A Type-Theoretic Approach to Induction with Higher-Order Encodings

Carsten Schürmann

Yale University
carsten@cs.yale.edu

Abstract. Reasoning by induction is common practice in computer science and mathematics. In formal logic, however, standard induction principles exist only for a certain class of inductively defined structures that satisfy the positivity condition. This is a major restriction considering that many structures in programming languages and logics are best expressed using higher-order representation techniques that violate exactly this condition. In this paper we develop induction principles for higher-order encodings in the setting of first-order intuitionistic logic. They differ from standard induction principles in that they rely on the concept of worlds [Sch01] which admits reasoning about open terms in regularly formed contexts. The soundness of these induction principles follows from external termination and coverage considerations about a realizability interpretation of proofs.

1 Introduction

Reasoning by induction is common practice in computer science and mathematics, it is well understood, and it is admissible when reasoning about objects whose types are inductively defined. But standard induction principles only exist for those types whose constructors satisfy the so called positivity condition [PM93,SH93]. This condition requires that the type that is being defined does not occur in a negative position in any of its constructor types. Types that violate this condition are not inductive and therefore excluded from inductive reasoning.

Many concepts however that are prevalent in programming languages, logics, and type theory such as for example operational semantics, compilers, transformations, logics, and proof systems, have elegant, dependently typed, higher-order encodings (see [Pfe99] for an overview) but violate this positivity condition. These encodings are therefore not inductive in the standard sense, which hampers their use in programming and renders them inappropriate for modeling in automated proof assistants based on standard induction principles, such as, for example, Coq [DFH+93] or Isabelle/HOL [Pau94].

In previous years, this problem has been actively worked on, and much progress has been made [DPS97,Hof99,GP99,HMS01] with a variety of ideas inspired from modal logic, category theory, FM set theory, and the π-calculus. In this paper we propose a type theoretic solution to this problem. Specifically, we define an induction principle for any higher-order encoding in the logical framework LF. Our technique is based on the observation that standard induction principles leverage off on the existence of canonical (β-normal η-long) forms. For example, in totally freely generated term algebras, canonical forms of first-order encodings are guaranteed to exist if only closed objects are being considered.

Canonical forms also exists in the higher-order setting and they are inductively defined. This makes them great candidates for inductive reasoning, however they have one serious problem. Canonical forms are in general open, i.e. they may contain free parameters, which is common to all higher-order encodings. If we include these parameters in the inductive reasoning process we obtain the sound induction principle described in this paper that is based on the idea of distinguishing cases of canonical forms. It is expressive enough to reason about properties of logics, operational semantics, and abstract machines. The underlying assumption is called the *regular world assumption* [Sch01] which allows quantifiers to range over open objects whose free variables are declared in a *regularly* formed (and hence subject to reasoning) context. Those regular contexts are characterized by so called *worlds* whose structure may be freely chosen, but must be fixed ahead of time [Sch00].

The main contribution of this paper is an induction principle for higher-order encodings that extends first-order intuitionistic logic. Its quantifiers range over objects which live in the dependent type theory LF [HHP93] for which canonical forms are known to exist. A prototype theorem prover for this logic has been implemented in the Twelf system [PS99].

This paper is organized as follows. In Section 2 we revisit standard induction principles for first-order encodings and give in Section 3 a brief overview over higher-order encodings. In Section 4 we then motivate and define our induction principle. In Section 5 we argue for the soundness of the design before we discuss related work in Section 6. We comment on the implementation and assess results and future work in Section 7.

2 Standard Induction Principles

One of the most intuitive forms of an induction principle is the one for natural numbers. But first, how shall we represent natural numbers? In this paper we represent all data structures uniformly in a logical framework. The logical framework of choice is LF [HHP93], but for now, the reader is invited to think of it as the simply typed λ-calculus. The representation of natural numbers leads to the following signature Σ of constant declarations:

$$\begin{aligned}\Sigma = \ & \mathsf{nat} : \text{type}, \\ & \mathsf{z} \quad : \mathsf{nat}, \\ & \mathsf{s} \quad : \mathsf{nat} \to \mathsf{nat}.\end{aligned}$$

The '$\to$' type constructor denotes the function space of the logical framework.

Standard induction principles, which are commonly used in proof assistants and automated theorem proving systems, require that the respective datatypes

$$\frac{}{\Psi \Vdash_\Sigma \top}\,\top\mathsf{I} \qquad \frac{\Psi, x : A \Vdash_\Sigma \mathcal{P}(x)}{\Psi \Vdash_\Sigma \forall x : A.\mathcal{P}(x)}\,\forall\mathsf{I}^x \qquad \frac{\Psi \Vdash_\Sigma \forall x : A.\mathcal{P}(x) \qquad \overline{\Psi} \vdash_\Sigma M \Uparrow A}{\Psi \Vdash_\Sigma \mathcal{P}(M)}\,\forall\mathsf{E}$$

$$\frac{\overline{\Psi} \vdash_\Sigma M \Uparrow A \qquad \Psi \Vdash_\Sigma \mathcal{P}(M)}{\Psi \Vdash_\Sigma \exists x : A.\mathcal{P}(x)}\,\exists\mathsf{I} \qquad \frac{\Psi \Vdash_\Sigma \exists x : A.\mathcal{P}(x) \qquad \Psi, x : A, \mathbf{x} \in \mathcal{P}(x) \Vdash_\Sigma \mathcal{P}'}{\Psi \Vdash_\Sigma \mathcal{P}'}\,\exists\mathsf{E}^{x,\mathbf{x}}$$

Fig. 1. Intuitionistic first-order logic $\mathcal{IL}$.

are inductive. They satisfy the "positivity condition" [PM93] which requires that the type of each constructor contains only positive occurrences of the type that is defined. Because z and s satisfy this condition, nat is inductive and a standard induction principle exists. This principle can be safely added to a logic, such as, for example, first-order intuitionistic logic $\mathcal{IL}$ whose first-order fragment is depicted in Figure 1.

$$\begin{array}{ll}\text{Formulas: } \mathcal{P} ::= \top \mid \forall x : A.\mathcal{P} \mid \exists x : A.\mathcal{P} \\ \text{Context: } \;\; \Psi ::= \cdot \mid \Psi, x : A \mid \Psi, \mathbf{x} \in \mathcal{P}\end{array}$$

We write $\Psi \Vdash_\Sigma \mathcal{P}$ for the logical entailment relation, where we use the symbol $\Vdash$ in order to distinguish it cleanly from the symbol $\vdash$ used by the logical framework. Ψ makes the list of term parameters $x : A$ (introduced by $\forall\mathsf{I}$ and $\exists\mathsf{E}$) and logical assumptions $\mathbf{x} \in \mathcal{P}$ (introduced by $\exists\mathsf{E}$) explicit. Objects that are substituted for parameters must be valid in the logical framework, which is indicated by the left premiss of $\forall\mathsf{E}$ and $\exists\mathsf{I}$. We write $\overline{\Psi}$ for the result of mapping a meta level context down to the logical framework level (by removing all declarations of the form $\mathbf{x} \in \mathcal{P}$). The induction principle for natural numbers is standard

$$\frac{\Psi \Vdash_\Sigma \mathcal{P}(\mathsf{z}) \qquad \Psi, n' : \mathsf{nat}, \mathbf{x} \in \mathcal{P}(n') \Vdash_\Sigma \mathcal{P}(\mathsf{s}\, n')}{\Psi, n : \mathsf{nat} \Vdash_\Sigma \mathcal{P}(n)}\,\mathsf{ind\_nat}^{n',\mathbf{x}}.$$

$\mathcal{P}(n)$ or $\Psi(n)$ indicate that n can occur freely in $\mathcal{P}$ or Ψ, respectively. $\mathsf{ind\_nat}$, $\forall\mathsf{I}$, and $\exists\mathsf{E}$ discharge assumptions that are listed with the rule name.

The logical framework LF [HHP93] extends the simply typed λ-calculus by dependent types. We write $\Gamma \vdash_\Sigma M : A$ for the LF typing judgment where Γ is the standard LF context, M is an object and A its type. Every object in LF has a canonical (β-normal η-long) form for which we write $\Gamma \vdash_\Sigma M \Uparrow A$. LF's formulation is standard and we take $\beta\eta$-conversion as the notion of definitional equality [HHP93,Coq91]. An induction principle for natural numbers in the LF setting is slightly more complicated then the one above, because dependencies among the declarations in Ψ must be respected.

$$\frac{\begin{array}{l}\Psi_1, \Psi_2(\mathsf{z}) \Vdash_\Sigma \mathcal{P}(\mathsf{z}) \\ \Psi_1, n' : \mathsf{nat}, \mathbf{x} \in \mathcal{P}(n'), \Psi_2(\mathsf{s}\, n') \Vdash_\Sigma \mathcal{P}(\mathsf{s}\, n')\end{array}}{\Psi_1, n : \mathsf{nat}, \Psi_2(n) \Vdash_\Sigma \mathcal{P}(n)}\,\mathsf{ind\_nat}^{n',\mathbf{x}}. \tag{1}$$

$$\frac{}{\Delta, G \vdash^{ND} G} \qquad \frac{\Delta, G_1 \vdash^{ND} G_2}{\Delta \vdash^{ND} G_1 \supset G_2} \supset \mathsf{I} \qquad \frac{\Delta \vdash^{ND} G_1 \supset G_2 \qquad \Delta \vdash^{ND} G_1}{\Delta \vdash^{ND} G_2} \supset \mathsf{E}$$

Fig. 2. Natural deduction calculus

3 Higher-order Encodings

Our approach to induction principles scales to arbitrary higher-order representations. In fact it scales to any deductive system that can be represented in LF. Our running example is the natural deduction calculus [Gen35], which is depicted in Figure 2. The judgment is hypothetical, and its encoding uses binding constructs to represent the hypotheses from Δ.

$$\textit{Hypotheses: } \Delta ::= \cdot \mid \Delta, G \tag{2}$$

Although we restrict this presentation to the implicational fragment of the natural deduction calculus, all technology developed in this paper scales to full first-order logic. Formulas $G_1 \supset G_2$ are represented in LF as $\ulcorner G_1 \urcorner \supset \ulcorner G_2 \urcorner$ where $\ulcorner \cdot \urcorner$ denotes the standard representation function. We write $\llcorner \cdot \lrcorner$ for its inverse whose existence we can assume. Consult [HHP93] for an in depth coverage of this example.

Example 1 (Natural Deduction Calculus).

$$\begin{aligned}
\Sigma = \mathsf{o} \quad & : \text{type}, \\
\supset \quad & : \mathsf{o} \to (\mathsf{o} \to \mathsf{o}), \\
\mathsf{nd} \quad & : \mathsf{o} \to \text{type}, \\
\mathsf{impI} \quad & : (\mathsf{nd}\ G_1 \to \mathsf{nd}\ G_2) \to \mathsf{nd}\ (G_1 \supset G_2), \\
\mathsf{impE} \quad & : \mathsf{nd}\ (G_1 \supset G_2) \to \mathsf{nd}\ G_1 \to \mathsf{nd}\ G_2.
\end{aligned}$$

Following standard practice [Pfe91], we omit in this presentation all implicit Π-abstractions from types.

When working with complex encodings such as the one of a proof calculus, the first question that should come to mind is if the representation is *adequate*, i.e. if natural deduction derivations are in one to one correspondence with canonical LF objects of appropriate type. Adequacy is proven by induction. In the case of natural deductions however, special care must be taken in formulating the induction hypothesis of the adequacy theorem. Hypothetical judgments are encoded as higher-order functions and consequently already the formulation of the theorem must establish the connection between free variables (or parameters) in an LF encoding and valid hypotheses in the natural deduction calculus.

Therefore adequacy is a property of open objects, and not just of closed objects as in the case of natural numbers. Here is a first attempt to formulate the adequacy property: For any canonical LF object $\Gamma \vdash_{\Sigma} M \Uparrow \mathsf{nd}\ \ulcorner G \urcorner$ there exists a *unique* derivation $\mathcal{D}$ of judgment $\Delta \vdash^{ND} G$ (written as $\mathcal{D} :: \Delta \vdash^{ND} G$), such that $M = \ulcorner \mathcal{D} \urcorner$ and $\Gamma = \ulcorner \Delta \urcorner$ — and vice versa.

Also for hypothetical judgments, which are represented as open objects, $\ulcorner \cdot \urcorner$ must be a bijection. Clearly, Δ can be mapped directly to Γ, but the reverse does not hold in general. Not every Γ corresponds to a Δ as $\Gamma = u_1 : \ulcorner G_1 \urcorner, \ldots, u_n : \ulcorner G_n \urcorner, A : \mathsf{o}$ shows. Consequently, in the interest of adequacy, we must restrict the set of Γ's to those that are structurally constructed the same way Δ is.

In fact, without loss of generality, we can always assume that Δ is a regularly formed list, and we propose a formal way of characterizing valid contexts (which we call *regular* for this reason). We write $\Gamma \in \mathcal{L}(\Phi)$ if Γ is an instance of *world* Φ, the "type of all valid contexts". With $\mathcal{L}(\Phi)$ we refer to the set of all regular contexts generated by world Φ.

$$\text{Worlds: } \Phi ::= L : \text{some } \Gamma_1 \text{ block } \Gamma_2 \mid \Phi + \Phi \mid \Phi^*$$

In our example, $\Gamma \in \mathcal{L}((L : \text{some } G : \mathsf{o} \text{ block } u : \mathsf{nd}\ G)^*)$. $\mathcal{L}(\Phi)$ is defined in Figure 3. Intuitively, worlds are regular expressions with "L : some Γ_1 block Γ_2" as terminals, where $+$ describes alternatives, and * repetition. The terminal "L : some Γ_1 block Γ_2" satisfies the invariant that Γ_1, Γ_2 form a valid context. L is used to label blocks of declarations of a context,which allows us to distinguish different blocks from each other. In context Γ_0, the set of regular contexts $\mathcal{L}(L : \text{some } \Gamma_1 \text{ block } \Gamma_2)$ consists of all α-variants of the block Γ_2 where its free variables declared in Γ_1 have been instantiated by objects (summarized as substitution σ) valid in Γ_0. We write $[\sigma]\Gamma_2$ for a context under a substitution. That σ is valid is enforced by the first premiss $\Gamma_0 \vdash \sigma : \Gamma_1$ of the "block"-rule whose definition we omit. The second premiss of the same rule is the standard α-conversion congruence $\Gamma_0 \vdash \Gamma_1 \equiv_\alpha \Gamma_2$ which permits tacit variable renaming on regular contexts. Consequently, without loss of generality all contexts in $\mathcal{L}(\Phi)$ are valid.

Worlds are the "types of valid contexts" which can also be interpreted as grammars that generate Δ, written as $\Delta \in \mathcal{L}(\llcorner \Phi \lrcorner)$. As example, compare the world $\Phi = (L : \text{some } G : \mathsf{o} \text{ block } u : \mathsf{nd}\ G)^*$ with (2).

Theorem 1 (Adequacy for worlds). *Let Φ be a world, and $\ulcorner \cdot \urcorner$ an adequate representation function for all types that occur in Φ. $\ulcorner \cdot \urcorner$ extends to a bijection between LF contexts Γ that satisfy $\cdot \vdash \Gamma \in \mathcal{L}(\Phi)$ and Δ that satisfy $\Delta \in \mathcal{L}(\llcorner \Phi \lrcorner)$.*

Theorem 2 (Adequacy of nd). *Let $\Phi = (L : \text{some } G : \mathsf{o} \text{ block } u : \mathsf{nd}\ G)^*$ and $\cdot \vdash \Gamma \in \mathcal{L}(\Phi)$. We have $\Gamma \vdash M \Uparrow \mathsf{nd}\ \ulcorner G \urcorner$ iff $M = \ulcorner \mathcal{D} \urcorner$ and $\mathcal{D} :: \llcorner \Gamma \lrcorner \vdash G$.*

Proof. By structural induction on the derivation of $\Gamma \vdash M \Uparrow \mathsf{nd}\ \ulcorner G \urcorner$ in one direction and $\mathcal{D}$ in the other.

This answers the first question. Natural deduction derivations can be adequately represented in LF. But are these representations inductive in the sense that they possess standard induction principles? Following standard definitions, they are not, because impl's type refers to nd in a negative position. However, every term in LF has a canonical form in a given world, and canonical forms are inductively defined, and therefore nd is still inductive as we elaborate on next.

$$\frac{\Gamma_0 \vdash \sigma : \Gamma_1 \qquad \Gamma_0 \vdash \Gamma \equiv_\alpha [\sigma]\Gamma_2}{\Gamma_0 \vdash \Gamma \in \mathcal{L}(L : \text{some } \Gamma_1 \text{ block } \Gamma_2)}\text{block}$$

$$\frac{}{\Gamma_0 \vdash \cdot \in \mathcal{L}(\Phi^*)}\text{empty} \qquad \frac{\Gamma_0 \vdash \Gamma_1 \in \mathcal{L}(\Phi) \qquad \Gamma_0, \Gamma_1 \vdash \Gamma_2 \in \mathcal{L}(\Phi^*)}{\Gamma_0 \vdash \Gamma_1, \Gamma_2 \in \mathcal{L}(\Phi^*)}\text{unfold}$$

$$\frac{\Gamma_0 \vdash \Gamma \in \mathcal{L}(\Phi_1)}{\Gamma_0 \vdash \Gamma \in \mathcal{L}(\Phi_1 + \Phi_2)}\text{left} \qquad \frac{\Gamma_0 \vdash \Gamma \in \mathcal{L}(\Phi_2)}{\Gamma_0 \vdash \Gamma \in \mathcal{L}(\Phi_1 + \Phi_2)}\text{right}$$

Fig. 3. Set of regular contexts generated by world Φ.

4 Higher-Order Induction Principles

Proofs by induction are common in computer science. However, not every type that can be defined inductively possesses an appropriate induction principle. Natural numbers do, lists do, trees do, essentially any data type that is defined satisfying the positivity condition [PM93] does. Higher-order representations however do not.

Recall that higher-order abstract syntax is characterized by using meta-level variables to encode object-level variables, and meta-level binders to encode object-level binders. $\supset \mathsf{I}$ introduces a new assumption, for example, which is captured by the functional argument to impl. Therefore, by design, higher-order representations may not satisfy the positivity condition. Consequently, induction on higher-order representations is fundamentally problematic.

This observation could lead one to believe that induction principles for higher-order encodings do not exist. Against this intuition speaks the observation that every LF object — functional or non-functional — has an inductively defined canonical (β-normal η-long) form. How to exploit this property to derive an induction principle for higher-order encodings is the main contribution of this paper. We demonstrate our solution by an induction principle for natural deduction derivations which is used in the proof of the following property: Every natural deduction derivation can be translated into a sequent derivation.

The sequent calculus [Gen35] defining judgment $\Delta \Longrightarrow G$ is depicted in Figure 4 is defined for the same fragment as the natural deduction calculus from Figure 2. For this example it is not so important what the sequent calculus actually is, but it is important how it is represented in LF [Pfe95].

Example 2 (Sequent deductions). Extends Σ from Example 1.

$$\begin{aligned}\Sigma = \ldots, \mathsf{hyp} &: \mathsf{o} \to \text{type},\\ \mathsf{conc} &: \mathsf{o} \to \text{type},\\ \mathsf{init} &: \mathsf{hyp}\ G \to \mathsf{conc}\ G,\\ \mathsf{impR} &: (\mathsf{hyp}\ G_1 \to \mathsf{conc}\ G_2) \to \mathsf{conc}\ (G_1 \supset G_2),\\ \mathsf{impL} &: \mathsf{conc}\ G_1 \to (\mathsf{hyp}\ G_2 \to \mathsf{conc}\ G_3) \to (\mathsf{hyp}\ (G_1 \supset G_2) \to \mathsf{conc}\ G_3),\\ \mathsf{cut} &: \mathsf{conc}\ G_1 \to (\mathsf{hyp}\ G_1 \to \mathsf{conc}\ G_2) \to \mathsf{conc}\ G_2.\end{aligned}$$

$$\frac{}{\Gamma, G \Longrightarrow G}\,\text{init} \qquad \frac{\Gamma \Longrightarrow G_1 \qquad \Gamma, G_1 \Longrightarrow G_2}{\Gamma \Longrightarrow G_2}\,\text{cut} \qquad \frac{\Gamma, G_1 \Longrightarrow G_2}{\Gamma \Longrightarrow G_1 \supset G_2}\,{\supset}\text{R}$$

$$\frac{\Gamma, G_1 \supset G_2 \Longrightarrow G_1 \qquad \Gamma, G_1 \supset G_2, G_2 \Longrightarrow G_3}{\Gamma, G_1 \supset G_2 \Longrightarrow G_3}\,{\supset}\text{L}$$

Fig. 4. Sequent calculus

Sequents $\Delta \Longrightarrow G$ are encoded by two separate type families. Hypotheses in Δ are encoded using hyp as LF contexts and G as an object of type $\mathsf{conc}\ \ulcorner G \urcorner$. The representation of the inference rules is straightforward, well understood, and adequacy can be established in the world $\Phi = (L : \text{some } G : \mathsf{o} \text{ block } h : \mathsf{hyp}\ G)^*$.

We begin now with the presentation of the central example of this paper: What is a formal proof of the following statement?

$$\forall G : \mathsf{o}.\forall D : \mathsf{nd}\ G.\exists Q : \mathsf{conc}\ G.\top \tag{3}$$

Informally, the proof of this theorem proceeds by structural induction on $\llcorner D \lrcorner :: \llcorner \mathsf{nd}\ G \lrcorner$ — it is a hypothetical proof because appeals of the induction hypothesis to the premiss of impl introduces a new nd-hypothesis. But there is a problem here! The adequacy result for natural deductions and for sequent derivations differ in what kind of declarations are permitted. For one it is $u : \mathsf{nd}\ \ulcorner G \urcorner$ and for the other $h : \mathsf{hyp}\ \ulcorner G \urcorner$. The solution involves merging both worlds into one, namely $\Phi = (L : \text{some } G : \mathsf{o} \text{ block } u : \mathsf{nd}\ G, h : \mathsf{hyp}\ G)^*$, and reproving the adequacy results for natural deduction and sequent encodings. Those proofs rely on the following two strengthen properties.

Lemma 1 (Strengthening natural deductions). *If h does not occur free in Γ' and $\Gamma, h : \mathsf{hyp}\ G_1, \Gamma' \vdash_\Sigma M \Uparrow \mathsf{nd}\ G_2$ then $\Gamma, \Gamma' \vdash_\Sigma M \Uparrow \mathsf{nd}\ G_2$.*

Lemma 2 (Strengthening sequent derivations). *If u does not occur free in Γ' and $\Gamma, u : \mathsf{nd}\ G_1, \Gamma' \vdash_\Sigma M \Uparrow \mathsf{conc}\ G_2$ then $\Gamma, \Gamma' \vdash_\Sigma M \Uparrow \mathsf{conc}\ G_2$.*

Theorem 3 (Adequacy for nd and $\mathsf{hyp}/\mathsf{conc}$). *Let $\cdot \vdash \Gamma \in \mathcal{L}(\Phi)$.*

1. *$\Gamma \vdash M \Uparrow \mathsf{nd}\ \ulcorner G \urcorner$ iff $M = \ulcorner \mathcal{D} \urcorner$ and $\mathcal{D} :: \llcorner \Gamma \lrcorner \vdash G$.*
2. *$\Gamma \vdash M \Uparrow \mathsf{conc}\ \ulcorner G \urcorner$ iff $M = \ulcorner \mathcal{D} \urcorner$ and $\mathcal{D} :: \llcorner \Gamma \lrcorner \Longrightarrow G$.*

Proof. By structural induction on the derivation of $\Gamma \vdash M \Uparrow \mathsf{nd}\ \ulcorner G \urcorner$, $\Gamma \vdash M \Uparrow \mathsf{conc}\ \ulcorner G \urcorner$ in one direction and $\mathcal{D}$ in the other, respectively, using Lemma 1 and Lemma 2.

As this example shows, worlds are as important for proving the adequacy of encodings as the signatures themselves. With this in mind, let us return to the definition of intuitionistic first-order logic presented in Figure 1 and index the judgment by $\Sigma; \Phi$ instead of just Σ. Endowing the logic with worlds has consequences on the definition of the context Ψ which we address below.

Contexts Ψ capture all assumptions that are valid in a respective state of a proof. $\mathbf{x} \in \mathcal{P}$, for example, assumes that $\mathcal{P}$ holds. Without worlds $x : A$ stands for the assumption that there exists a *closed* LF object of type A. But in the presence of worlds this setting may be generalized; we let $x : A$ stand for the assumption that there exists an *open* object that is valid in some context $\Gamma \in \mathcal{L}(\Phi)$. But how can we express assumptions about the form of Γ, such as that Γ contains certain parameters?

The answer to this question has already been proposed in [Sch00,Sch01]. A new concept of *block variables* ρ^L is required, that ranges over valid parameter blocks in accordance with the world Φ (written as $\rho^L : \Phi$). Block variables are of the form $\rho ::= \cdot \mid \rho, \underline{x} : A$. Each block variable follows the structure Γ_2 in "L : some Γ_1 block Γ_2" and is labeled with the appropriate label L. They capture assumptions about the context valid in a respective state of a proof.

$$\Psi ::= \cdot \mid \Psi, x : A \mid \Psi, [\rho]^L \mid \Psi, \mathbf{x} \in \mathcal{P}$$

Still, $\overline{\Psi}$ stands for the LF-context obtained from Ψ by removing all declarations of the form $\mathbf{x} \in \mathcal{P}$, and flattening out the individual declarations in $[\rho]^L$.

Example 3. Let $\Phi = (L : \text{some } G : \mathsf{o} \text{ block } \underline{u} : \mathsf{nd}\ G, \underline{h} : \mathsf{hyp}\ G)^*$ then $\Psi = G : \mathsf{o}, [\underline{u} : \mathsf{nd}\ G, \underline{h} : \mathsf{hyp}\ G]^L$ is a valid context.

What will happen if the induction hypothesis (3) is applied to the premiss of the impl rule $D : \mathsf{nd}\ \ulcorner G_1 \urcorner \to \mathsf{nd}\ \ulcorner G_2 \urcorner$? Informally one would assume G_1 and then apply the induction hypothesis to $\llcorner D \lrcorner$ which guarantees that there exists a sequent derivation of G_2. Formally, one first assumes $\underline{u} : \mathsf{nd}\ \ulcorner G_1 \urcorner$ and $\underline{h} : \mathsf{hyp}\ \ulcorner G_1 \urcorner$ (to be conform with Φ) followed by an appeal to the induction hypothesis to $\ulcorner G_2 \urcorner$ and $(D\ \underline{u}) : \mathsf{nd}\ \ulcorner G_2 \urcorner$. By Lemma 2 and a process called abstraction defined below we conclude

$$\Psi \Vdash_{\Sigma;\Phi} \exists x : \mathsf{hyp}\ G_1 \to \mathsf{conc}\ G_2.\top. \tag{4}$$

For the general case, dependency relations pose an answer on how to strengthen in a logical framework [Vir99]. We say that one type is dependent on another if objects of the former are build in terms of objects of the latter. For example, '$\mathsf{conc}\ G_2$' is dependent on '$\mathsf{hyp}\ G_1$' by rule init, but not '$\mathsf{nd}\ G_1$' by Lemma 2. Formally we write '$\mathsf{hyp}\ G_1 \prec \mathsf{conc}\ G_2$', but '$\mathsf{nd}\ G_1 \not\prec \mathsf{conc}\ G_2$'.

The process of incorporating assumptions of the form $x : A$ in a type or formula is called *abstraction*. Intuitively, it turns derivations of hypothetical judgments into functions while applying the a priori proven strengthening results as stringently as possible. In the example above, we write '$\nu(\underline{u} : \mathsf{nd}\ G_1, \underline{h} : \mathsf{hyp}\ G_1).\exists x : \mathsf{conc}\ G_2.\top$' for $\exists x : \mathsf{hyp}\ G_1 \to \mathsf{conc}\ G_2.\top$ for the abstracted formula, and '$\nu(\underline{u} : \mathsf{nd}\ G_1, \underline{h} : \mathsf{hyp}\ G_1).\mathsf{conc}\ G_2$' for the abstracted LF type.

Definition 1 (Abstraction).

1. *Type-level abstraction:*

$$\nu\rho.\, A_2 = \begin{cases} A_2 & \text{if } \rho = \cdot \\ \nu\rho'.\, A_2 & \text{if } \rho = x : A_1, \rho' \text{ and } A_1 \not\prec A_2 \\ \Pi x : A_1.\, (\nu\rho'.\, A_2) & \text{if } \rho = x : A_1, \rho' \text{ and } A_1 \prec A_2 \end{cases}$$

2. *Object-level abstraction: Let M be well-typed of type A_2*

$$\nu\rho.\, M = \begin{cases} M & \textit{if } \rho = \cdot \\ \nu\rho'.\, M & \textit{if } \rho = x : A_1, \rho' \textit{ and } A_1 \not\prec A_2 \\ \lambda x : A_1.\, (\nu\rho'.\, M) & \textit{if } \rho = x : A_1, \rho' \textit{ and } A_1 \prec A_2 \end{cases}$$

3. *Object-level application: Let M be well-typed of type $\nu\rho.\, A_2$*

$$M\ \rho = \begin{cases} M & \textit{if } \rho = \cdot \\ M\ \rho' & \textit{if } \rho = x : A_1, \rho' \textit{ and } A_1 \not\prec A_2 \\ (M\ x)\ \rho' & \textit{if } \rho = x : A_1, \rho' \textit{ and } A_1 \prec A_2 \end{cases}$$

4. *Meta-level abstraction: Let F be a well-formed formula*

$$\nu\rho.\, \mathcal{P} = \begin{cases} \top & \textit{if } \mathcal{P} = \top \\ \forall x : (\nu\rho.\, A).\nu\rho.\, \mathcal{P}'(x\ \rho) & \textit{if } \mathcal{P} = \forall x : A.\mathcal{P}'(x) \\ \exists x : (\nu\rho.\, A).\nu\rho.\, \mathcal{P}'(x\ \rho) & \textit{if } \mathcal{P} = \exists x : A.\mathcal{P}'(x) \end{cases}$$

Lemma 3 (Soundness of Abstraction).

1. *If $\Gamma, \rho \vdash A : type$ then $\Gamma \vdash \nu\rho.\, A : type$*
2. *If $\Gamma, \rho \vdash M : A$ then $\Gamma \vdash \nu\rho.\, M : \nu\rho.\, A$*
3. *If $\Gamma \vdash M : \nu\rho.\, A$ then $\Gamma \vdash M\ \rho : A$*

With all this machinery in place, we can give a specialized induction principle for the higher-order type of natural deductions nd.

Definition 2 (Specialized induction principle for type family 'nd'). *This induction principle is designed for the world*

$$\Phi = (L : \text{some } G : \mathsf{o} \text{ block } \underline{u} : \mathsf{nd}\ G, \underline{h} : \mathsf{hyp}\ G)^*$$

Let $\mathcal{P}$ be the property to be proven. $\mathsf{ind\_nd}$ *is the induction principle for* nd.

$$\frac{\begin{array}{l} \Psi_1, G : \mathsf{o}, [\underline{u} : \mathsf{nd}\ G, \underline{h} : \mathsf{hyp}\ G]^L, \Psi_2(G, \underline{u}) \\ \quad \Vdash_{\Sigma;\Phi} \mathcal{P}(G, \underline{u}) \\ \Psi_1, G_1 : \mathsf{o}, G_2 : \mathsf{o}, D' : \mathsf{nd}\ G_1 \to \mathsf{nd}\ G_2, \\ \quad \mathbf{x} \in \nu(\underline{u} : \mathsf{nd}\ G_1, \underline{h} : \mathsf{hyp}\ G_1).\mathcal{P}(G_2, D'\ \underline{u}), \Psi_2(G_1 \supset G_2, \mathsf{impI}\ D') \\ \quad \Vdash_{\Sigma;\Phi} \mathcal{P}(G_1 \supset G_2, \mathsf{impI}\ D') \\ \Psi_1, G_1 : \mathsf{o}, G_2 : \mathsf{o}, D_1 : \mathsf{nd}\ (G_2 \supset G_1), D_2 : \mathsf{nd}\ G_2, \\ \quad \mathbf{x}_1 \in \mathcal{P}(G_2 \supset G_1, D_1), \mathbf{x}_2 \in \mathcal{P}(G_2, D_2), \Psi_2(G_1, \mathsf{impE}\ D_1\ D_2) \\ \quad \Vdash_{\Sigma;\Phi} \mathcal{P}(G_1, \mathsf{impE}\ D_1\ D_2) \end{array}}{\Psi_1, G : \mathsf{o}, D : \mathsf{nd}\ G, \Psi_2(G, D) \Vdash_{\Sigma;\Phi} \mathcal{P}(G, D)}\ \mathsf{ind\_nd}$$

The first premiss enforces that $\mathcal{P}$ holds for an arbitrary but fixed parameter block $[\underline{u} : \mathsf{nd}\ G, \underline{h} : \mathsf{hyp}\ G]^L \in \Gamma$. The second premiss guarantees that $\mathcal{P}$ holds for any derivation ending in ($\mathsf{impI}\ D'$), and the third that $\mathcal{P}$ holds for any derivation ending in ($\mathsf{impE}\ D_1\ D_2$).

The soundness of this induction principle follows from considerations presented in Section 5. With this induction principle formula (3) is directly, elegantly, and formally provable.

Proof.

$\mathcal{P}_1 :: G : \mathsf{o}, [\underline{u} : \mathsf{nd}\ G, \underline{h} : \mathsf{hyp}\ G]^L \vdash \mathsf{init}\ \underline{h} : \mathsf{conc}\ G$ in LF
$\mathcal{P}_2 :: G : \mathsf{o}, [\underline{u} : \mathsf{nd}\ G, \underline{h} : \mathsf{hyp}\ G]^L \Vdash_{\Sigma;\Phi} \exists Q : \mathsf{conc}\ G.\top$ by $\exists\mathsf{I}$ on $\mathcal{P}_1$ and $\top\mathsf{I}$

Let $\Psi_2 = G_1 : \mathsf{o}, G_2 : \mathsf{o}, D' : \mathsf{nd}\ G_1 \to \mathsf{nd}\ G_2, \mathbf{x} \in \exists Q_1 : \mathsf{hyp}\ G_1 \to \mathsf{conc}\ G_2.\top$.
$\mathcal{Q}_1 :: \Psi_2 \Vdash_{\Sigma;\Phi} \exists Q_1 : \mathsf{hyp}\ G_1 \to \mathsf{conc}\ G_2.\top$ by assumption
$\mathcal{Q}_2 :: \Psi_2, Q' : \mathsf{hyp}\ G_1 \to \mathsf{conc}\ G_2 \vdash \mathsf{impR}\ Q' : \mathsf{conc}\ (G_1 \supset G_2)$ in LF
$\mathcal{Q}_3 :: \Psi_2, Q' : \mathsf{hyp}\ G_1 \to \mathsf{conc}\ G_2 \Vdash_{\Sigma;\Phi} \exists Q : \mathsf{conc}\ (G_1 \supset G_2).\top$ by $\exists\mathsf{I}$ on $\mathcal{Q}_2$ and $\top\mathsf{I}$
$\mathcal{Q}_4 :: \Psi_2 \Vdash_{\Sigma;\Phi} \exists Q : \mathsf{conc}\ (G_1 \supset G_2).\top$ by $\exists\mathsf{E}$ on $\mathcal{Q}_1$ and $\mathcal{Q}_3$

Let $\Psi_3 = G_1 : \mathsf{o}, G_2 : \mathsf{o}, D_1 : \mathsf{nd}\ (G_1 \supset G_2), D_2 : \mathsf{nd}\ G_2,$
$\mathbf{x}_1 : \exists Q_1 : \mathsf{conc}\ (G_2 \supset G_1).\top, \mathbf{x}_2 : \exists Q_2 : \mathsf{conc}\ G_2.\top$.
and $\Psi_3' = \Psi_3, Q_1 : \mathsf{conc}\ (G_2 \supset G_1), Q_2 : \mathsf{conc}\ G_2$
$\mathcal{R}_1 :: \Psi_3 \vdash \exists Q_1 : \mathsf{conc}\ (G_2 \supset G_1).\top$ by assumption
$\mathcal{R}_2 :: \Psi_3, Q_1 : \mathsf{conc}\ (G_2 \supset G_1) \vdash \exists Q_2 : \mathsf{conc}\ G_2.\top$ by assumption
$\mathcal{R}_3 :: \Psi_3', h : \mathsf{hyp}(G_2 \supset G_1) \vdash \mathsf{impL}\ Q_2\ \mathsf{init}\ h : \mathsf{conc}\ G_1$ in LF
$\mathcal{R}_4 :: \Psi_3' \vdash \mathsf{impL}\ Q_2\ \mathsf{init} : \mathsf{hyp}(G_2 \supset G_1) \to \mathsf{conc}\ G_1$ in LF using $\mathcal{R}_3$
$\mathcal{R}_5 :: \Psi_3' \vdash \mathsf{cut}\ Q_1 (\mathsf{impL}\ Q_2\ \mathsf{init}) : \mathsf{conc}\ G_1$ in LF using $\mathcal{R}_4$
$\mathcal{R}_6 :: \Psi_3' \Vdash_{\Sigma;\Phi} \exists Q : \mathsf{conc}\ G_1.\top$ by $\exists\mathsf{I}$ on $\mathcal{R}_5$ and $\top\mathsf{I}$
$\mathcal{R}_7 :: \Psi_3, Q_1 : \mathsf{conc}\ (G_2 \supset G_1) \Vdash_{\Sigma;\Phi} \exists Q : \mathsf{conc}\ G_1.\top$ by $\exists\mathsf{E}$ on $\mathcal{R}_2$ and $\mathcal{R}_6$
$\mathcal{R}_8 :: \Psi_3 \Vdash_{\Sigma;\Phi} \exists Q : \mathsf{conc}\ G_1.\top$ by $\exists\mathsf{E}$ on $\mathcal{R}_1$ and $\mathcal{R}_7$

$\mathcal{D}_1 :: G : \mathsf{o}, D : \mathsf{nd}\ G \Vdash_{\Sigma;\Phi} \exists Q : \mathsf{conc}\ G.\top$ by $\mathsf{ind\_nd}$ on $\mathcal{P}_2, \mathcal{Q}_4$ and $\mathcal{R}_8$
$\mathcal{D}_2 :: G : \mathsf{o} \Vdash_{\Sigma;\Phi} \forall D : \mathsf{nd}\ G.\exists Q : \mathsf{conc}\ G.\top$ by $\forall\mathsf{I}$ on $\mathcal{D}_1$
$\mathcal{D}_3 :: \cdot \Vdash_{\Sigma;\Phi} \forall G : \mathsf{o}.\forall D : \mathsf{nd}\ G.\exists Q : \mathsf{conc}\ G.\top$ by $\forall\mathsf{I}$ on $\mathcal{D}_2$

We begin now with the presentation of the general form of the induction principle that works for all higher-order encodings in LF. This principle ends in a conclusion of the form $\Psi \Vdash_{\Sigma;\Phi} \mathcal{P}$. The challenge in designing this principle is (a) we need to pick one or more declarations in Ψ as case subjects (b) we can't rely on patterns to be linear (in the sense of linear pattern as in pattern-matching), and (c) case subjects might be instantiated by parameters from an LF context. All three points are illustrated by the principle $\mathsf{ind\_nd}$: It shows that (a) we distinguish cases over G, and D simultaneously, (b) G occurs by itself and as an index to D, and (c) D might be instantiated by an $\underline{u} : \mathsf{nd}\ G \in \Gamma \in \mathcal{L}(\Phi)$.

Our solution employs substitutions as patterns, i.e. patterns in the sense of pattern matching. A pattern is a pair of a context which gathers all free variables that occur in the pattern, and a substitution σ that is defined as follows:

$$\sigma ::= \cdot \mid \sigma, M/x \mid \sigma, \rho'/\rho \mid \sigma, \mathbf{x}'/\mathbf{x}$$

The domain of σ is always Ψ. Example $\mathsf{ind\_nd}$ used three patterns $(\Psi_i; \sigma_i)$:

$\Psi_1 = G : \mathsf{o}, [\underline{u} : \mathsf{nd}\ G, \underline{h} : \mathsf{hyp}\ G]^L$
$\sigma_1 = G/G, \underline{u}/D$

$\Psi_2 = G_1 : \mathsf{o}, G_2 : \mathsf{o}, D' : \mathsf{nd}\ G_1 \to \mathsf{nd}\ G_2, \mathbf{x}_1 \in \exists x : \mathsf{hyp}\ G_1 \to \mathsf{conc}\ G_2.\top$
$\sigma_2 = G_1 \supset G_2/G, \mathsf{impI}\ D'/D$

$\Psi_3 = G_1 : \mathsf{o}, G_2 : \mathsf{o}, D_1 : \mathsf{nd}\ (G_2 \supset G_1), D_2 : \mathsf{nd}\ G_2,$
$\mathbf{x}_1 \in \exists x : \mathsf{conc}\ (G_2 \supset G_1).\top, \mathbf{x}_2 \in \exists x : \mathsf{conc}\ G_2.\top$
$\sigma_3 = G_1/G, \mathsf{impE}\ D_1\ D_2/D$

$$\frac{}{\Psi' \Vdash_{\Sigma;\Phi} \cdot : \Psi}\text{sub-empty} \qquad \frac{\Psi' \Vdash_{\Sigma;\Phi} \sigma : \Psi \qquad \overline{\Psi} \vdash_{\Sigma} M : [\sigma]A}{\Psi' \Vdash_{\Sigma;\Phi} \sigma, M/x : \Psi, x : A}\text{sub-lf}$$

$$\frac{\Psi' \Vdash_{\Sigma;\Phi} \sigma : \Psi \qquad [\rho']^L \in \Psi' \qquad \overline{\Psi'} \vdash \rho' \equiv_\alpha [\sigma]\rho}{\Psi' \Vdash_{\Sigma;\Phi} \sigma, \rho'/\rho : \Psi, [\rho]^L}\text{sub-block}$$

$$\frac{\Psi' \Vdash_{\Sigma;\Phi} \sigma : \Psi \qquad \Psi'(\mathbf{x}') = [\sigma]\mathcal{P}}{\Psi' \Vdash_{\Sigma;\Phi} \sigma, \mathbf{x}'/\mathbf{x} : \Psi, \mathbf{x} \in \mathcal{P}}\text{sub-meta}$$

Fig. 5. Well-defined substitutions

We write $\Psi_i \Vdash_{\Sigma;\Phi} \sigma_i : \Psi$ for well-defined substitutions which are defined in Figure 5. Not every pair $(\Psi';\sigma)$ however is a valid pattern. Intuitively, all free variables in σ should be instantiated when matching against an instantiation of Ψ'. This requirement excludes Ψ''s that are too large containing unnecessary declarations. It also excludes patterns that contain variables that may not be instantiated by matching. These are variables that occur in flex/flex positions during matching, i.e. variables that do not occur in strict positions in the pattern. Therefore, we restrict arbitrary context/substitutions pairs to patterns. Third when new hypotheses $\mathbf{x} \in \mathcal{P}$ are introduced, we must carefully ensure that they are "smaller" than the property to be proven.

Definition 3 (Patterns). *$(\Psi';\sigma)$ is a pattern for $(\Psi;\mathcal{P})$ iff $\Psi' \Vdash_{\Sigma;\Phi} \sigma : \Psi$ and there exists a well-founded ordering on substitutions $\prec$, s.t. Ψ' is* safe. *Ψ' is an safe context iff*

1. *$\Psi' = \cdot$.*
2. *$\Psi' = \Psi'', x : A$ and Ψ'' is safe and x occurs in a strict position in σ.*
3. *$\Psi' = \Psi'', [\rho]^L$ and Ψ'' is safe and $\underline{u} \in \rho$ occurs in a strict position in σ.*
4. *$\Psi' = \Psi'', [\rho]^L$ and Ψ'' is safe and ρ occurs in a strict position in σ.*
5. *$\Psi' = \Psi'', \mathbf{x} \in \mathcal{P}'$ and Ψ'' is safe and $\mathbf{x}$ occurs in a strict position in σ.*
6. *$\Psi' = \Psi'', \mathbf{x} \in \mathcal{P}'$ and Ψ'' is safe and $\mathcal{P}' = \nu\rho_1. \ldots \nu\rho_n. [\sigma']\mathcal{P}$ and $\sigma' \prec \sigma$ for some block variables $\rho_1^{L_1}, \ldots, \rho_n^{L_n} \in \Phi$*

Properties 1. through 5. guarantee that each variable declared in Ψ' will be instantiated once σ is matched against an instantation of Ψ. 6. enforces the well-foundedness of the principle.

Finally, we can address the question when an induction principle covers all cases. Because LF possesses canonical forms, any object of given type has only finitely many head constructors (buried under finitely many abstractions). Let Φ be a world, $\Gamma \in \mathcal{L}(\Phi)$, and $\Psi \Vdash_{\Sigma;\Phi} \mathcal{P}$ the conclusion of the induction principle. Furthermore, let $\{(\Psi_1;\sigma_1)\ldots(\Psi_n;\sigma_n)\}$ a set of patterns. The induction principle is valid only if for any possible instantiation of declarations in Ψ one pattern exists that matches it.

Definition 4 (Cover). *A set of patterns* $\{(\Psi_1;\sigma_1)\dots(\Psi_n;\sigma_n)\}$ *for* $(\Psi;\mathcal{P})$ *is a* cover *for a world* Φ *if for any* $\Gamma \in \mathcal{L}(\Phi)$ *and* η *with satisfies* $\Gamma \vdash \eta : \Psi$ *there exists* $1 \leq i \leq n$ *and an* η' *for which* $\Gamma \vdash \eta' : \Psi^i$*, s.t.*

$$\eta = \sigma_i \circ \eta'$$

Definition 5 (Induction principle for higher-order LF encodings). *Let* $\{(\Psi_1;\sigma_1)\dots(\Psi_n;\sigma_n)\}$ *a set of patterns for* $(\Psi;\mathcal{P})$ *that form a* cover *for the world* Φ*. Then we define the induction principle for higher-order encodings as follows.*

$$\frac{\Psi_1 \Vdash_{\Sigma;\Phi} \mathcal{P}[\sigma_1] \qquad \dots \qquad \Psi_n \Vdash_{\Sigma;\Phi} \mathcal{P}[\sigma_n]}{\Psi \Vdash_{\Sigma;\Phi} \mathcal{P}}\,\mathsf{ind}$$

In the interest of space, we omit the two syntactical criteria for termination and coverage [Sch00].

5 Meta-Theory

The fragment of $\mathcal{IL}$ that is defined in Figure 1 and used throughout this paper is closely connected to the λ-calculus through a realizability interpretation similar to the Curry-Howard isomorphism. Every rule in $\mathcal{IL}$ can be endowed with a proof term, that corresponds to a total function. Formulas correspond to types. The calculus of recursive functions for higher-order encodings [Sch01] forms the type-theoretic foundation of this realizer calculus. In this section we show, that $\mathcal{IL}$ endowed with rule ind (written as $\mathcal{IL}^{\mathsf{ind}}$) is a sound extension of $\mathcal{IL}$.

The induction principle ind is compatible with this view of provability, and it extends the realizability interpretation from above in a natural and straightforward way. Concretely, it allows realizers to be defined by case analysis and recursion. In addition, $\mathcal{IL}^{\mathsf{ind}}$'s derivations still correspond to proof terms, because ind satisfies Definition 4 which means that the functions corresponding to derivations are guaranteed to cover all cases at all times, and since ind also satisfies Condition 6. of Definition 3, recursion is always terminating.

We show that $\mathcal{IL}^{\mathsf{ind}}$ is sound by extending the realizability interpretation of $\mathcal{IL}$. First we show the admissibility of the ind rule in Lemma 4 which we generalize to a proof that any derivation in $\mathcal{IL}^{\mathsf{ind}}$ corresponds to a realizer in Theorem 4.

Lemma 4 (Admissibility of $\mathcal{IL}^{\mathsf{ind}}$). *Let* $\overset{\mathcal{D}}{\Psi \Vdash_{\Sigma;\Phi} \mathcal{P}}$ *be a derivation in* $\mathcal{IL}^{\mathsf{ind}}$ *with* ind *being the last applied rule whose premisses* $\mathcal{D}_1 \dots \mathcal{D}_n$ *have corresponding realizers. Let* $\Gamma \in \mathcal{L}(\Phi)$*. Then, for all substitutions (environments)* η*,* $\Gamma \Vdash_{\Sigma;\Phi} \eta : \Psi$*, there exists a total function* f *(of type* $\mathcal{P}$*) that realizes* $\mathcal{D}$*.*

Proof. Let $\mathcal{D}$ be the following derivation in $\mathcal{IL}^{\mathsf{ind}}$.

$$\frac{\overset{\mathcal{D}_1}{\Psi_1 \Vdash_{\Sigma;\Phi} \mathcal{P}[\sigma_1]} \qquad \dots \qquad \overset{\mathcal{D}_n}{\Psi_n \Vdash_{\Sigma;\Phi} \mathcal{P}[\sigma_n]}}{\Psi \Vdash_{\Sigma;\Phi} \mathcal{P}}\,\mathsf{ind}$$

For any instantiation of Ψ, there exists a pattern $(\Psi_i; \sigma_i)$ that matches it, by Definition 4. This matching operation instantiates all declarations in Ψ (by Properties 1. to 5. in Definition 3) that occur strictly in σ_i by LF objects (for $x : A$), parameters from the LF contexts (for the $[\rho]^L$), and values computed by the realizers (for $\mathbf{x} \in \mathcal{P}$). Variable $\mathbf{x}$ declared in Ψ that do not occur strictly have to be computed recursively an operation that terminates by Property 6. in Definition 3. Once fully applied, the realizer that corresponds to $\mathcal{D}_i$ computes the desired result.

Theorem 4 (Soundness of $\mathcal{IL}^{\mathsf{ind}}$). *Let* $\Psi \overset{\mathcal{D}}{\Vdash}_{\Sigma;\Phi} \mathcal{P}$ *be a derivation in* $\mathcal{IL}^{\mathsf{ind}}$ *and* Γ *be a valid world. Then, for all substitutions (environments)* η, $\Gamma \Vdash_{\Sigma;\Phi} \eta : \Psi$, *there exists a total function* f *of type* $\mathcal{P}$ *that realizes* $\mathcal{D}$.

Proof. By induction on $\mathcal{D}$ using Lemma 4 and the Curry-Howard isomorphism.

6 Related Work

In earlier work we have used modal logic to add induction and primitive recursion principles to the simply typed λ-calculus while preserving higher-order encodings [DPS97,Hof99]. The calculus differs from this work in that it defines only one language suitable for representation and reasoning simultaneously. In its original formulation, the calculus was lacking dependent types, parts of which were added in [Lel98]. Others have followed a categorical approach. Hofmann [Hof99], for example, suggests a number of different induction and reasoning principles for higher-order abstract syntax and he proved all adequate using a category-theoretic method. Earlier, [DH94] has proposed techniques to reformulate natural higher-order as first-order encodings using auxiliary types reducing hereby the problem of induction for higher-order to standard induction. With this approach, however, the user remains responsible for proving substitution lemmas.

One advantage of using a logical framework such as LF is that the choice of variable names can be left to the logical framework. The same goal is pursued by [GM96,GP99] by exploiting permutation properties among variable names. Their work has been implemented in FreshML [PG00] and is therefore related to the realizer calculus underlying this work [Sch01]. FreshML supports higher-order abstract syntax as well but it does not support dependent types.

Honsell et al. [HMS01] use higher-order representation techniques to formalize concepts from process algebras. They have combined higher-order encodings with coinduction principles and implemented their design in Coq [DFH$^+$93].

Reasoning by induction is also supported by the calculus of partial inductive definitions [Hal87] and definitional reflection [SH93], however, both designs require induction subjects to be closed.

7 Conclusion

Higher-order encodings supported by logical frameworks such as LF bring many advantages in terms of elegance, efficiency, and maintenance, especially when representing complex systems, such as derivation systems, logics, type systems, operational semantics and others. They bring, however, at least one disadvantage: it is not easy to reason about them. In this paper we have developed a general induction principle for higher-order encodings and added it to first-order intuitionistic logic. The soundness of the design follows from a type theoretic argument by restricting LF contexts to be regularly formed.

We have implemented a prototype version of a theorem prover that uses worlds and the ind-rule in Twelf [PS99]. Twelf has successfully proven, for example, the equivalence of Hilbert's calculus, natural deduction calculus, and the sequent calculus. Other experiments include the proof of the Church-Rosser theorem for the untyped and the simply-typed λ-calculus, and cut-elimination results for various propositional and first-order sequent calculi.

Our proposal provides one possible solution for the tension between reasoning by induction and higher-order encodings. It is applicable to the untyped, simply typed, and dependently typed setting. In addition, it does not interfere with any properties of the underlying logical framework, such as properties that are associated with contexts or variables.

References

[Coq91] Thierry Coquand. An algorithm for testing conversion in type theory. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 255–279. Cambridge University Press, 1991.

[DFH+93] Gilles Dowek, Amy Felty, Hugo Herbelin, Gérard Huet, Chet Murthy, Catherine Parent, Christine Paulin-Mohring, and Benjamin Werner. The Coq proof assistant user's guide. Rapport Techniques 154, INRIA, Rocquencourt, France, 1993. Version 5.8.

[DH94] Joëlle Despeyroux and André Hirschowitz. Higher-order abstract syntax with induction in Coq. In Frank Pfenning, editor, *Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning*, pages 159–173, Kiev, Ukraine, July 1994. Springer-Verlag LNAI 822.

[DPS97] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. In R. Hindley, editor, *Proceedings of the Third International Conference on Typed Lambda Calculus and Applications (TLCA'97)*, pages 147–163, Nancy, France, April 1997. Springer-Verlag LNCS 1210.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

[GM96] Andrew D. Gordon and Tom Melham. Five axioms of alpha-conversion. In J. von Wright, J. Grundy, and J. Harrison, editors, *Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'96)*, pages 173–191, Turku, Finland, August 1996. Springer-Verlag LNCS 1125.

[GP99] Murdoch Gabbay and Andrew Pitts. A new approach to abstract syntax involving binders. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 214–224, Trento, Italy, July 1999. IEEE Computer Society Press.

[Hal87] Lars Hallnäs. A note on the logic of a logic program. In *Proceedings of the Workshop on Programming Logic*. University of Göteborg and Chalmers University of Technology, Report PMG-R37, 1987.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[HMS01] Furio Honsell, Marino Miculan, and Ivan Scagnetto. π-calculus in (Co)inductive type theory. *Theoretical Computer Science*, 253(2):239–285, 2001.

[Hof99] Martin Hofmann. Semantical analysis for higher-order abstract syntax. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 204–213, Trento, Italy, July 1999. IEEE Computer Society Press.

[Lel98] Pierre Leleu. *Induction et Syntaxe Abstraite d'Ordre Supérieur dans les Théories Typées.* PhD thesis, Ecole Nationale des Ponts et Chaussees, Marne-la-Vallee, France, December 1998.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover.* Springer-Verlag LNCS 828, 1994.

[Pfe91] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[Pfe95] Frank Pfenning. Structural cut elimination. In D. Kozen, editor, *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, pages 156–166, San Diego, California, June 1995. IEEE Computer Society Press.

[Pfe99] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume II. Elsevier Science Publishers, 1999.

[PG00] A. M. Pitts and M. J. Gabbay. A metalanguage for programming with bound names modulo renaming. In R. Backhouse and J. N. Oliveira, editors, *Mathematics of Program Construction, MPC2000, Proceedings, Ponte de Lima, Portugal, July 2000*, volume 1837 of *Lecture Notes in Computer Science*, pages 230–255. Springer-Verlag, Heidelberg, 2000.

[PM93] Christine Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 328–345, Utrecht, The Netherlands, March 1993. Springer-Verlag LNCS 664.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.

[Sch00] Carsten Schürmann. *Automating the Meta-Theory of Deductive Systems.* PhD thesis, Carnegie Mellon University, 2000. CMU-CS-00-146.

[Sch01] Carsten Schürmann. Recursion for higher-order encodings. In Laurent Fribourg, editor, *Proceedings of the Conference on Computer Science Logic (CSL 2001)*, pages 585–599, Paris, France, August 2001. Springer-Verlag LNCS 2142.

[SH93] Peter Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 222–232, Montreal, Canada, June 1993.

[Vir99] Roberto Virga. *Higher-Order Rewriting with Dependent Types.* PhD thesis, Carnegie Mellon University, 1999. CMU-CS-99-167.