is a constant function with value $\mathcal{P}$, we write $X \to \mathcal{P}$ in stead of $\prod_{t\in X} \mathcal{P}$.

Definition 4. *If $\mathcal{A}$ is a combinatory algebra, $\mathcal{P}$ a polyset structure over $\mathcal{A}$ and $\mathcal{N}$ the predicative structure over $\mathcal{P}$, then we call the tuple $\langle \mathcal{A}, \mathcal{P}, \mathcal{N} \rangle$ a $\lambda P2$-model.*

The predicative structure over a polyset structure $\mathcal{P}$ is intended to give a domain of interpretation for the kinds. For example, if the type σ is interpreted as the polyset X, then the kind $\sigma \to \sigma \to \star$ is interpreted as $\prod_{t\in X} \prod_{q\in X} \mathcal{P}$, for which we usually write $X \to X \to \mathcal{P}$.

We now define three interpretation functions, one for kinds, $\mathcal{V}(-)$, that maps kinds to elements of $\mathcal{N}$, one for constructors (and types), $[\![-]\!]$, that maps constructors to elements of $\bigcup \mathcal{N}$ (and types to elements of $\mathcal{P}$, which is a subset of $\bigcup \mathcal{N}$) and one for objects, $(\![-]\!)$, that maps objects to elements of the combinatory algebra $\mathcal{A}$. All these interpretations are parametrized by *valuations*, assigning values to the free variables (declared in the context).

Let in the following $\mathcal{M} = \langle \mathcal{A}, \mathcal{P}, \mathcal{N} \rangle$ be a $\lambda P2$-model: $\mathcal{A} = \langle \mathbf{A}, \cdot, \mathbf{k}, \mathbf{s} \rangle$ is a combinatory algebra, $\mathcal{P}$ is a polyset structure over $\mathcal{A}$ and $\mathcal{N}$ is the predicative structure over the polyset structure $\mathcal{P}$.

Definition 5. *A* constructor variable valuation *is a map ξ from* $\mathrm{Var}^{\mathsf{Kind}}$ *to* $\bigcup \mathcal{N}$. *An* object variable valuation *is a map ρ from* $\mathrm{Var}^{\star}$ *to* $\mathbf{A}$.

Definition 6. *For ρ an object variable valuation, we define the map $(\![-]\!)^{\mathcal{M}}_{\rho}$ from the set of objects to $\mathbf{A}$ as follows. (We leave the model $\mathcal{M}$ implicit.)*

$$
\begin{aligned}
(\![x]\!)_{\rho} &:= \rho(x), \\
(\![tq]\!)_{\rho} &:= (\![t]\!)_{\rho} \cdot (\![q]\!)_{\rho}, \ \textit{if } q \textit{ is an object}, \\
(\![tQ]\!)_{\rho} &:= (\![t]\!)_{\rho}, \ \textit{if } Q \textit{ is a constructor}, \\
(\![\lambda x{:}\sigma.t]\!)_{\rho} &:= \lambda^{*} v.(\![t]\!)_{\rho(x:=v)}, \ \textit{if } \sigma \textit{ is a type}, \\
(\![\lambda \alpha{:}A.t]\!)_{\rho} &:= (\![t]\!)_{\rho}, \ \textit{if } A \textit{ is a kind.}
\end{aligned}
$$

Definition 7. *For ρ an object variable valuation and ξ a constructor variable valuation, we define the maps $\mathcal{V}(-)^{\mathcal{M}}_{\xi\rho}$ and $[\![-]\!]^{\mathcal{M}}_{\xi\rho}$ respectively from kinds to $\mathcal{N}$ and from constructors to $\bigcup \mathcal{N}$ as follows. (We leave the model $\mathcal{M}$ implicit.)*

$$
\begin{aligned}
\mathcal{V}(\star)_{\xi\rho} &:= \mathcal{P}, \\
\mathcal{V}(\Pi x{:}\sigma.B)_{\xi\rho} &:= \prod_{t \in [\![\sigma]\!]_{\xi\rho}} \mathcal{V}(B)_{\xi\rho(x:=t)}, \\
[\![\alpha]\!]_{\xi\rho} &:= \xi(\alpha), \\
[\![\Pi \alpha{:}A.\tau]\!]_{\xi\rho} &:= \bigcap_{a \in \mathcal{V}(A)_{\xi\rho}} [\![\tau]\!]_{\xi(\alpha:=a)\rho}, \ \textit{if } A \textit{ is a kind}, \\
[\![\Pi x{:}\sigma.\tau]\!]_{\xi\rho} &:= \Pi_{t \in [\![\sigma]\!]_{\xi\rho}} [\![\tau]\!]_{\xi\rho(x:=t)}, \ \textit{if } \sigma \textit{ is a type}, \\
[\![Pt]\!]_{\xi\rho} &:= [\![P]\!]_{\xi\rho}((\![t]\!)_{\rho}), \\
[\![\lambda x{:}\sigma.P]\!]_{\xi\rho} &:= \lambda t \in [\![\sigma]\!]_{\xi\rho}.[\![P]\!]_{\xi\rho(x:=t)}.
\end{aligned}
$$

Note that $\mathcal{V}(A)_{\xi\rho}$ and $[\![P]\!]_{\xi\rho}$ may be undefined. For example, in the definition of $[\![Pt]\!]_{\xi\rho}$, $(\![t]\!)_\rho$ may not be in the domain of $[\![P]\!]_{\xi\rho}$, in the definition of $[\![\Pi x{:}\sigma.\tau]\!]_{\xi\rho}$, $[\![\sigma]\!]_{\xi\rho}$ may not be a polyset and in the definition of $\mathcal{V}(\Pi x{:}\sigma.B)_{\xi\rho}$, $[\![\sigma]\!]_{\xi\rho}$ may not be defined. From the Soundness Theorem (1) it will follow that, under certain natural conditions for ξ and rho, $\mathcal{V}(A)_{\xi\rho}$ and $[\![P]\!]_{\xi\rho}$ are well-defined.

Definition 8. *For Γ a $\lambda P2$-context, ρ an object variable valuation and ξ a constructor variable valuation, we say that ξ, ρ* fulfills *Γ, notation $\xi, \rho \models \Gamma$, if for all $x \in \mathrm{Var}^\star$ and $\alpha \in \mathrm{Var}^{\mathsf{Kind}}$, $x : \sigma \in \Gamma \Rightarrow \rho(x) \in [\![\sigma]\!]_{\xi\rho}$ and $\alpha : A \in \Gamma \Rightarrow \xi(\alpha) \in \mathcal{V}(A)_{\xi\rho}$.*

It is (implicit) in the definition that $\xi\rho \models \Gamma$ only if for all declarations $x{:}\sigma \in \Gamma$, $[\![\sigma]\!]_{\xi\rho}$ is defined (and similarly for $\alpha{:}A \in \Gamma$).

Definition 9. *The notion of* truth in a $\lambda P2$-model, *notation $\models^{\mathcal{M}}$ and of* truth, *notation $\models$ are defined as follows. For Γ a context, t an object, σ a type, P a constructor and A a kind of $\lambda P2$,*

$$\Gamma \models^{\mathcal{M}} t : \sigma \ \mathit{if}\, \forall \xi, \rho[\xi, \rho \models \Gamma \Rightarrow (\![t]\!)_\rho \in [\![\sigma]\!]_{\xi\rho}],$$
$$\Gamma \models^{\mathcal{M}} P : A \ \mathit{if}\, \forall \xi, \rho[\xi, \rho \models \Gamma \Rightarrow [\![P]\!]_{\xi\rho} \in \mathcal{V}(A)_{\xi\rho}].$$

Quantifying over the class of all $\lambda P2$-models, we define, for M an object or a constructor of $\lambda P2$,

$$\Gamma \models M : T \quad \mathit{if} \quad \Gamma \models^{\mathcal{M}} M : T \ \textit{for all } \lambda P2\textit{-models } \mathcal{M}.$$

Soundness states that if a judgment $\Gamma \vdash M : T$ is derivable, then it is true in all models. It is proved 'model-wise', by induction on the derivation in $\lambda P2$.

Theorem 1 (Soundness). *For Γ a context, M an object or a constructor and T a type or a kind of $\lambda P2$,*

$$\Gamma \vdash M : T \Rightarrow \Gamma \models M : T.$$

Example 3. Let $\mathcal{A}$ be a weca.

1. The *full $\lambda P2$-model* over $\mathcal{A}$ is $\mathcal{M} = \langle \mathcal{A}, \mathcal{P}, \mathcal{N} \rangle$, where $\mathcal{P}$ is the full polyset structure over $\mathcal{A}$ (as defined in Example 2).
2. The *simple $\lambda P2$-model* over $\mathcal{A}$ is $\mathcal{M} = \langle \mathcal{A}, \mathcal{P}, \mathcal{N} \rangle$, where $\mathcal{P}$ is the simple polyset structure over $\mathcal{A}$. (So $\mathcal{P} = \{\emptyset, \mathbf{A}\}$.)
3. The simple $\lambda P2$-model over the degenerate $\mathcal{A}$ is also called the *proof-irrelevance model* or *PI-model* for $\lambda P2$.
4. For C a set of constants, the *$\lambda P2$-model generated from C* is defined by $\mathcal{M} = \langle \mathbf{\Lambda}(C), \mathcal{P}, \mathcal{N} \rangle$, where $\mathcal{P}$ is the polyset structure generated from C.

4 Non-derivability results in $\lambda P2$

We now show that the induction-principle is not derivable in $\lambda P2$ by constructing a counter-model. We first introduce some notation and then we study some specific models and their properties.

In a logical model, validity of a formula φ means that the interpretation of φ is true in the model. In a type theoretical model, we call a type *valid* if its interpretation is nonempty. This conforms with the 'formulas-as-types' embedding from PRED2 to $\lambda P2$, where a formula is interpreted as the type of its proofs. (Hence, a formula is provable iff its associated type is nonempty.)

Definition 10. *For $\mathcal{M}$ a $\lambda P2$-model, Γ a context, σ a type in Γ and ξ, ρ valuations such that $\xi, \rho \models \Gamma$, we say that σ* is valid *in $\mathcal{M}$ under ξ, ρ, notation $\mathcal{M}, \xi, \rho \Vdash^{\lambda P2} \sigma$, if*

$$[\![\sigma]\!]^{\mathcal{M}}_{\xi\rho} \neq \emptyset.$$

In case the model $\mathcal{M}$ is clear from the context, we omit it. Similarly we omit ξ and/or ρ if they are clear from the context or if the specific choice of ξ or ρ is irrelevant (e.g. in case of a closed type σ).

So, to prove the non-derivability of ind in $\lambda P2$, we are looking for a $\lambda P2$-model $\mathcal{M}$ such that

$$\mathcal{M} \not\Vdash^{\lambda P2} \mathsf{ind}.$$

Definition 11. *A $\lambda P2$-model $\mathcal{M}$ is* consistent *if $\emptyset \in \mathcal{P}$.*

For a $\lambda P2$-model, being consistent is equivalent to saying that $[\![\bot]\!] = \emptyset$, because $[\![\bot]\!]$ is the minimal element (w.r.t. $\subseteq$) of $\mathcal{P}$. Here, $\bot$ is defined as usual as $\Pi\alpha{:}\star.\alpha$.

Note that the polyset structures of Example 2 all yield a consistent $\lambda P2$-model.

Convention 12 *From now on we only discuss consistent $\lambda P2$-models.*

Definition 13. *In a $\lambda P2$-model $\mathcal{M} = \langle \mathcal{A}, \mathcal{P}, \mathcal{N}\rangle$ we define the 'connectives' $\bot$, $\neg$, $\wedge$, $\vee$ and $\exists$ as follows. ($X, Y \in \mathcal{P}$, $F : X \to \mathcal{P}$ and $Y_i \in \mathcal{P}$ for all $i \in I$; as in types, we let brackets associate to the right.)*

$$\begin{array}{rlrl} \bot &:= \bigcap_{Z\in\mathcal{P}} Z, & \neg X &:= X \to \bot, \\ X \wedge Y &:= \bigcap_{Z\in\mathcal{P}} (X\to Y\to Z)\to Z, & X \vee Y &:= \bigcap_{Z\in\mathcal{P}} (X\to Z)\to(Y\to Z)\to Z, \\ \exists_{x\in X} F(x) &:= \bigcap_{Z\in\mathcal{P}} (\Pi_{x\in X} F(x)\to Z)\to Z, & \exists_{i\in I} Y_i &:= \bigcap_{Z\in\mathcal{P}} (\bigcap_{i\in I} Y_i \to Z)\to Z. \end{array}$$

Note that, due to the assumptions on a polyset structure, these are all elements of $\mathcal{P}$.

Remark 1. The definition of $\exists_{i\in I} Y_i$ is close to the *union*. If we define the elements F and G of the weca $\mathbf{A}$ by $F := \lambda^* x.xI$ and $G := \lambda^* xh.hx$ (where I denotes the identity in $\mathbf{A}$: $I := \mathbf{skk}$), then $F \in \exists_{i\in I} Y_i \to \bigcup_{i\in I} Y_i$ and $G \in \bigcup_{i\in I} Y_i \to \exists_{i\in I} Y_i$

even with $F \circ G = I$[2]. Note however, that $\bigcup_{i \in I} Y_i$ need not be an element of $\mathcal{P}$[3], but we do have $\exists_{i \in I} Y_i = \emptyset \Leftrightarrow \bigcup_{i \in I} Y_i = \emptyset$.

Lemma 1. *The following holds in arbitrary (consistent) $\lambda P2$-models $\mathcal{M}$.*

$$\neg X = \emptyset \Leftrightarrow X \neq \emptyset, \tag{1}$$
$$X \to Y \neq \emptyset \Leftrightarrow \textit{if } X \neq \emptyset \textit{ then } Y \neq \emptyset, \tag{2}$$
$$X \wedge Y \neq \emptyset \Leftrightarrow X \neq \emptyset \textit{ and } Y \neq \emptyset, \tag{3}$$
$$X \vee Y \neq \emptyset \Leftrightarrow X \neq \emptyset \textit{ or } Y \neq \emptyset, \tag{4}$$
$$\exists_{x \in X} F(x) \neq \emptyset \Leftrightarrow \exists t \in X(F(t) \neq \emptyset), \tag{5}$$
$$\exists_{i \in I} Y_i \neq \emptyset \Leftrightarrow \exists i \in I(Y_i \neq \emptyset), \tag{6}$$
$$\Pi_{x \in X} F(x) \neq \emptyset \Rightarrow \forall t \in X(F(t) \neq \emptyset), \tag{7}$$
$$\bigcap_{i \in I} Y_i \neq \emptyset \Rightarrow \forall i \in I(Y_i \neq \emptyset). \tag{8}$$

Proof. We reason classically in the meta-theory of the models (otherwise $\Leftarrow$ in (2) and $\Rightarrow$ in (4)-(6) are problematic).
(1) follows immediately from $\perp = \emptyset$ (i.e. the consistency of the $\lambda P2$-model).
For (2), $\Rightarrow$ is immediate. For $\Leftarrow$, we distinguish cases: if $X \neq \emptyset$, then $Y \neq \emptyset$, say $q \in Y$, and hence $\lambda^* x.q \in X \to Y$; if $X = \emptyset$, then $\lambda^* x.x \in X \to Y$. For (3), $\Rightarrow$: $M \in X \wedge Y$, then $M\mathbf{k} \in X$ and $M(\mathbf{ki}) \in Y$ (where $\mathbf{i}$ is the identity in the weca, $\mathbf{i} := \mathbf{skk}$). $\Leftarrow$: if $M_1 \in X$, $M_2 \in Y$, then $\lambda^* h.hM_1M_2 \in X \wedge Y$.
For (4), $\Rightarrow$: let $M \in X \vee Y$ and suppose $X = Y = \emptyset$. Then $Maa \in \emptyset$ ($a \in \mathcal{A}$ arbitrary), contradiction. So $X \neq \emptyset$ or $Y \neq \emptyset$ $\Leftarrow$: if $M \in X$, then $\lambda^* hg.hM \in X \vee Y$ and similarly for $M \in Y$.
For (5), $\Rightarrow$: let $M \in \exists_{x \in X} F(x)$ and suppose $\forall x \in X(F(x) = \emptyset)$. Then $M(\lambda^* x.\lambda^* y.y) \in \emptyset$, contradiction, so $\exists x \in X(F(x) \neq \emptyset)$. $\Leftarrow$: If $q \in F(t)$ for certain $t \in X$, then $\lambda^* h.htq \in \exists_{x \in X} F(x)$.
(6) follows from Remark 1 and (7) and (8) are immediate.

Remark 2. The reverse implications in Lemma 1, cases (7) and (8), do not hold in general. A counterexample can be found by looking at the full polyset structure over $\mathbf{A} = \mathbf{\Lambda}$. Define $F : \mathbf{A} \to \mathcal{P}$ by $F(t) = \mathbf{\Lambda} \setminus \{t\}$. Then $F(t) \neq \emptyset$ for all $t \in \mathbf{\Lambda}$. Now suppose $M \in \Pi_{x \in X} F(x)$. Then $Mt \neq t$ for all $t \in \mathbf{\Lambda}$, but this is not possible, since M has a fixed point. This contradicts the reverse implication of (7). If we consider $\bigcap_{x \in \mathbf{A}} F(x)$, we immediately find a counterexample to the reverse implication of (8).

Lemma 2. *For a simple $\lambda P2$-model over $\mathcal{A}$ the reverse implications in Lemma 1, cases (7) and (8), hold. Similarly for a $\lambda P2$-model generated from a set C.*

[2] In a weca $\mathbf{A}$, composition is defined as usual by $a \circ b := \lambda^* x.a \cdot (b \cdot x)$.
[3] The example $\mathcal{P}$s of Example 2 are all closed under arbitrary union and at this moment we don 't know of any $\mathcal{P}$ that is *not* closed under unions. However, Definition 2 does not a priori require a $\mathcal{P}$ to be closed under union.

Proof. Case (8) is immediate: $\bigcap_{i\in I} Y_i$ can only be empty if one of the Y_i is empty. For (7), if for all $t \in X$, $Ft \neq \emptyset$, then there is an element q such that $\forall t \in X(q \in Ft)$ (this is a peculiar feature of these models) and hence $\lambda^* x.q \in \Pi_{t\in X} Ft$.

Lemma 3. *All $\lambda P2$-models satisfy classical logic, i.e.*

$$\neg\neg X \to X \neq \emptyset$$

for all $X \in \mathcal{P}$ in all $\lambda P2$-models.

Proof. We reason classically in the models, using Lemma 1. Let $X \in \mathcal{P}$. If $X \neq \emptyset$, say $t \in X$, then $\neg\neg X \to X \neq \emptyset$, because e.g. $\lambda^* x.t \in \neg\neg X \to X$. If $X = \emptyset$, then $\neg X = \mathbf{\Lambda}$, so $\neg\neg X = \emptyset$, so $\neg\neg X \to X = \mathbf{\Lambda}$.

Remark 3. It is not the case that $\cap_{X\in\mathcal{P}} \neg\neg X \to X \neq \emptyset$ in all $\lambda P2$-models. In fact we have the following.

1. In the full $\lambda P2$-model over $\mathbf{\Lambda}$, $\cap_{X\in\mathcal{P}} \neg\neg X \to X = \emptyset$.
2. In simple $\lambda P2$-models or models generated by some C, $\cap_{X\in\mathcal{P}} \neg\neg X \to X \neq \emptyset$.

The first is proved by defining $X_i = \{x_i\}$ for all $i \in \mathbb{N}$ (with, of course all x_i different). Then $\neg\neg X_i = \mathbf{\Lambda}$. Now, suppose $M \in \cap_{X\in\mathcal{P}} \neg\neg X \to X$. Then for any $N \in \mathbf{\Lambda}$, we find that $\forall i \in \mathbb{N}(MN \in X_i)$, i.e. $MN =_\beta x_i$ for all i, which is not possible, as MN contains only finitely many free variables.
The second is proved by noticing that, in these models there is an element P such that $X \neq \emptyset \Rightarrow P \in X$. Hence $\lambda^* x.P \in \cap_{X\in\mathcal{P}} \neg\neg X \to X$, following the reasoning in the proof of Lemma 3.

Equality is defined in $\lambda P2$ using Leibniz equality: for $\sigma : \star$, $M, N : \sigma$

$$M =_\sigma N \quad := \quad \Pi P{:}\sigma \to \star.(PM) \to (PN).$$

In case the type is clear from the context, we often do not write it as a subscript in the Leibniz equality. The notion of 'Proof-Irrelevance', meaning that for any type σ, all terms of type σ are equal, is defined by PI $:= \Pi\alpha{:}\star.\Pi x, y{:}\alpha.x =_\alpha y$.

Lemma 4. *Given a $\lambda P2$-model $\mathcal{M}$, a type σ and terms $M, N : \sigma$, we have*

$$\mathcal{M}, \xi, \rho \models M =_\sigma N \Leftrightarrow (\![M]\!)_\rho = (\![N]\!)_\rho.$$

Proof. $\Rightarrow$: Suppose $\cap_{Q\in[\![\sigma]\!]\to\mathcal{P}} Q(\![M]\!)_\rho \to Q(\![N]\!)_\rho \neq \emptyset$. Take Q such that $Qx \neq \emptyset$ iff $x = (\![M]\!)_\rho$ Then it is the case that $Q(\![N]\!)_\rho \neq \emptyset$, hence $(\![M]\!)_\rho = (\![N]\!)_\rho$.
$\Leftarrow$: If $(\![M]\!)_\rho = (\![N]\!)_\rho$, then $Q(\![M]\!)_\rho = Q(\![N]\!)_\rho$, so $\lambda^* x.x \in \cap_{Q\in[\![\sigma]\!]\to\mathcal{P}} Q(\![M]\!)_\rho \to Q(\![N]\!)_\rho$.

Corollary 1. $\mathcal{M} \models PI \Leftrightarrow \mathcal{M}$ *is the PI-model.*

In this paper we focus especially on the induction principle for (an arbitrary encoding of) the natural numbers. We therefore characterize when a $\lambda P2$-model satisfies induction for the natural numbers.

Definition 14. *Given a closed $\lambda P2$-type N and closed terms $0 : N$ and $S : N {\to} N$, we define the type* $\mathsf{ind}_{N,0,S}$ *by*

$$\Pi P{:}N{\to}\star.P0{\to}(\Pi x{:}N.Px{\to}P(Sx)){\to}\Pi x{:}N.Px.$$

Lemma 5. *For $\mathcal{M} = \langle \mathcal{A}, \mathcal{P}, \mathcal{N} \rangle$ a $\lambda P2$-model,*

$$\mathcal{M} \models \mathsf{ind}_{N,0,S} \;\Rightarrow\; [\![N]\!] = \{S^n 0 \mid n \in \mathbb{N}\}$$

If, moreover, the test-for-zero and the predecessor function are definable on the type N in the model $\mathcal{M}$, then also

$$[\![N]\!] = \{S^n 0 \mid n \in \mathbb{N}\} \;\Rightarrow \mathcal{M} \;\models \mathsf{ind}_{N,0,S}.$$

Proof. For simplicity, we denote the interpretations of N, 0 and S in the model just by N, 0 and S. Suppose $\mathcal{M} \models \mathsf{ind}_{N,0,S}$. Then

$$\bigcap_{Q \in N \to \mathcal{P}} Q0{\to}(\Pi_{t \in N} Qt{\to}Q(St)){\to}\Pi_{t\in N} Qt \;\neq\; \emptyset.$$

Let X be some non-empty element of $\mathcal{P}$. Define $Q : N{\to}\mathcal{P}$ as follows: $Qt = X$ if $t = S^n 0$ for some $n \in \mathbb{N}$ and $Qt = \emptyset$ otherwise. Then $Q0 \neq \emptyset$ and $\Pi_{t\in N} Qt{\to}Q(St) \neq \emptyset$, hence $\Pi_{t\in N} Qt \neq \emptyset$, say $M \in \Pi_{t\in N} Qt$. Now, suppose $q \in N$ with $q \neq S^n 0$ (for all $n \in \mathbb{N}$). Then $Qq = \emptyset$ but also $Mq \in Qq$, contradiction. So all $q \in N$ are of the form $S^n 0$.

For the reverse implication, suppose that the test-for-zero and the predecessor function are definable in the model and suppose that $N = \{S^n 0 \mid n \in \mathbb{N}\}$. To prove that $\bigcap_{Q \in N \to \mathcal{P}} Q0{\to}(\Pi_{t\in N} Qt{\to}Q(St)){\to}\Pi_{t\in N} Qt \neq \emptyset$, let $Q \in N{\to}\mathcal{P}$ arbitrary and let $Z \in Q0$, $F \in \Pi_{t\in N} Qt{\to}Q(St)$. We are looking for an element of $\Pi_{t\in N} Qt$, which is given by an H which is a solution to

$$Hx = \text{if Zero}(x) \text{ then } Z \text{ else } F(x-1)(H(x-1)).$$

This can be obtained by taking for H a fixed point of $\lambda^* hx.$if Zero(x) then Z else $F(x-1)(h(x-1))$. Note that we need the test-for-zero and predecessor to be able to define this H.

Theorem 2. *Induction over the natural numbers is not derivable in $\lambda P2$ for any type N and terms $0 : N$, $S : N{\to}N$.*

Proof. In the simple $\lambda P2$-model over $\mathbf{\Lambda}$ (see Example 3), the interpretation of N is $\mathbf{\Lambda}$. So, using the Lemma, we conclude that $\mathsf{ind}_{N,0,S}$ is not valid in the model and hence $\mathsf{ind}_{N,0,S}$ is not inhabited in $\lambda P2$.

As can be observed from the proof, the non-derivability of induction in $\lambda P2$ is not caused by the fact that the logic of $\lambda P2$ is constructive. Note that, taking the PI-model in the proof of the Theorem does not work, because then $[\![N]\!] = 1 = \{S^n 0 \mid n \in \mathbb{N}\}$, so we do not obtain a counterexample.

The arguments of Lemma 5 and Theorem 2 also apply to other data types like lists and trees and even to a finite data type like the booleans. So, induction is not derivable for any data type.

Remark 4. It is in general not the case in $\lambda P2$ that the induction principle for one data type (say the natural numbers) implies the induction principle for another data type (say booleans). For a counterexample consider the context $\Gamma = N : \star, 0 : N, S : N\to N, h : \mathsf{ind}_{N,0,S}$ and the $\lambda P2$-model $\langle \mathbf{\Lambda}(C), \mathcal{P}, \mathcal{N}\rangle$, where $C = \{S^n(0) \mid n \in \mathbb{N}\}$ (so the $S^n(0)$ are considered as constants) and $\mathcal{P}$ is the polyset structure generated from C. (See Example 2.)

Now, take valuations ξ and ρ with $\xi(N) = C$, $\rho(0) = 0$, $\rho(S) = S$ and $\rho(h) = \lambda^* zfx.0$. Then $\rho(h) \in [\![\mathsf{ind}_{N,0,S}]\!]_{\xi\rho}$:

$$\lambda^* zfx.0 \in \bigcap_{Q\in C\to\mathcal{P}} Q0\to(\Pi_{t\in C}Qt\to Q(St))\to\Pi_{t\in C}Qt,$$

because for $Q \in C\to\mathcal{P}$, $Z \in Q0$, $G \in \Pi_{t\in C}Qt\to Q(St)$ and $t \in C$, we find that $t = S^n(0)$ (def of C) and for all $n \in \mathbb{N}$, $Q(S^n(0)) \neq \emptyset$ (induction on n, using Z and G), so $0 \in Qt$. We conclude that $\xi, \rho \models \Gamma$.

So, $\mathcal{M}, \xi, \rho \models\!\models \mathsf{ind}_{N,0,S}$. On the other hand, for any closed type B (the 'booleans') with closed terms $T : B$ and $F : B$, $[\![B]\!] \supsetneq \{(\![F]\!), (\![T]\!)\}$, so induction over booleans is not valid.

One may wonder what happens with the counterexample in the proof of Theorem 2 if we add induction over natural numbers to $\lambda P2$ as a primitive concept, together with the associated reduction rules. Let's take a closer look at this situation.

We extend $\lambda P2$ with a type constant N and term constants $0 : N$, $S : N\to N$, $R : \Pi P{:}N\to\star.(P0)\to(\Pi y{:}N.Py\to P(Sy))\to\Pi x{:}N.(Px)$. Furthermore we add reduction rules

$$RPzf0 \longrightarrow_r z \quad \text{and} \quad RPzf(Sx) \longrightarrow_r fx(RPzfx).$$

To make a model of this extension of $\lambda P2$ we have to give an interpretation to the constants in such a way that the equality rule for R is preserved. For $\mathbf{\Lambda}$ (that we used in the counter-model of 2), this can be achieved by adding primitive constants 0, S and R to Λ, with the reduction rules

$$Rzf0 \longrightarrow_r z \quad \text{and} \quad Rzf(Sx) \longrightarrow_r fx(Rzfx).$$

Let's denote this extension of λ-calculus (it is a weca) by $\mathbf{\Lambda}^+$. (So we interpret 0 by 0, S by S and R by R.) Now consider the simple $\mathbf{\Lambda}^+$-model determined by the polyset structure $\{\emptyset, \mathbf{\Lambda}\}$ and notice that it is *not* a model of this $\lambda P2$ extension, because $\mathsf{ind}_{N,0,S}$ is empty in this model (so we can not interpret R).

We give one more non-derivability result in $\lambda P2$, based on our models.

Lemma 6. *There are closed types σ, τ and a relation $R : \sigma\to\tau\to\star$ in $\lambda P2$ for which the Axiom of Choice, $(\Pi x{:}\sigma.\exists y{:}\tau.Rxy)\to(\exists f{:}\sigma\to\tau.\Pi x{:}\sigma.Rx(fx))$, is not derivable.*

Proof. The counterexample is similar to the one in Remark 2. Take $\sigma = \tau = \mathsf{nat}$ and $Rxy := x \neq_{\mathsf{nat}} y$ and consider the simple $\lambda P2$-model over $\mathbf{A} = \mathbf{\Lambda}$. Now

$\mathcal{M} \models \Pi x{:}\sigma.\exists y{:}\tau.Rxy$, because this is equivalent to (using Lemmas 1 and 4) $\forall t \in \mathbf{\Lambda} \exists q \in \mathbf{\Lambda}(t \neq_\beta q)$. On the other hand, $\mathcal{M} \not\models \exists f{:}\sigma{\to}\tau.\Pi x{:}\sigma.Rx(fx)$, because this is equivalent to the statement $\exists g \in \mathbf{\Lambda} \forall t \in \mathbf{\Lambda}(gt \neq_\beta t)$, which is not possible, because every element of $\mathbf{\Lambda}$ has a fixed point.

The proof of non-derivability of the Axiom of Choice bears a strong similarity to a proof in [Barendregt 1973], credited originally to Scott, showing that classical Combinatory Logic extended with the Axiom of Choice is inconsistent.

Acknowledgments Thanks to the referees for pointing out some mistakes in the original manuscripts and suggesting several improvements. Furthermore I want to thank Thierry Coquand for raising the question of derivability of induction in $\lambda P2$ and for some valuable discussions on the topic.

References

[Barendregt 1973] H.P. Barendregt, Combinatory Logic and the Axiom of Choice, in *Indagationes Mathematicae*, vol. 35, nr. 3, pp. 203 – 221.

[Barendregt 1992] H.P. Barendregt, Typed lambda calculi. In *Handbook of Logic in Computer Science*, eds. Abramski et al., Oxford Univ. Press.

[Berardi 1993] S. Berardi, Encoding of data types in Pure Construction Calculus: a semantic justification, in *Logical Environments*, eds. G. Huet and G. Plotkin, Cambridge University Press, pp 30–60.

[Coquand 1990] Th. Coquand, Metamathematical investigations of a calculus of constructions. In *Logic and Computer Science*, ed. P.G. Odifreddi, APIC series, vol. 31, Academic Press, pp 91-122.

[Coquand and Huet 1988] Th. Coquand and G. Huet, The calculus of constructions, *Information and Computation*, 76, pp 95-120.

[Geuvers 1993] J.H. Geuvers, *Logics and Type systems*, Ph.D. Thesis, University of Nijmegen, Netherlands.

[Geuvers 1996] J.H. Geuvers, Extending models of second order logic to models of second order dependent type theory, *Computer Science Logic*, Utrecht, eds. D. van Dalen and M. Bezem, LNCS 1258, 1997, pp 167–181.

[Hyland and Ong 1993] J.M.E. Hyland and C.-H. L. Ong, Modified realizability toposes and strong normalization proofs. In *Typed Lambda Calculi and Applications*, Proceedings, eds. M. Bezem and J.F. Groote, LNCS 664, pp. 179–194, Springer-Verlag, 1993.

[Girard et al. 1989] J.-Y. Girard, Y. Lafont and P. Taylor, *Proofs and types*, Camb. Tracts in Theoretical Computer Science 7, Cambridge University Press.

[Longo and Moggi 1988] G. Longo and E. Moggi, Constructive Natural Deduction and its "Modest" Interpretation. Report CMU-CS-88-131.

[Stefanova and Geuvers 1996] M. Stefanova and J.H. Geuvers, A Simple Model Construction for the Calculus of Constructions, in *Types for Proofs and Programs*, Int. Workshop, Torino, eds. S. Berardi and M. Coppo, LNCS 1158, 1996, pp. 249–264.

[Streicher 1991] T. Streicher, Independence of the induction principle and the axiom of choice in the pure calculus of constructions, *TCS* 103(2), pp 395 - 409.