Induction and Co-induction in Sequent Calculus

Alberto Momigliano[1,2] and Alwen Tiu[3,4]

[1] LFCS, University of Edinburgh
[2] DSI, University of Milan
amomigl1@inf.ed.ac.uk
[3] LIX, École polytechnique
[4] Computer Science and Engineering Department, Penn State University
tiu@cse.psu.edu

Abstract. Proof search has been used to specify a wide range of computation systems. In order to build a framework for reasoning about such specifications, we make use of a sequent calculus involving induction and co-induction. These proof principles are based on a proof theoretic notion of *definition* [26, 9, 13] Definitions are essentially stratified logic programs. The left and right rules for defined atoms treat the definitions as defining fixed points. The use of definitions makes it possible to reason intensionally about syntax, in particular enforcing free equality via unification. The full system thus allows inductive and co-inductive proofs involving higher-order abstract syntax. We extend earlier work by allowing induction and co-induction on general definitions and show that cut-elimination holds for this extension. We present some examples involving lists and simulation in the lazy λ-calculus.

1 Introduction

A common approach to specifying computation systems is via deductive systems, e.g., structural operational semantics. Such specifications can be represented as logical theories in a suitably expressive formal logic in which *proof-search* can then be used to model the computation. This use of logic as a specification language is along the line of *logical frameworks* [21]. The representation of the syntax of computation systems inside formal logic can benefit from the use of *higher-order abstract syntax* (HOAS), a high-level and declarative treatment of object-level bound variables and substitution. At the same time, we want to use such a logic in order to reason over the *meta-theoretical* properties of object languages, for example type preservation in operational semantics [14], soundness and completeness of compilation [18] or congruence of bisimulation in transition systems [15]. Typically this involves reasoning by (structural) induction and, when dealing with infinite behaviour, co-induction [5].

The need to support both inductive and co-inductive reasoning and some form of HOAS requires some careful design decisions, since the two are prima facie notoriously incompatible. While any meta-language based on a λ-calculus can be used to specify and possibly perform computations over HOAS encodings, meta-reasoning has traditionally involved (co)inductive specifications both at the level of the syntax and of the judgements as well (which are of course unified at the type-theoretic level). The first provides crucial freeness properties for datatypes constructors, while the second offers principle of case analysis and (co)induction. This is well-known to be problematic, since HOAS specifications lead to non-monotone (co)inductive definitions, which by cardinality and consistency reasons are not permitted in inductive logical frameworks. Moreover, even when HOAS is weakened so as to be made compatible with standard proof assistants [6] such as HOL or Coq, the latter tend to be still too *strong*, in sense of allowing the existence of too many functions and yielding the so called *exotic* terms. This causes a loss of adequacy in HOAS specifications, which is one of the pillar of formal verification. On the other hand, logics such as LF [10] that are weak by design in order to support this style of syntax are not directly endowed with (co)induction principles.

The contribution of this paper lies in the design of a new logic, called Linc (for a logic with λ-terms, induction and co-induction), that carefully adds principles of induction and co-induction to a higher-order intuitionistic logic based on a proof theoretic notion of definition, following on work (among others) by Schroeder-Heister [26], Girard [9] and McDowell and Miller [13]. Definitions are akin to logic programs, but allow to view theories as "closed" or defining fixed points. This alone allows us to perform case analysis. Our approach to formalizing induction and co-induction is via the least and greatest solutions of the fixed point equations specified by the definitions. Such least and greatest solutions are guaranteed to exist by a stratification condition on definitions (which basically ensures monotonicity). The proof rules for induction and co-induction makes use of the notion of *pre-fixed points* and *post-fixed points* respectively. In the inductive case, this corresponds to the induction invariant, while in the co-inductive one to the so-called simulation.

The simply typed language underlying Linc and the notion of definition make it possible to reason *intensionally* about syntax, in particular enforcing *free* equality via unification, which can be used on first-order terms or higher-order λ-terms. In fact, we can support HOAS encodings of constructors without requiring them to belong to a datatype. In particular we can *prove* the freeness properties of those constructors, namely injectivity, distinctness and case exhaustion. Judgements are encoded as definitions accordingly to their informal semantics, either inductive, co-inductive or regular, i.e. true in every fixed point. Given the stratification condition, we (currently) fall short of the LF-like idea of *Full* HOAS, although, exploiting the equivalence with the completion of a logic program [25], the monotonicity requirement can be weakened beyond the scope of current induction-based proof-assistants.

Linc can be proved to be a conservative extension of $FO\lambda^{\Delta\mathbb{N}}$ [13] and a generalization to the higher-order case of Martin-Löf [12] first-order theory of iterated inductive definitions. Moreover, at the best of our knowledge, it is the first sequent calculus with a cut-elimination theorem for co-inductive definitions. Further, its modular design makes its extension easy, for example in the direction of $FO\lambda^{\nabla}$ [17] or the regular world assumption [27].

The rest of the paper is organized as follows. Section 2 introduces the proof system for the logic Linc. Section 3 shows some examples of using induction and co-induction to prove several properties of list-related predicates and the lazy λ-calculus. Section 4 gives an overview of the cut-elimination procedure, the detailed proof of which is available in [30]. Section 5 surveys the related work and Section 6 concludes this paper.

$$\frac{B,B,\Gamma \longrightarrow C}{B,\Gamma \longrightarrow C}\; c\mathcal{L} \quad \frac{\Gamma \longrightarrow C}{B,\Gamma \longrightarrow C}\; w\mathcal{L} \qquad \frac{}{\bot,\Gamma \longrightarrow B}\; \bot\mathcal{L} \quad \frac{}{\Gamma \longrightarrow \top}\; \top\mathcal{R}$$

$$\frac{B,\Gamma \longrightarrow D}{B \wedge C,\Gamma \longrightarrow D}\; \wedge\mathcal{L} \quad \frac{C,\Gamma \longrightarrow D}{B \wedge C,\Gamma \longrightarrow D}\; \wedge\mathcal{L} \qquad \frac{\Gamma \longrightarrow B \quad \Gamma \longrightarrow C}{\Gamma \longrightarrow B \wedge C}\; \wedge\mathcal{R}$$

$$\frac{B,\Gamma \longrightarrow D \quad C,\Gamma \longrightarrow D}{B \vee C,\Gamma \longrightarrow D}\; \vee\mathcal{L} \qquad \frac{\Gamma \longrightarrow B}{\Gamma \longrightarrow B \vee C}\; \vee\mathcal{R} \quad \frac{\Gamma \longrightarrow C}{\Gamma \longrightarrow B \vee C}\; \vee\mathcal{R}$$

$$\frac{Bt,\Gamma \longrightarrow C}{\forall x.Bx,\Gamma \longrightarrow C}\; \forall\mathcal{L} \quad \frac{\Gamma \longrightarrow By}{\Gamma \longrightarrow \forall x.Bx}\; \forall\mathcal{R} \quad \frac{By,\Gamma \longrightarrow C}{\exists x.Bx,\Gamma \longrightarrow C}\; \exists\mathcal{L} \quad \frac{\Gamma \longrightarrow Bt}{\Gamma \longrightarrow \exists x.Bx}\; \exists\mathcal{R}$$

$$\frac{\Gamma \longrightarrow B \quad C,\Gamma \longrightarrow D}{B \supset C,\Gamma \longrightarrow D}\; \supset\mathcal{L} \qquad \frac{B,\Gamma \longrightarrow C}{\Gamma \longrightarrow B \supset C}\; \supset\mathcal{R}$$

$$\frac{}{C \longrightarrow C}\; init \quad \frac{\Delta_1 \longrightarrow B_1 \quad \cdots \quad \Delta_n \longrightarrow B_n \quad B_1,\dots,B_n,\Gamma \longrightarrow C}{\Delta_1,\dots,\Delta_n,\Gamma \longrightarrow C}\; mc,\ \text{where } n > 0$$

Fig. 1. Inference rules for the core Linc

2 The Logic Linc

The logic Linc shares the core fragment with $FO\lambda^{\Delta\mathbb{N}}$, which is an intuitionistic version of Church's Simple Theory of Types. Formulae in the logic are built from predicate symbols and the usual logical connectives $\bot$, $\top$, $\wedge$, $\vee$, $\supset$, $\forall_\tau$ and $\exists_\tau$. Following Church, formulae will be given type o. The quantification type τ can have higher types, but those are restricted to not contain o. Thus the logic has a first-order proof theory but allows for the encoding of higher-order abstract syntax. The core fragment of the logic is presented in the sequent calculus in Figure 1. A sequent is denoted by $\Gamma \longrightarrow C$ where C is a formula and Γ is a multiset of formulae. Notice that in the presentation of the rule schemes, we make use of HOAS, e.g., in the application Bx it is implicit that B has no free occurrence of x. In the $\forall\mathcal{R}$ and $\exists\mathcal{L}$ rules, y is an eigenvariable that is not free in the lower sequent of the rule. Whenever we write down a sequent, it is assumed implicitly that the formulae are well-typed and in $\beta\eta$-long normal forms: the type context, i.e., the types of the constants and the eigenvariables used in the sequent, is left implicit as well. The *mc* rule is a generalization of the cut rule that simplifies the presentation of the cut-elimination proof.

We extend the core logic in Figure 1 by allowing the introduction of non-logical constants. An atomic formula, i.e., a formula that contains no occurrences of logical constants, can be defined in terms of other logical or non-logical constants. Its left and right rules are, roughly speaking, carried out by replacing the formula corresponding to its definition with the atom itself. A defined atom can thus be seen as a generalized connective, whose behaviour is determined by its defining clauses. The syntax of definition clauses used by McDowell and Miller [13] resembles that of logic programs, that is, a definition clause consists of a head and a body, with the usual pattern matching in the head; for example, the predicate *nat* for natural numbers is written $\{nat\ z \stackrel{\triangle}{=} \top,\ nat\ s\ x \stackrel{\triangle}{=} nat\ x\}$. We adopt here a simpler presentation by putting all pattern matching in the body and combining multiple clauses with the same head in one clause with disjunctive body. Of course, this will require us to have explicit equality as part of our syntax. The corresponding *nat* predicate in our syntax will be written

$$nat\,x \stackrel{\triangle}{=} [x = z] \lor \exists y.[x = s\,y] \land nat\,y$$

and corresponds to the notion of *iff-completion* of a logic program.

Definition 1. *A* definition clause *is written* $\forall \bar{x}[p\bar{x} \stackrel{\triangle}{=} B\bar{x}]$*, where* p *is a predicate constant. The atomic formula* $p\bar{x}$ *is called the* head *of the clause, and the formula* $B\bar{x}$ *is called the* body*. The symbol* $\stackrel{\triangle}{=}$ *is used simply to indicate a definition clause: it is not a logical connective. A* definition *is a (perhaps infinite) set of definition clauses. A predicate may occur only at most once in the heads of the clauses of a definition.*

We will generally omit the outer quantifiers to simplify the presentation. Not all definition clauses are admitted in our logic, e.g., we rule out definitions with circular calling through implications (negations) that can lead to inconsistency [24]. The notion of *level* of a formula allows to define a proper stratification on definitions. To each predicate p we associate a natural number $\mathrm{lvl}(p)$, the level of p.

Definition 2. *Given a formula* B*, its* level $\mathrm{lvl}(B)$ *is defined as follows:*

1. $\mathrm{lvl}(p\bar{t}) = \mathrm{lvl}(p)$, $\mathrm{lvl}(\bot) = \mathrm{lvl}(\top) = 0$
2. $\mathrm{lvl}(B \land C) = \mathrm{lvl}(B \lor C) = \max(\mathrm{lvl}(B), \mathrm{lvl}(C))$
3. $\mathrm{lvl}(B \supset C) = \max(\mathrm{lvl}(B) + 1, \mathrm{lvl}(C))$
4. $\mathrm{lvl}(\forall x.Bx) = \mathrm{lvl}(\exists x.Bx) = \mathrm{lvl}(Bt)$*, for any term* t*.*

The level of a sequent $\Gamma \longrightarrow C$ is the level of C. A definition clause $\forall \bar{x}[p\bar{x} \stackrel{\triangle}{=} B\bar{x}]$ is stratified if $\mathrm{lvl}(B\bar{x}) \leq \mathrm{lvl}(p)$. An occurrence of a formula A in a formula C is *strictly positive* if that particular occurrence of A is not to the left of any implication in C. Stratification then implies that all occurrences of the head in the body are strictly positive.

Given a definition clause $p\bar{x} \stackrel{\triangle}{=} B\bar{x}$, the right and left rules for predicate p are

$$\frac{B\bar{t}, \Gamma \longrightarrow C}{p\bar{t}, \Gamma \longrightarrow C}\ def\mathcal{L} \qquad \frac{\Gamma \longrightarrow B\bar{t}}{\Gamma \longrightarrow p\bar{t}}\ def\mathcal{R}$$

The rules for equality predicates makes use of substitutions. We assume the usual definition of capture-avoiding substitutions. We use θ, ρ, δ and σ to denote those and their application is written in post-fix notation, e.g., $t\theta$. The left and right rules for equality are as follows

$$\frac{\{\Gamma\rho \longrightarrow C\rho \mid s\rho =_{\beta\eta} t\rho, \rho \in CSU(s,t)\}}{s = t, \Gamma \longrightarrow C}\ \mathrm{eq}\mathcal{L} \qquad \frac{}{\Gamma \longrightarrow t = t}\ \mathrm{eq}\mathcal{R}$$

The substitution ρ in $\mathrm{eq}\mathcal{L}$ is called a *unifier* of s and t. The set $CSU(s,t)$ is a *complete set of unifiers*, i.e., given any unifier θ_1 of s and t, there is a unifier $\theta_2 \in CSU(s,t)$ such that $\theta_1 = \theta_2 \circ \gamma$, for some substitution γ. In the first order case, a set containing just the most general unifier is a complete set of unifiers. In general, however, the complete set of unifiers may contain more than one unifier and therefore we specify a set of sequents as the premise of the $\mathrm{eq}\mathcal{L}$ rule, which is to say that each sequent in the set is a premise of the rule. Note that in applying $\mathrm{eq}\mathcal{L}$, eigenvariables can be instantiated as a result.

A definition $px \stackrel{\triangle}{=} Bx$ can be seen as a fixed point equation saying that for every term t, pt if and only if Bt holds. Since our notion of definition requires strict positivity of occurrences of p in B, existence of fixed points is always guaranteed. Hence the provability of $p\,t$ means that t is in a solution of the corresponding fixed point equation, although not necessarily in the least (or greatest) solution (see e.g., [9] for an example). Therefore we add extra rules that reflect the least and the greatest solutions, respectively. Since we are in a monotone setting, we can use the pre-fixed point and the post-fixed point as an approach to the least and greatest fixed points. In the following we assume, for simplicity of presentation, that predicates are not mutual-recursively defined. The more general case where mutual recursion is treated can be found in [30].

Let $p\,\bar{x} \overset{\triangle}{=} B\,\bar{x}$ be a definition clause and let S be a term of the same type as p. The induction rules for p are

$$\frac{(B\,\bar{x})[S/p] \longrightarrow S\,\bar{x} \quad \Gamma, S\,\bar{t} \longrightarrow C}{\Gamma, p\,\bar{t} \longrightarrow C}\ \mathcal{IL} \qquad \frac{\Gamma \longrightarrow B\,\bar{t}}{\Gamma \longrightarrow p\,\bar{t}}\ \mathcal{IR}$$

The abstraction S is an invariant of the induction. The variables $\bar{x}$ are new eigenvariables. An informal reading of $\mathcal{IL}$ is to consider S as denoting a set (i.e., $\bar{t} \in S$ iff $S\,\bar{t}$ holds), B as denoting a fixed point operator and S as a pre-fixed point of B, i.e., $B[S/p] \subseteq S$. Notice that the right-rule for induction is $\mathit{def}\mathcal{R}$. The co-induction rules are defined dually.

$$\frac{B\,\bar{t}, \Gamma \longrightarrow C}{p\,\bar{t}, \Gamma \longrightarrow C}\ \mathcal{CIL} \qquad \frac{\Gamma \longrightarrow S\,\bar{t} \quad S\,\bar{x} \longrightarrow (B\,\bar{x})[S/p]}{\Gamma \longrightarrow p\,\bar{t}}\ \mathcal{CIR},\ \text{where } \mathrm{lvl}(S) \leq \mathrm{lvl}(p)$$

S can be seen as denoting a *post-fixed point*, i.e., $S \subseteq B[S/p]$. The $\mathcal{CIL}$ rule is the $\mathit{def}\mathcal{L}$ rule. The proviso in $\mathcal{CIR}$, although mainly technical, is satisfied by every example we have examined, since it requires the given predicate to be used "monotonically" in the simulation.

To avoid inconsistency, some care must be taken in applying induction or co-induction in a proof. One obvious pitfall is when the fixed point equation corresponding to a definition clause has different least and greatest solutions. In such case, mixing induction and co-induction on the same definition clause can lead to inconsistency. For example, let $p \overset{\triangle}{=} p$ be a definition clause. Given the scheme of rules above without any further restriction, we can construct the following derivation

$$\dfrac{\dfrac{\dfrac{}{\longrightarrow \top}\ \top\mathcal{R} \qquad \dfrac{}{\top \longrightarrow \top}\ \top\mathcal{R}}{\longrightarrow p}\ \mathcal{CIR} \qquad \dfrac{\dfrac{}{\bot \longrightarrow \bot}\ \bot\mathcal{L} \qquad \dfrac{}{\bot \longrightarrow \bot}\ \bot\mathcal{L}}{p \longrightarrow \bot}\ \mathcal{IL}}{\longrightarrow \bot}\ \mathit{cut}$$

In the above derivation we use $\top$ and $\bot$ as the invariant and the simulation in the instance of $\mathcal{CIR}$ and $\mathcal{IL}$ rules. This example suggests that we have to use a definition clause consistently through out the proof, either inductively or co-inductively, but not both. To avoid this problem, we introduce markings into a definition, whose role is to indicate which rules are applicable to the corresponding defined atoms.

Definition 3. *An* extended definition *is a stratified definition $\mathcal{D}$ together with a label, that indicates whether the clause is either* inductive, co-inductive, *or* regular. *An inductive clause is written as $p\,\bar{x} \overset{\mu}{=} B\,\bar{x}$, a co-inductive clause is written as $p\,\bar{x} \overset{\nu}{=} B\,\bar{x}$ and a regular clause is written as $p\,\bar{x} \overset{\triangle}{=} B\,\bar{x}$.*

Since we shall only be concerned with extended definitions from now on, we shall refer those simply as definitions. The induction and co-induction rules need additional provisos. The $I\mathcal{L}$ and $I\mathcal{R}$ rules can be applied only to an inductively defined atom. Dually, the $\mathcal{C}I\mathcal{L}$ and $\mathcal{C}I\mathcal{R}$ rules can only be applied to a co-inductively defined atom. The $def\mathcal{L}$ and $def\mathcal{R}$ rules apply only to regular atoms. However, we can show that $def\mathcal{L}$ and $def\mathcal{R}$ are derived rules for (co-)inductively defined atoms.

Proposition 1. *The $def\mathcal{L}$ and $def\mathcal{R}$ are admissible rules in the core* Linc *system with the induction and/or the co-induction rules.*

Proof. Consider inferring $def\mathcal{L}$ using core Linc and induction rules, the other case being dual. Let $p\bar{x} \stackrel{\triangle}{=} B\bar{x}$ be the definition under consideration: $def\mathcal{L}$ can be inferred from $I\mathcal{L}$ using the body B as the invariant.

$$\dfrac{\overset{\Pi}{B[B/p]\bar{x} \longrightarrow B\bar{x}} \quad B\bar{t}, \Gamma \longrightarrow C}{p\bar{t}, \Gamma \longrightarrow C}\ I\mathcal{L}$$

We can construct the derivation Π by induction on the size of B: since p occurs strictly positively in B by stratification, the only non-trivial base case we need to consider is when we reach the sub-formula $p\bar{t}$ of $B\bar{x}$, in which case we just apply the $I\mathcal{R}$ rule.

3 Examples

We now give some examples, starting with some that make essential use of HOAS.

3.1 Lazy λ-Calculus

We consider an untyped version of the pure λ-calculus with lazy evaluation, following the usual HOAS style, i.e., object-level λ-operator and application are encoded as constants $\mathrm{lam} : (tm \to tm) \to tm$ and $@ : tm \to tm \to tm$, where tm is the syntactic category of object-level λ-terms. The evaluation relation is encoded as the following inductive definition

$$\begin{aligned} M \Downarrow N \stackrel{\mu}{=} \ & (\exists M'.[M = \mathrm{lam}\, M'] \wedge [M = N]) \vee \\ & (\exists M_1 \exists M_2 \exists P.[M = M_1 @ M_2] \wedge M_1 \Downarrow \mathrm{lam}\, P \wedge (P\, M_2) \Downarrow N) \end{aligned}$$

Notice that object-level substitution is realized via β-reduction in the meta-logic.

The notion of *applicative simulation* of λ-expressions can be encoded as the (stratified) co-inductive definition

$$sim\ R\ S \stackrel{\nu}{=} \forall T.R \Downarrow \mathrm{lam}\, T \supset \exists U.S \Downarrow \mathrm{lam}\, U \wedge \forall P.sim\ (T\,P)\ (U\,P).$$

Given this encoding, we can prove the reflexivity property of simulation, i.e., $\forall s.sim\ s\ s$. This is proved co-inductively by using the simulation $\lambda x \lambda y.x = y$. After applying $\forall\mathcal{R}$ and $\mathcal{C}I\mathcal{R}$, it remains to prove the sequents $\longrightarrow s = s$, and

$$x = y \longrightarrow \forall x_1.x \Downarrow \mathrm{lam}\, x_1 \supset (\exists x_2.y \Downarrow \mathrm{lam}\, x_2 \wedge \forall x_3.(x_1\, x_3) = (x_2\, x_3))$$

The first sequent is provable by an application of eq$\mathcal{R}$ rule. The second sequent is proved as follows.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{z\Downarrow \mathrm{lam}\,x_1 \longrightarrow z\Downarrow \mathrm{lam}\,x_1}\;\mathit{init} \qquad \dfrac{\dfrac{}{z\Downarrow \mathrm{lam}\,x_1 \longrightarrow (x_1\,x_3) = (x_1\,x_3)}\;\mathrm{eq}\mathcal{R}}{z\Downarrow \mathrm{lam}\,x_1 \longrightarrow \forall x_3.(x_1\,x_3) = (x_1\,x_3)}\;\forall\mathcal{R}}{z\Downarrow \mathrm{lam}\,x_1 \longrightarrow (z\Downarrow \mathrm{lam}\,x_1 \wedge \forall x_3.(x_1\,x_3) = (x_1\,x_3))}\;\wedge\mathcal{R}}{z\Downarrow \mathrm{lam}\,x_1 \longrightarrow (\exists x_2.z\Downarrow \mathrm{lam}\,x_2 \wedge \forall x_3.(x_1\,x_3) = (x_2\,x_3))}\;\exists\mathcal{R}}{x = y, x\Downarrow \mathrm{lam}\,x_1 \longrightarrow (\exists x_2.y\Downarrow \mathrm{lam}\,x_2 \wedge \forall x_3.(x_1\,x_3) = (x_2\,x_3))}\;\mathrm{eq}\mathcal{L}}{x = y \longrightarrow x\Downarrow \mathrm{lam}\,x_1 \supset (\exists x_2.y\Downarrow \mathrm{lam}\,x_2 \wedge \forall x_3.(x_1\,x_3) = (x_2\,x_3))}\;\supset\mathcal{R}}{x = y \longrightarrow \forall x_1.x\Downarrow \mathrm{lam}\,x_1 \supset (\exists x_2.y\Downarrow \mathrm{lam}\,x_2 \wedge \forall x_3.(x_1\,x_3) = (x_2\,x_3))}\;\forall\mathcal{R}
$$

The transitivity property is expressed as $\forall r \forall s \forall t.\mathit{sim}\ r\ s \wedge \mathit{sim}\ s\ t \supset \mathit{sim}\ r\ t$. Its proof involves co-induction on $\mathit{sim}\ r\ t$ with the simulation $\lambda u \lambda v.\exists w.\mathit{sim}\ u\ w \wedge \mathit{sim}\ w\ v$, followed by case analyses (i.e., $\mathit{def}\mathcal{L}$ and eq$\mathcal{L}$ rules) on $\mathit{sim}\ r\ s$ and $\mathit{sim}\ s\ t$. The rest of the proof is purely logical.

We can also show the existence of divergent terms. Divergence is encoded as follows.

$$
\begin{aligned}
\mathrm{divrg}\ T \overset{\nu}{=} {} & (\exists T_1 \exists T_2.T = (T_1 @ T_2) \wedge \mathrm{divrg}\ T_1) \vee \\
& (\exists T_1 \exists T_2.T = (T_1 @ T_2) \wedge \exists E.T_1 \Downarrow \mathrm{lam}\,E \wedge \mathrm{divrg}\ (E\,T_2)).
\end{aligned}
$$

Let Ω be the term $(\mathrm{lam}\,x.(x @ x)) @ (\mathrm{lam}\,x.(x @ x))$. We show that divrg Ω holds. The proof is straightforward by co-induction using the simulation $S := \lambda s.s = \Omega$. Applying the $\mathcal{CIR}$ produces the sequents $\longrightarrow \Omega = \Omega$ and $T = \Omega \longrightarrow S_1 \ \vee\ S_2$ where

$$S_1 := \exists T_1 \exists T_2.T = (T_1 @ T_2) \wedge (S\,T_1), \text{ and}$$

$$S_2 := \exists T_1 \exists T_2.T = (T_1 @ T_2) \wedge \exists E.T_1 \Downarrow \mathrm{lam}\,E \wedge S\,(E\,T_2).$$

Clearly, only the second disjunct is provable, i.e., by instantiating T_1 and T_2 with the same term $\mathrm{lam}\,x.(x @ x)$, and E with the function $\lambda x.(x @ x)$.

3.2 Lists

Lists over some fixed type α are encoded as the type *lst*, with the usual constructor nil : *lst* for empty list and :: of type $\alpha \to \mathit{lst} \to \mathit{lst}$. We consider here the append predicate for both the finite and infinite case.

Finite lists The usual append predicate on finite lists can be encoded as the inductive definition

$$
\begin{aligned}
\mathrm{app}\ L_1\ L_2\ L_3 \overset{\mu}{=} {} & (L_1 = \mathrm{nil} \wedge L_2 = L_3) \vee \\
& \exists x \exists L_1' \exists L_3'.L_1 = x :: L_1' \wedge L_3 = x :: L_3' \wedge \mathrm{app}\ L_1'\ L_2\ L_3'.
\end{aligned}
$$

Associativity of append is stated formally as

$$\forall l_1 \forall l_2 \forall l_{12} \forall l_3 \forall l_4.(\mathrm{app}\ l_1\ l_2\ l_{12} \wedge \mathrm{app}\ l_{12}\ l_3\ l_4) \supset \forall l_{23}.\mathrm{app}\ l_2\ l_3\ l_{23} \supset \mathrm{app}\ l_1\ l_{23}\ l_4.$$

Proving this formula requires us to prove first that the definition of append is functional, that is,

$$\forall l_1 \forall l_2 \forall l_3 \forall l_4 . \mathrm{app}\ l_1\ l_2\ l_3 \wedge \mathrm{app}\ l_1\ l_2\ l_4 \supset l_3 = l_4 .$$

This is done by induction on l_1, i.e., we apply the $I\mathcal{L}$ rule on app $l_1\ l_2\ l_3$, after the introduction rules for $\forall$ and $\supset$, of course. The invariant in this case is

$$S := \lambda r_1 \lambda r_2 \lambda r_3 . \forall r . \mathrm{app}\ r_1\ r_2\ r \supset r = r_3 .$$

It is a simple case analysis to check that this is the right invariant. Back to our original problem: after applying the introduction rules for the logical connectives in the formula, the problem of associativity is reduced to the following sequent

$$\mathrm{app}\ l_1\ l_2\ l_{12},\ \mathrm{app}\ l_{12}\ l_3\ l_4,\ \mathrm{app}\ l_2\ l_3\ l_{23} \longrightarrow \mathrm{app}\ l_1\ l_{23}\ l_4 . \qquad (1)$$

We then proceed by induction on the list l_1, that is, we apply the $I\mathcal{L}$ rule to the hypothesis app $l_1\ l_2\ l_{12}$. The invariant is simply

$$S := \lambda l_1 \lambda l_2 \lambda l_{12} . \forall l_3 \forall l_4 . \mathrm{app}\ l_{12}\ l_3\ l_4 \supset \forall l_{23} . \mathrm{app}\ l_2\ l_3\ l_{23} \supset \mathrm{app}\ l_1\ l_{23}\ l_4 .$$

Applying the $I\mathcal{L}$ rule, followed by $\vee\mathcal{L}$, to sequent (1) reduces the sequent to the following sub-goals

(i) $S\ l_1\ l_2\ l_{12},\ \mathrm{app}\ l_{12}\ l_3\ l_4,\ \mathrm{app}\ l_2\ l_3\ l_{23} \longrightarrow \mathrm{app}\ l_1\ l_{23}\ l_4,$
(ii) $(l_1 = \mathrm{nil} \wedge l_2 = l_3) \longrightarrow S\ l_1\ l_2\ l_3,$
(iii) $\exists x, l_1', l_3' . l_1 = x :: l_1' \wedge l_3 = x :: l_3' \wedge S\ l_1'\ l_2\ l_3' \longrightarrow S\ l_1\ l_2\ l_3$

The proof for the second sequent is straightforward. The first sequent reduces to

$$\mathrm{app}\ l_{12}\ l_3\ l_4, \mathrm{app}\ l_{12}\ l_3\ l_{23} \longrightarrow \mathrm{app}\ \mathrm{nil}\ l_{23}\ l_4 .$$

This follows from the functionality of append and $I\mathcal{R}$. The third sequent follows by case analysis. Of course, these proofs could have been simplified by using a *derived* principle of *structural* induction. While this is easy to do, we have preferred here to use the primitive $I\mathcal{L}$ rule.

Infinite lists The append predicate on infinite lists is defined via co-recursion, that is, we define the behaviour of *destructor operations* on lists (i.e., taking the head and the tail of the list). In this case we never construct explicitly the result of appending two lists, rather the head and the tail of the resulting lists are computed as needed. The co-recursive append requires case analysis on all arguments.

$$\begin{aligned} \mathrm{coapp}\ L_1\ L_2\ L_3 \overset{\nu}{=} {} & (L_1 = \mathrm{nil} \wedge L_2 = \mathrm{nil} \wedge L_3 = \mathrm{nil}) \vee \\ & (L_1 = \mathrm{nil} \wedge \exists x \exists L_2' \exists L_3' . L_2 = x :: L_2' \wedge L3 = x :: L_3' \ \wedge\ \mathrm{coapp}\ \mathrm{nil}\ L_2'\ L_3') \\ & \vee\ (\exists x \exists L_1' \exists L_3' . L_1 = x :: L_1' \wedge L_3 = x :: L_3' \ \wedge\ \mathrm{coapp}\ L_1'\ L_2\ L_3'). \end{aligned}$$

The corresponding associativity property is stated analogously to the inductive one and the main statement reduces to proving the sequent

$$\mathrm{coapp}\ l_1\ l_2\ l_{12},\ \mathrm{coapp}\ l_{12}\ l_3\ l_4,\ \mathrm{coapp}\ l_2\ l_3\ l_{23} \longrightarrow \mathrm{coapp}\ l_1\ l_{23}\ l_4 .$$

We apply the $\mathcal{CIR}$ rule to coapp l_1 l_{23} l_4, using the simulation

$$S := \lambda l_1 \lambda l_2 \lambda l_{12}.\exists l_{23} \exists l_3 \exists l_4.\text{coapp } l_{12}\ l_3\ l_4 \wedge \text{ coapp } l_2\ l_3\ l_{23} \wedge \text{ coapp } l_1\ l_{23}\ l_4.$$

Subsequent steps of the proof involve mainly case analysis on coapp l_{12} l_3 l_4. As in the inductive case, we have to prove the sub-cases when l_{12} is nil. However, unlike in the former case, case analyses on the arguments of coapp suffices.

4 Cut-Elimination

A central result of our work is cut-elimination, from which consistency of the logic follows. Gentzen's classic proof of cut-elimination for first-order logic uses an induction on the size of the cut formula, i.e., the number of logical connectives in the formula. The cut-elimination procedure consists of a set of reduction rules that reduce a cut of a compound formula to cuts on its sub-formulae of smaller size. In the case of Linc, the use of induction/co-induction complicates the reduction of cuts. Consider for example a cut involving the induction rules

$$\cfrac{\cfrac{\begin{matrix}\Pi_1\\ \Delta \longrightarrow Bt\end{matrix}}{\Delta \longrightarrow pt}\ \mathcal{IR} \qquad \cfrac{\begin{matrix}\Pi_B\\ B[S/p]y \longrightarrow Sy\end{matrix} \quad \begin{matrix}\Pi\\ St, \Gamma \longrightarrow C\end{matrix}}{pt, \Gamma \longrightarrow C}\ \mathcal{IL}}{\Delta, \Gamma \longrightarrow C}\ mc$$

There are at least two problems in reducing this cut. First, any permutation upwards of the cut will necessarily involve a cut with S that can be of larger size than p, and hence a simple induction on the size of cut formula will not work. Second, the invariant S does not appear in the conclusion of the left premise of the cut. The latter means that we need to transform the left premise so that its end sequent will agree with the right premise. Any such transformation will most likely be *global*, and hence simple induction on the height of derivations will not work either. We define a proof transformation that we call *unfolding* to deal with the cut involving $\mathcal{IL}/\mathcal{IR}$ and $\mathcal{CIR}/\mathcal{CIL}$ pairs.

In the following definition, we refer to a premise of a rule as a *minor premise* if it is the left-premise of $\supset\mathcal{L}$ or $\mathcal{IL}$, or the right-premise of $\mathcal{CIR}$ or *mc*, otherwise it is a *major premise*. A derivation of a minor [major] premise is a *minor [major] premise derivation*. To simplify the definitions of unfolding, we restrict the *init*-rule to the atomic form. Non-atomic *init*-rule can easily be shown to be derivable using only structural rules, logical rules and atomic *init*. We shall refer to this non-atomic *init* derivation as Π^{Id}. We use the notation $\Pi\theta$ to denote the application of the substitution θ to Π, which amounts to applying the substitution to every sequent in Π.

Definition 4. Inductive unfolding. *Let $p\bar{x} \overset{\mu}{=} B\bar{x}$ be an inductive definition. Suppose we are given a derivation Π of $\Gamma \longrightarrow C$ where each occurrence of p in C is strictly positive, and a derivation Π_S of $B[S/p]\bar{x} \longrightarrow S\bar{x}$, for some closed term S. We define the derivation $\mu(\Pi,\Pi_S)$ of $\Gamma \longrightarrow C[S/p]$ as follows. If $C[S/p] = C$, then $\mu(\Pi,\Pi_S) = \Pi$. Otherwise, we define $\mu(\Pi,\Pi_S)$ based on the last rule in Π.*

1. *If* Π *ends with init on atom* $p\bar{t}$, *then* $\mu(\Pi,\Pi_S)$ *is the derivation*

$$\dfrac{\overset{\Pi_S}{B[S/p]\bar{x} \longrightarrow S\bar{x}} \quad \overset{\Pi^{Id}}{S\bar{t} \longrightarrow S\bar{t}}}{p\bar{t} \longrightarrow S\bar{t}}\ I\mathcal{L}$$

2. *If* Π *ends with* $\supset \mathcal{R}$

$$\dfrac{\overset{\Pi'}{\Gamma, C_1 \longrightarrow C_2}}{\Gamma \longrightarrow C_1 \supset C_2} \supset \mathcal{R} \quad \text{then } \mu(\Pi,\Pi_S) \text{ is} \quad \dfrac{\overset{\mu(\Pi',\Pi_S)}{\Gamma, C_1 \longrightarrow C_2[S/p]}}{\Gamma \longrightarrow C_1 \supset C_2[S/p]} \supset \mathcal{R}$$

Note that the restriction on the occurrence of p *in* C *implies that* $(C_1 \supset C_2)[S/p] = C_1 \supset C_2[S/p]$.

3. *If* Π *ends with* $I\mathcal{R}$ *on* $p\bar{u}$, *for some terms* $\bar{u}$,

$$\dfrac{\overset{\Pi'}{\Gamma \longrightarrow B\bar{u}}}{\Gamma \longrightarrow p\bar{u}}\ I\mathcal{R} \quad \text{then } \mu(\Pi,\Pi_S) \text{ is} \quad \dfrac{\overset{\mu(\Pi',\Pi_S)}{\Gamma \longrightarrow B[S/p]\bar{u}} \quad \overset{\Pi_S[\bar{u}/\bar{x}]}{B[S/p]\bar{u} \longrightarrow S\bar{u}}}{\Gamma \longrightarrow S\bar{u}}\ mc$$

4. *Otherwise, if* Π *ends with any other rule, with the minor premise derivations* $\Xi_1,\ldots,\Xi_m$ *for some* $m \geq 0$ *and the major premise derivations* $\{\Pi_i\}_{i\in I}$ *for some index set* I, *then* $\mu(\Pi,\Pi_S)$ *ends with the same rule, with the same minor premises and the major premises* $\{\mu(\Pi_i,\Pi_S)\}_{i\in I}$.

Definition 5. Co-inductive unfolding. *Let* $p\bar{x} \overset{\nu}{=} B\bar{x}$ *be a co-inductive definition. Let* C *be a formula in which every occurrence of* p *is strictly positive. Suppose we are given a derivation* Π *of* $\Gamma \longrightarrow C[S/p]$ *and a derivation* Π_S *of* $S\bar{x} \longrightarrow B[S/p]\bar{x}$, *for some closed term* S. *We define the derivation* $\nu(\Pi,\Pi_S)$ *of* $\Gamma \longrightarrow C$ *as follows. If* $C[S/p] = C$, *then* $\nu(\Pi,\Pi_S) = \Pi$. *If* $C = p\bar{t}$ *for some terms* $\bar{t}$, *then* $C[S/p]\bar{t} = S\bar{t}$ *and* $\nu(\Pi,\Pi_S)$ *is the derivation*

$$\dfrac{\overset{\Pi}{\Gamma \longrightarrow S\bar{t}} \quad \overset{\Pi_S}{S\bar{x} \longrightarrow B\bar{x}}}{\Gamma \longrightarrow p\bar{t}}\ CI\mathcal{R}$$

Otherwise, we define $\nu(\Pi,\Pi_S)$ *based on the last rule in* Π. *If* Π *ends with* $\supset \mathcal{R}$

$$\dfrac{\overset{\Pi'}{\Gamma, C_1 \longrightarrow C_2[S/p]}}{\Gamma \longrightarrow C_1 \supset C_2[S/p]} \supset \mathcal{R} \quad \text{then } \mu(\Pi,\Pi_S) \text{ is} \quad \dfrac{\overset{\nu(\Pi',\Pi_S)}{\Gamma, C_1 \longrightarrow C_2}}{\Gamma \longrightarrow C_1 \supset C_2} \supset \mathcal{R}.$$

If Π *ends with any other rule, with the minor premise derivations* $\Xi_1,\ldots,\Xi_m$ *for some* $m \geq 0$ *and the major premise derivations* $\{\Pi_i\}_{i\in I}$ *for some index set* I, *then* $\mu(\Pi,\Pi_S)$ *also ends with the same rule, with the same minor premises and the major premises* $\{\mu(\Pi_i,\Pi_S)\}_{i\in I}$.

Our proof of cut-elimination uses the technique of reducibility originally due to Tait. The method was applied by Martin-Löf [12] to the setting of natural deduction, and to sequent calculus by McDowell and Miller for the logic $FO\lambda^{\Delta\mathbb{N}}$ [13]. The original idea of Martin-Löf was to use derivations directly as a measure by defining a well-founded ordering on them. The basis for the latter relation is a set of reduction rules that are used to eliminate the applications of cut rule. For the cases involving logical connectives, the cut-reduction rules used to prove the cut-elimination for Linc are the same to those of $FO\lambda^{\Delta\mathbb{IN}}$. The crucial cases involving (co)-induction are given in the following definition. For simplicity of presentation, we assume the reduction involves the leftmost and the rightmost premise derivations of *mc*.

Definition 6. Cut-reduction. *Let* Ξ *be the derivation*

$$\dfrac{\overset{\Pi_1}{\Delta_1 \longrightarrow D_1} \quad \cdots \quad \overset{\Pi_n}{\Delta_n \longrightarrow D_n} \quad \overset{\Pi}{D_1,\ldots,D_n,\Gamma \longrightarrow C}}{\Delta_1,\ldots,\Delta_n,\Gamma \longrightarrow}\ mc.$$

Case $*/I\mathcal{L}$**:** *If* $D_1 = p\bar{t}$*, where* $p\bar{x} \overset{\mu}{=} B\bar{x}$*, and* Π *ends with* $I\mathcal{L}$ *on the cut formula* $p\bar{t}$

$$\dfrac{\overset{\Pi_S}{B[S/p]\bar{x} \longrightarrow S\bar{x}} \quad \overset{\Pi'}{S\bar{t},D_2,\ldots,D_n,\Gamma \longrightarrow C}}{p\bar{t},D_2,\ldots,D_n,\Gamma \longrightarrow C}\ I\mathcal{L}$$

then Ξ *reduces to*

$$\dfrac{\overset{\mu(\Pi_1,\Pi_S)}{\Delta_1 \longrightarrow S\bar{t}} \quad \overset{\Pi_2}{\Delta_2 \longrightarrow D_2} \quad \cdots \quad \overset{\Pi_n}{\Delta_n \longrightarrow D_n} \quad \overset{\Pi'}{S\bar{t},D_2,\ldots,D_n,\Gamma \longrightarrow C}}{\Delta_1,\ldots,\Delta_n,\Gamma \longrightarrow C}\ mc$$

Case $CI\mathcal{R}/CI\mathcal{L}$**:** *If* $D_1 = p\bar{t}$*, where* $p\bar{x} \overset{\nu}{=} B\bar{x}$*, and* Π_1 *and* Π *are*

$$\dfrac{\overset{\Pi_1'}{\Delta_1 \longrightarrow S\bar{t}} \quad \overset{\Pi_S}{S\bar{x} \longrightarrow B[S/p]\bar{x}}}{\Delta_1 \longrightarrow p\bar{t}}\ CI\mathcal{R} \qquad \dfrac{\overset{\Pi'}{B\bar{t},D_2,\ldots,D_n,\Gamma \longrightarrow C}}{p\bar{t},D_2,\ldots,D_n,\Gamma \longrightarrow C}\ CI\mathcal{L}$$

then Ξ *reduces to*

$$\dfrac{\dfrac{\overset{\Pi_1'}{\Delta_1 \longrightarrow S\bar{t}} \quad \overset{\Xi_1}{S\bar{t} \longrightarrow B\bar{t}}}{\Delta_1 \longrightarrow B\bar{t}}\ mc \quad \cdots \quad \overset{\Pi_n}{\Delta_n \longrightarrow D_n} \quad \overset{\Pi'}{B\bar{t},\ldots,D_n,\Gamma \longrightarrow C}}{\Delta_1,\ldots,\Delta_n,\Gamma \longrightarrow C}\ mc$$

where $\Xi_1 = \nu(\Pi_S,\Pi_S)[\bar{t}/\bar{x}]$.

Notice that these two reductions are not symmetric. This is because we use an asymmetric measure to show the termination of cut-reduction, that is, the complexity of cut is always reduced on the right premise. The difficulty in getting a symmetric measure, in the presence of contraction and implication (in the body of definition), is already observed in [24].

To show the termination of cut-reduction, we define two orderings on derivations: *normalizability* and *reducibility* (called computability in [12]). The well-foundedness of the normalizability ordering immediately implies that the cut-elimination process terminates. Reducibility is a superset of normalizability and hence its well-foundedness implies the well-foundedness of normalizability. The main part of the proof lies in showing that all derivations in Linc are reducible, and hence normalizable. This is stated in the Lemma 1, of which cut-elimination is a simple corollary.

Lemma 1. *For any derivation* Π *of* $B_1,\ldots,B_n,\Gamma \longrightarrow C$*, reducible derivations* $\Pi_1,\ldots,\Pi_n$ *of* $\Delta_1 \longrightarrow B_1,\ldots,\Delta_n \longrightarrow B_n$ $(n \geq 0)$*, and substitutions* $\delta_1,\ldots,\delta_n,\gamma$ *such that* $B_i\delta_i = B_i\gamma$*, for every* $i \in \{1,\ldots,n\}$*, the following derivation* Ξ *is reducible.*

$$\frac{\begin{array}{c}\Pi_1\delta_1\\ \Delta_1\delta_1 \longrightarrow B_1\delta_1\end{array} \quad \cdots \quad \begin{array}{c}\Pi_n\delta_n\\ \Delta_n\delta_n \longrightarrow B_n\delta_n\end{array} \quad \begin{array}{c}\Pi\gamma\\ B_1\gamma,\ldots,B_n\gamma,\Gamma\gamma \longrightarrow C\gamma\end{array}}{\Delta_1\delta_1,\ldots,\Delta_n\delta_n,\Gamma\gamma \longrightarrow C\gamma}\ mc$$

The proof proceeds by induction on the height of Π with subordinate inductions on n and on the (well-founded) reduction tree of $\Pi_1,\ldots,\Pi_n$. We give a general idea of the proof for the cases $*/\mathcal{IL}$ and $\mathcal{CIR}/\mathcal{CIL}$ in Definition 6, and refer to [30] for full details. In the following description, we refer to Definition 6 for the particular shapes of the derivations Π_1 and Π. In the $*/\mathcal{IL}$ case, it is sufficient to show that given the reducibility of Π_1, the unfolding derivation $\mu(\Pi_1,\Pi_S)$ is still reducible. This is done by induction on the construction of $\mu(\Pi_1,\Pi_S)$. The non-trivial case is when new cuts (*mc*) is introduced. But here we see that this instance of *mc* is always cutting with Π_S, and hence by the outer induction hypothesis (Π_S is of smaller height than Π) this instance of *mc* is reducible. The $\mathcal{CIR}/\mathcal{CIL}$ case is more complicated. In addition to showing that the co-inductive unfolding preserves reducibility, we also need to show that the unfolded derivation $\nu(\Pi_S,\Pi_S)$ is "closed" with respect to cut, that is, for every reducible derivation Ψ of $\Delta' \longrightarrow S\,\bar{u}$, the resulting derivation obtained by cutting Ψ with $\nu(\Pi_S,\Pi_S)[\bar{u}/\bar{x}]$ is reducible. This case is dealt with by building into the notion of reducibility this closure condition.

5 Related Work

Linc has been designed as an *intentionally* weak logical framework to be used as a meta-language for reasoning over deductive systems encoded via HOAS. In particular, it can be seen as the meta-theory of the simply typed λ-calculus, in the same sense in which Schürmann's $\mathcal{M}_\omega$ [27] is the meta-theory of *LF* [10]. $\mathcal{M}_\omega$ is a constructive first-order logic, whose quantifiers range over possibly open LF objects over a signature. By the adequacy of the encoding, the proof of the existence of the appropriate LF object(s) guarantees the proof of the corresponding object-level property. It must be remarked that $\mathcal{M}_\omega$ does not support co-induction yet. However, LF can be used directly to specify an inductive meta-theorem as a relation between judgements, with a logic programming interpretation providing the operational semantics.

Of course, there is a long association between mathematical logic and inductive definitions [2] and in particular with proof-theory, possibly the earliest relevant entry being Martin-Löf's original formulation of the theory of *iterated inductive definitions* [12]. From the impredicative encoding of inductive types [4] and the introduction of (co)recursion [7] in system F, (co)inductive types became common [16] and made it into type-theoretic proof assistants such as Coq [19], first via a primitive recursive operator, but eventually in the let-rec style of functional programming languages, as in Gimenez's *Calculus of Infinite Constructions* [8]; here termination (resp. guardedness) is ensured by a syntactic check (see also [1]). Note that this has severe limitations (e.g., in the possibility of using lemmas in the body of a guarded proof) that do not applies to our approach. Circular proofs are also connected with the emerging proof-theory of *fixed point logics* and *process calculi* [23, 28, 29], in particular w.r.t. the relation between systems with local and global induction, that is, between fixed point vs. well-founded and guarded induction (i.e. circular proofs).

In higher order logic (co)inductive definitions are obtained via the usual Tarski fixed point constructions, as realized for example in Isabelle/HOL [20]. As we mentioned before, those approaches are at odd with HOAS even at the level of the syntax. Several compromises have been proposed: the *Theory of Contexts* [11] (ToC) marries *Weak* HOAS with an axiomatic approach encoding basic properties of names. *Hybrid* [3] is a λ-calculus on top of Isabelle/HOL which provides the user a *Full* HOAS syntax, compatible with a classical (co)-inductive setting. Linc improves on the latter on several counts. First it disposes of Hybrid notion of *abstraction*, which is used to carve out the "parametric" function space from the full HOL space. Moreover it is not restricted to second-order abstract syntax, as the current Hybrid version is (and as ToC cannot escape from being). Finally, at higher types, reasoning via $def\mathcal{L}$ is more powerful than inversion: for example $\forall y.\lambda x.y \neq \lambda x.0$ is provable in Linc, but fails both in Isabelle/HOL and Coq – the latter for extensionality reasons.

6 Conclusion and Future Work

We have presented a proof theoretical treatment of both induction and co-induction in a sequent calculus compatible with HOAS encodings. The proof principle underlying the explicit proof rules is basically fixed point (co)induction. Our proof system is, as far as we know, the first which incorporates a co-induction proof rule and still preserves cut-elimination. We have shown several examples where informal (co)inductive proofs using invariants and simulations are reproduced formally in Linc. Consistency of the logic is an easy consequence of cut-elimination.

We currently have two prototype implementations of Linc. The one in the Hybrid system [3, 18] is better characterized as an approximation: definitional reflection is mimicked by the elimination rules of (co)inductive definitions, which also provides (co)induction principles, while the Hybrid λ-calculus takes care of the freeness properties: notwithstanding the limitations mentioned in Section 5, the implementation has the benefit of inheriting all the automation of Isabelle/HOL on whose top Hybrid is realized. The second is a direct implementation of Linc rules in λProlog, with a Java graphical user interface (available on the web at `http://www.lix.polytechnique.fr/~tiu`). This prototype is currently limited to be a proof-checker. A serious implementation would require more study on the proof search properties of Linc. It is true that with induction and co-induction there is no hope of automation in general. Nevertheless, a large subset of the logic may still admit some uniformity in proof search.

On the theoretical level, we conjecture that the proviso in the $\mathcal{CIR}$ rule can be eliminated. Similarly, we can loosen the stratification condition for example in the sense of *local* stratification and of terminating higher-order logic programs [22], possibly allowing to encode proofs such as type preservation in operational semantics directly in Linc rather than with the 2-level approach [14, 18].

Another interesting problem to investigate is the connection with *circular proofs*, which is particularly attractive from the viewpoint of proof search, both inductively and co-inductively. This could be realized by directly proving a cut-elimination result for a logic where circular proofs, under termination and guardedness conditions completely replace (co)inductive rules. Alternatively, we could reduce "global" proofs in such a system to "local" proofs in Linc, similarly to [29]. Finally, extensions of Linc, for example in the direction of $FO\lambda^{\nabla}$ [17] are worth investigating.

Acknowledgements The Linc logic was developed in collaboration with Dale Miller. Alberto Momigliano has been supported by EPSRC grant GR/M98555 and partly by the MRG project (IST-2001-33149), funded by the EC under the FET proactive initiative on Global Computing. Alwen Tiu has been supported in part by NSF grants CCR-9912387, INT-9815645, and INT-9815731 and LIX at École polytechnique.

References

[1] A. Abel and T. Altenkirch. A predicative strong normalisation proof for a λ-calculus with interleaving inductive types. In T. Coquand, P. Dybjer, B. Nordström, and J. Smith, editors, *TYPES '99*, vol 1956 of *LNCS*, pp 21–40. Springer-Verlag, 2000.

[2] P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, vol 90 of *Studies in Logic and the Foundations of Mathematics*, pp 739–782. North-Holland, Amsterdam, 1977.

[3] S. Ambler, R. Crole, and A. Momigliano. Combining higher order abstract syntax with tactical theorem proving and (co)induction. In V. A. Carreño, editor, *TPHOLs 2002*, vol 2342 of *LNCS*. Springer Verlag, 2002.

[4] C. Bohm and A. Berarducci. Automatic synthesis of typed lambda programs on term algebras. *Theoretical Computer Science*, 39(2-3):135–153, Aug. 1985.

[5] R. L. Crole. Lectures on [Co]Induction and [Co]Algebras. Technical Report 1998/12, Department of Mathematics and Computer Science, University of Leicester, 1998.

[6] J. Despeyroux and A. Hirschowitz. Higher-order abstract syntax with induction in Coq. In *Fifth Conference on Logic Programming and Automated Reasoning*, pp 159–173, 1994.

[7] H. Geuvers. Inductive and coinductive types with iteration and recursion. In B. Nordström, K. Pettersson, and G. Plotkin, editors, *Proceedings of Workshop on Types for Proofs and Programs*, pp 193–217. Dept. of Computing Science, Chalmers Univ. of Technology, 1992.

[8] E. Giménez. *Un Calcul de Constructions Infinies et son Application a la Verification des Systemes Communicants*. PhD thesis, Ecole Normale Supérieure de Lyon, Dec. 1996.

[9] J.-Y. Girard. A fixpoint theorem in linear logic. Email to the linear@cs.stanford.edu mailing list, February 1992.

[10] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

[11] F. Honsell, M. Miculan, and I. Scagnetto. An axiomatic approach to metareasoning on systems in higher-order abstract syntax. In *Proc. ICALP'01*, number 2076 in LNCS, pp 963–978. Springer-Verlag, 2001.

[12] P. Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, vol 63 of *Studies in Logic and the Foundations of Mathematics*, pp 179–216. North-Holland, 1971.

[13] R. McDowell and D. Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.

[14] R. McDowell and D. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic*, 3(1):80–136, January 2002.

[15] R. McDowell, D. Miller, and C. Palamidessi. Encoding transition systems in sequent calculus. *TCS*, 294(3):411–437, 2003.

[16] N. P. Mendler. Inductive types and type constraints in the second order lambda calculus. *Annals of Pure and Applied Logic*, 51(1):159–172, 1991.

[17] D. Miller and A. Tiu. A proof theory for generic judgments: An extended abstract. Proceedings of LICS'03, January 2003.

[18] A. Momigliano and S. Ambler. Multi-level meta-reasoning with higher order abstract syntax. In A. Gordon, editor, *FOSSACS'03*, vol 2620 of *LNCS*, pp 375–392. Springer, 2003.

[19] C. Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. In M. Bezem and J. Groote, editors, *TCLA*, pp 328–345, Springer-Verlag LNCS 664, 1993.

[20] L. C. Paulson. Mechanizing coinduction and corecursion in higher-order logic. *Journal of Logic and Computation*, 7(2):175–204, Mar. 1997.

[21] F. Pfenning. Logical frameworks. In *Handbook of Automated Reasoning*, pp 1063–1147. MIT Press, 2001.

[22] E. Rohwedder and F. Pfenning. Mode and termination analysis for higher-order logic programs. In *Proc. of the European Symposium on Programming*, pp 296–310, April 1996.

[23] L. Santocanale. A calculus of circular proofs and its categorical semantics. In M. Nielsen and U. Engberg editors, *Proc. of FoSSaCS 2002*, pp 357–371, Grenoble, Apr. 2002, Springer-Verlag LNCS 2303.

[24] P. Schroeder-Heister. Cut-elimination in logics with definitional reflection. In D. Pearce and H. Wansing, editors, *Nonclassical Logics and Information Processing*, vol 619 of *LNCS*, pp 146–171. Springer, 1992.

[25] P. Schroeder-Heister. Definitional reflection and the completion. In R. Dyckhoff, editor, *Proceedings of WELP'93*, pp 333–347. Springer-Verlag LNAI 798, 1993.

[26] P. Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *Eighth Annual Symposium on Logic in Computer Science*, pp 222–232. IEEE Press, June 1993.

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

[28] A. K. Simpson. Compositionality via cut-elimination: Hennessy-Milner logic for an arbitrary GSOS. In D. Kozen, editor, *Proceedings of LICS'95*, pp 420–430, San Diego, California, June 1995. IEEE Computer Society Press.

[29] C. Spenger and M. Dams. On the structure of inductive reasoning: Circular and tree-shaped proofs in the μ-calculus. In A. Gordon, editor, *FOSSACS'03*, vol 2620 of *LNCS*, pp 425–440,. Springer Verlag, 2003.

[30] A. Tiu. Cut-elimination for a logic with induction and co-induction. Draft, available via `http://www.cse.psu.edu/~tiu/lce.pdf`, Sept. 2003.