# Reflective $\boldsymbol{\lambda}$-Calculus

Jesse Alt and Sergei Artemov[⋆]

Cornell University,
Ithaca, NY 14853, U.S.A.
{jma35, artemov}@cs.cornell.edu

**Abstract.** We introduce a general purpose typed $\lambda$-calculus $\boldsymbol{\lambda}^\infty$ which contains intuitionistic logic, is capable of internalizing its own derivations as $\lambda$-terms and yet enjoys strong normalization with respect to a natural reduction system. In particular, $\boldsymbol{\lambda}^\infty$ subsumes the typed $\lambda$-calculus. The Curry-Howard isomorphism converting intuitionistic proofs into $\lambda$-terms is a simple instance of the internalization property of $\boldsymbol{\lambda}^\infty$. The standard semantics of $\boldsymbol{\lambda}^\infty$ is given by a proof system with proof checking capacities. The system $\boldsymbol{\lambda}^\infty$ is a theoretical prototype of reflective extensions of a broad class of type-based systems in programming languages, provers, AI and knowledge representation, etc.

## 1 Introduction

According to the Curry-Howard isomorphism, the calculus of intuitionistic propositions (types) and the calculus of typed $\lambda$-terms (proof terms) constitute a pair of isomorphic though distinct structures. Combining those logical and computational universes has been considered a major direction in theoretical logic and applications (propositions-as-types, proofs-as-programs paradigms, etc., cf. [5,7,11,12]). Modern computational systems are often capable of performing logical derivations, formalizing their own derivations and computations, proof and type checking, normalizing $\lambda$-terms, etc. A basic theoretical prototype of such a system would be a long anticipated joint calculus of propositions (types) and typed $\lambda$-terms (proofs, programs). There are several natural requirements for such a calculus raising from the intended reading of the type assertion $t{:}F$ as a proposition *t is a proof of F*.

1. A type assertion $t{:}F$ should be treated as a legitimate proposition (hence a type). The intended informal semantics of $t : F$ could be *t is a proof of F* or *t has type F*. In particular, $t : F$ could participate freely in constructing new types (propositions). Such a capacity would produce types containing $\lambda$-terms (proofs, programs) inside. For example, the following types should be allowed $t : F \to s : G$, $t : F \wedge s : G$, $s : (t : F)$, etc. In programmistic terms this means a possibility of encoding computations inside types. A system should contain

[⋆] The research described in this paper was supported in part by ARO under the MURI program "Integrated Approach to Intelligent Systems", grant DAAH04-96-1-0341, by DARPA under program LPE, project 34145.

both the **intuitionistic logic** (as a calculus of types) and the **simply typed $\lambda$-calculus**.

2. Such a system should contain the **reflection principle** $t : F \to F$ representing a fundamental property of proofs *if $t$ is a proof of $F$ then $F$ holds.* An alternative type-style reading of this principle says *if $t$ of type $F$ then $F$ is inhabited.*

3. A system should contain the explicit **proof checking** (the **type checking**) **operation** "!" and the principle $t{:}F \to {!t}{:}(t{:}F)$. The informal reading of "!" is that given a term $t$ the term $!t$ describes a computation verifying $t{:}F$ (i.e. that $t$ is a proof of $F$, or that $t$ has type $F$). The proof checking and the reflection principles enable us to naturally connect relevant types up and down from a given one. Such a possibility has been partly realized by means of modal logic in modal principles $\Box F \to F$ and $\Box F \to \Box\Box F$ respectively, though this presentation lacks the explicit character of typed terms (cf. [3,6]).

4. A system should accommodate the Curry-Howard isomorphism that maps natural derivations in intuitionistic logic to well defined typed $\lambda$-terms. A fundamental closure requirement suggests generalizing the Curry-Howard isomorphism to the **Internalization Property** of the system: *if $A_1, A_2, \ldots, A_n \vdash B$ then for fresh variables $x_1, x_2, \ldots, x_n$ and some term $t(x_1, x_2, \ldots, x_n)$*

$$x_1{:}A_1, x_2{:}A_2, \ldots, x_n{:}A_n \vdash t(x_1, x_2, \ldots, x_n){:}B.$$

5. Desirable properties of terms in such system include strong normalizability with respect to $\beta$-reduction, projection reductions, etc., as well as other natural properties of typed $\lambda$-terms.

The main goal of this paper is to find such a system. The reflective $\lambda$-calculus $\boldsymbol{\lambda}^{\infty}$ introduced below is the minimal system that meets these specifications.

The Logic of Proofs **LP** ([1,2,3]) where proofs are represented by advanced combinatory terms (called *proof polynomials*), satisfy properties 1–4 above. The property 5 is missing in **LP** since the combinatory format does not admit reductions and thus does not leave a room for nontrivial normalizations.

Reflexive $\lambda$-terms correspond to proof polynomials over intuitionistic logic in the basis $\{\to, \wedge\}$. Those polynomials are built from constants and variables by two basic operations: "$\cdot$" (application), "!" (proof checker). The logical identities for those proof polynomials are described by the corresponding fragment of the Logic of Proofs **LP**. In **LP** the usual set of type formation rules is extended by a new one: given a proof polynomial $p$ and formula $F$ build a new type (propositional formula) $p{:}F$. Valid identities in this language are given by propositional formulas in the extended language generated by the calculus having axioms

*A0. Axiom schemes and rules of intuitionistic logic*
*A1.* $t{:}F \to F$ *"verification"*
*A2.* $t{:}(F \to G) \to (s{:}F \to (t{\cdot}s){:}G)$ *"application"*
*A3.* $t{:}F \to {!t}{:}(t{:}F)$ *"proof checker"*

Rule *axiom necessitation*:

*given a constant $c$ and an axiom $F$ from A0–A3 infer $c{:}F$.*

The standard semantics of proof polynomials is operations on proofs in a proof system containing intuitionistic logic. The operation "application" has the usual meaning as applying a proof $t$ of $F \to G$ to a proof $s$ of $F$ to get a proof $t \cdot s$ of $G$. The "proof checking" takes a proof $t$ of $F$ and returns a proof $!t$ of the sentence $t$ *is a proof of* $F$. Logic of Proofs in the combinatory format finds applications in those areas where modal logics, epistemic logics, logics of knowledge work.

The $\lambda$-version of the whole class of proof polynomials is not normalizing. Indeed, the important feature of proof polynomials is their polymorphism. In particular, a variable can be assigned a finite number of arbitrary types. If a variable $x$ has two types $A$ and $A \to A$ then the $\lambda$-term $(\lambda x.xx) \cdot (\lambda x.xx)$ does not have a normal form. In order to find a natural normalizing subsystem we limit our considerations to $\lambda$-terms for single-conclusion (functional) proof systems, i.e. systems where each proof proves only one theorem. Such terms will have unique types. Proof polynomials in the combinatory format for functional proof systems has been studied in [8,9].

Requirements 1–4 above in the $\lambda$-term format inevitably lead to the system $\boldsymbol{\lambda}^{\infty}$ below. The key idea of having nested copies of basic operations on $\lambda$-terms in $\boldsymbol{\lambda}^{\infty}$ can be illustrated by the following examples. By the Internalization Property, a propositional derivation $A \to B, A \vdash B$ yields a $\lambda$-term derivation[1]

$$x_1{:}(A \to B), y_1{:}A \vdash (x_1 \circ y_1){:}B,$$

which yields the existence of a term $t(x_2, y_2)$ such that

$$x_2{:}x_1{:}(A \to B), y_2{:}y_1{:}A \vdash t(x_2, y_2){:}(x_1 \circ y_1){:}B.$$

(here and below ":" is meant to be right-associative). Naturally, we opt to accept such $t$ as a basic operation and call it $\circ^2$. The defining identity for $\circ^2$ is thus

$$x_2{:}x_1{:}(A \to B), y_2{:}y_1{:}A \vdash (x_2 \circ^2 y_2){:}(x_1 \circ y_1){:}B,$$

or, in the alternative notation $t^F$ for type assigmnents

$$x_2^{x_1^{A \to B}}, y_2^{y_1^{A}} \vdash (x_2 \circ^2 y_2)^{(x_1 \circ y_1)^B}.$$

Likewise, the Internalization Property yields the existence of the operation $\circ^3$ such that

$$x_3{:}x_2{:}x_1{:}(A \to B), y_3{:}y_2{:}y_1{:}A \vdash (x_3 \circ^3 y_3){:}(x_2 \circ^2 y_2){:}(x_1 \circ y_1){:}B$$

Similar nested counterparts appear for $\lambda$-abstraction, pairing, and projections.

[1] Here "$\circ$" denotes application on $\lambda$-terms.

New series of operations are generated by the reflection and proof checking principles. The reflection has form $t : A \vdash A$. The internalized version of this derivation gives a unary operation $\Downarrow$ such that

$$x_1 : t : A \vdash \Downarrow x_1 : A.$$

Likewise, for some unary operation $\Downarrow^{\mathbf{2}}$

$$x_2 : x_1 : t : A \vdash \Downarrow^{\mathbf{2}} x_2 : \Downarrow x_1 : A,$$

etc.

The proof checking derivation $t : A \vdash\, !t : t : A$ gives rise to a unary operation $\Uparrow$ such that

$$x_1 : t : A \vdash \Uparrow x_1 : !t : t : A.$$

Further application of Internalization produces $\Uparrow^{\mathbf{2}}$, $\Uparrow^{\mathbf{3}}$, etc. such that

$$x_2 : x_1 : t : A \vdash \Uparrow^{\mathbf{2}} x_2 : \Uparrow x_1 : !t : t : A,$$

$$x_3 : x_2 : x_1 : t : A \vdash \Uparrow^{\mathbf{3}} x_3 : \Uparrow^{\mathbf{2}} x_2 : \Uparrow x_1 : !t : t : A,$$

etc.

Theorem 1 below demonstrates that such a set of nested operations on λ-terms is in fact necessary and sufficient to guarantee the Internalization Property for the whole of $\boldsymbol{\lambda}^{\infty}$. Therefore, there is nothing arbitrary in this set, and nothing is missing there either.

Normalization of terms depends upon a choice of reductions, which may vary from one application to another. In this paper we consider a system of reductions motivated by our provability reading of $\boldsymbol{\lambda}^{\infty}$. We consider a normalization process as a kind of search for a direct proof of a given fixed formula (type). In particular, we try to avoid changing a formula (type) during normalization. This puts some restriction on our system of reductions. As a result, the reflective λ-terms under the chosen reductions are strongly normalizable, but not confluent. A given term may have different normal forms. An additional motivation for considering those reductions is that they extend the usual set of λ-calculus reductions and subsume strong normalization for such components of $\boldsymbol{\lambda}^{\infty}$ as the intuitionistic calculus and the simply typed λ-calculus.

## 2 Reflective $\boldsymbol{\lambda}$-Calculus

The reflective λ-calculus $\boldsymbol{\lambda}^{\infty}$ below is a joint calculus of propositions (types) and proofs (λ-terms) with rigid typing. Every term and all subterms of a term carry a fixed type. In other words, in $\boldsymbol{\lambda}^{\infty}$ we assume a Church style rigid typing rather than a Curry style type assignment system.

## 2.1 Types and Typed Terms

The language of reflective $\lambda$-calculus includes

propositional letters $p_1, p_2, p_3, \ldots$

type constructors (connectives) $\rightarrow, \wedge$

term constructors (functional symbols): unary $!$ , $\Uparrow^n$, $\Downarrow^n$, $\boldsymbol{\pi}_0^n$, $\boldsymbol{\pi}_1^n$; binary $\circ^n$, $\mathbf{p}^n$, for $n = 1, 2, 3 \ldots$

operator symbols ':', $\lambda^1$, $\lambda^2$, ..., $\lambda^n$, ...;

a countably infinite supply of *variables* $x_1, x_2, x_3, \ldots$ of each type $F$ (definition below), each variable $x$ is a term of its unique pre-assigned type.

*Types* and *(well-typed, well-defined, well-formed) terms* are defined by a simultaneous induction according to the calculus $\boldsymbol{\lambda}^{\infty}$ below.

1. Propositional letters are *(atomic) types*
2. *Types* (formulas) $F$ are built according to the grammar

$$F = p \mid F \rightarrow F \mid F \wedge F \mid t{:}F,$$

where $p$ is an atomic type, $t$ a well-formed term having type $F$. Types of format $t{:}F$ where $t$ is a term and $F$ a type are called *type assertions* or *quasi-atomic types.* Note that only correct type assertions $t{:}F$ are syntactically allowed inside types. The informal semantics for $t{:}F$ is *$t$ is a proof of $F$*; so a formula

$$t_n{:}t_{n-1}{:}\ldots{:}t_1{:}A$$

can be read as "$t_n$ is a proof that $t_{n-1}$ is a proof that ... is a proof that $t_1$ proves $A$". For the sake of brevity we will refer to types as terms of depth 0.

3. *Inhabited types* and well-formed terms (or terms for short) are constructed according to the calculus $\boldsymbol{\lambda}^{\infty}$ below.

A derivation in $\boldsymbol{\lambda}^{\infty}$ is a rooted tree with nodes labelled by types, in particular, type assertions. Leaves of a derivation are labelled by axioms of $\boldsymbol{\lambda}^{\infty}$ which are arbitrary types or type assertions $x : F$ where $F$ is a type and $x$ a variable of type $F$. Note that the set of axioms is thus also defined inductively according to $\boldsymbol{\lambda}^{\infty}$: as soon as we are able to establish that $F$ is a type (in particular, for a quasi-atomic type $s{:}G$ this requires establishing by means of $\boldsymbol{\lambda}^{\infty}$ that $s$ indeed is a term of type $G$), we are entitled to use variables of type $F$ as new axioms.

A *context* is a collection of quasi-atomic types $x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$ where $x_i$, $x_j$ are distinct variables for $i \neq j$. A derivation tree is *in a context $\Gamma$* if all leaves of the derivation are labelled by some quasi-atomic types from $\Gamma$.

A step down from leaves to the root is performed by one of the inference rules of $\boldsymbol{\lambda}^{\infty}$. Each rule comes in levels $n = 0, 1, 2, 3, \ldots$. A rule has one or two premises which are types (in particular, type assertions), and a conclusion. The intended reading of such a rule is that if premises are inhabited types, then the conclusion is also inhabited. If the level of a rule is greater than 0, then the premise(s) and the conclusion are all type assertions. Such a rule is regarded also as a term formation rule with the intended reading: *the conclusion $t{:}F$ is a correct type assertion provided the premise(s) are correct.*

If $t{:}F$ appears as a label in (the root of) a derivation tree, we say that $t$ is a *term of type* $F$. We also refer to terms as well-defined, well-typed, well-formed terms.

In $\boldsymbol{\lambda}^{\infty}$ we use the natural deduction format, where derivations are represented by proof trees with assumptions, both open (charged) and closed (discharged). We will also use the sequent style notation for derivations in $\boldsymbol{\lambda}^{\infty}$ by reading $\Gamma \vdash F$ as an $\boldsymbol{\lambda}^{\infty}$-derivation of $F$ in $\Gamma$. Within the current definition below we assume that $n = 0, 1, 2, \ldots$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$. In particular, if $n = 0$ then $\boldsymbol{v}$ is empty. We also agree on the following vector-style notations:

$\boldsymbol{t}{:}A$ denotes $t_n{:}t_{n-1}{:}\ldots{:}t_1{:}A$ (e.g. $\boldsymbol{t}{:}A$ is $A$, when $n = 0$),

$\boldsymbol{t}{:}\{A_1, A_2, \ldots, A_n\}$ denotes $\{t_1{:}A_1, t_2{:}A_2, \ldots, t_n{:}A_n\}$,

$\lambda^n \boldsymbol{x}.\boldsymbol{t}{:}B$ denotes $\lambda^n x_n.t_n{:}\lambda^{n-1}x_{n-1}.t_{n-1}{:}\ldots{:}\lambda x_1.t_1{:}B$,

$(\boldsymbol{t} \circ^n \boldsymbol{s}){:}B$ denotes $(t_n \circ^n s_n){:}(t_{n-1} \circ^{n-1} s_{n-1}){:}\ldots{:}(t_1 \circ s_1){:}B$,

$\Uparrow^n \boldsymbol{t}{:}B$ denotes $\Uparrow^n t_n{:}\Uparrow^{n-1} t_{n-1}{:}\ldots{:}\Uparrow t_1{:}B$,

likewise for all other functional symbols of $\boldsymbol{\lambda}^{\infty}$.

Derivations are generated by the following clauses. Here $A, B, C$ are formulas, $\Gamma$ a finite set of types, $\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{u}$ are $n$-vectors of pseudo-terms, $\boldsymbol{x}$ are $n$-vectors of variables, $n = 0, 1, 2, \ldots$.

| | *Natural deduction rule* | *Its sequent form* |
|---|---|---|
| **(Ax)** | $\boldsymbol{x}{:}A$ | $\Gamma \vdash \boldsymbol{x}{:}A$, if $\boldsymbol{x}{:}A$ is in $\Gamma$ |

**($\boldsymbol{\lambda}$)** $$\frac{\boldsymbol{t}{:}B}{\lambda^n \boldsymbol{x}.\boldsymbol{t}{:}(A \to B)}$$

provided $x_n{:}x_{n-1}{:}\ldots{:}x_1{:}A$, $x_i$ occurs free neither in $t_j$ for $i \neq j$ nor in $A \to B$. Premises corresponding to $x_n{:}x_{n-1}{:}\ldots{:}x_1{:}A$ (if any) are discharged. In the full sequent form this rule is

$$\frac{\Gamma, x_n{:}x_{n-1}{:}\ldots{:}x_1{:}A \vdash t_n{:}t_{n-1}{:}\ldots{:}t_1{:}B}{\Gamma \vdash \lambda^n x_n.t_n{:}\lambda^{n-1}x_{n-1}.t_{n-1}{:}\ldots{:}\lambda x_1.t_1{:}(A \to B)}$$

where none of $\boldsymbol{x}$ occurs free in the conclusion sequent.

All the rules below do not bind/unbind variables.

**(App)** $$\frac{\boldsymbol{t}{:}(A \to B) \quad \boldsymbol{s}{:}A}{(\boldsymbol{t} \circ^n \boldsymbol{s}){:}B} \qquad\qquad \frac{\Gamma \vdash \boldsymbol{t}{:}(A \to B) \quad \Gamma \vdash \boldsymbol{s}{:}A}{\Gamma \vdash (\boldsymbol{t} \circ^n \boldsymbol{s}){:}B}$$

**(p)** $$\frac{\boldsymbol{t}{:}A \quad \boldsymbol{s}{:}B}{\mathbf{p}^n(\boldsymbol{t}, \boldsymbol{s}){:}(A \wedge B)} \qquad\qquad \frac{\Gamma \vdash \boldsymbol{t}{:}A \quad \Gamma \vdash \boldsymbol{s}{:}B}{\Gamma \vdash \mathbf{p}^n(\boldsymbol{t}, \boldsymbol{s}){:}(A \wedge B)}$$

$$(\boldsymbol{\pi})\quad \frac{\boldsymbol{t}\!:\!(A_0 \wedge A_1)}{\boldsymbol{\pi}_i^n \boldsymbol{t}\!:\!A_i}\ (i=0,1) \qquad\qquad \frac{\Gamma \vdash \boldsymbol{t}\!:\!(A_0 \wedge A_1)}{\Gamma \vdash \boldsymbol{\pi}_i^n \boldsymbol{t}\!:\!A_i}\ (i=0,1)$$

$$(\Uparrow)\quad \frac{\boldsymbol{t}\!:\!u\!:\!A}{\Uparrow^n \boldsymbol{t}\!:\,!u\!:\!u\!:\!A} \qquad\qquad \frac{\Gamma \vdash \boldsymbol{t}\!:\!u\!:\!A}{\Gamma \vdash \Uparrow^n \boldsymbol{t}\!:\,!u\!:\!u\!:\!A}$$

$$(\Downarrow)\quad \frac{\boldsymbol{t}\!:\!u\!:\!A}{\Downarrow^n \boldsymbol{t}\!:\!A} \qquad\qquad \frac{\Gamma \vdash \boldsymbol{t}\!:\!u\!:\!A}{\Gamma \vdash \Downarrow^n \boldsymbol{t}\!:\!A}$$

*Remark 1.* The intuitionistic logic for implication/conjunction and $\lambda$-calculus are the special cases for rules with $n = 0$ and $n = 1$ only, respectively, if we furthermore restrict all of the displayed formulas to types which do not contain quasi-atoms.

*Example 1.* Here are some examples of $\boldsymbol{\lambda}^\infty$-derivations in the sequent format (cf. 3.2). We skip the trivial axiom parts for brevity.

$$1)\quad \frac{y\!:\!x\!:\!A \vdash \Downarrow y\!:\!A}{\vdash \lambda y.\Downarrow y\!:\!(x\!:\!A \to A)} \qquad\qquad 2)\quad \frac{y\!:\!x\!:\!A \vdash \Uparrow y\!:\,!x\!:\!x\!:\!A}{\vdash \lambda y.\Uparrow y\!:\!(x\!:\!A \to\, !x\!:\!x\!:\!A)}$$

However, one can derive neither $\vdash A \to x : A$, nor $\vdash \lambda x.!x : (A \to x : A)$, since $x$ occurs free there.

$$3)\quad \cfrac{\cfrac{u\!:\!x\!:\!A, v\!:\!y\!:\!B \vdash \mathbf{p}^2(u,v)\!:\!\mathbf{p}(x,y)\!:\!(A \wedge B)}{u\!:\!x\!:\!A \vdash \lambda^2 v.\mathbf{p}^2(u,v)\!:\!\lambda y.\mathbf{p}(x,y)\!:\!(B \to (A \wedge B))}}{\vdash \lambda^2 uv.\mathbf{p}^2(u,v)\!:\!\lambda xy.\mathbf{p}(x,y)\!:\!(A \to (B \to (A \wedge B)))}$$

$$4)\quad \cfrac{\cfrac{u\!:\!x\!:\!A, v\!:\!y\!:\!B \vdash \mathbf{p}^2(u,v)\!:\!\mathbf{p}(x,y)\!:\!(A \wedge B)}{u\!:\!x\!:\!A, v\!:\!y\!:\!B \vdash \Uparrow \mathbf{p}^2(u,v)\!:\,!\mathbf{p}(x,y)\!:\!\mathbf{p}(x,y)\!:\!(A \wedge B)}}{\vdash \lambda uv.\Uparrow \mathbf{p}^2(u,v)\!:\!(x\!:\!A \to (y\!:\!B \to\, !\mathbf{p}(x,y)\!:\!\mathbf{p}(x,y)\!:\!(A \wedge B)))}$$

Note that unlike in the previous example we cannot introduce $\lambda^2$ in place of $\lambda$ at the last stage here since the resulting sequent would be

$$\vdash \lambda^2 uv.\Uparrow \mathbf{p}^2(u,v)\!:\!\lambda xy.!\mathbf{p}(x,y)\!:\!(A \to (B \to \mathbf{p}(x,y)\!:\!(A \wedge B)))$$

containing abstraction over variables $x, y$ which are left free in the conclusion, which is illegal.

Here is an informal explanation of why such a derivation should not be permitted. Substituting different terms for $x$ and $y$ in the last sequent produces different types from $A \to (B \to \mathbf{p}(x,y) : (A \wedge B))$, whereas neither of the terms $\lambda xy.!\mathbf{p}(x,y)$ and $\lambda^2 uv.\Uparrow \mathbf{p}^2(u,v)$ changes after such substitutions. This is bad syntactically, since the same terms will be assigned different types. Semantically this is bad either, since this would violate the one proof - one theorem convention.

**Proposition 1.** *(Closure under substitution) If $t(x)$ is a well-defined term, $x$ a variable of type $A$, $s$ a term of type $A$ free for $x$ in $t(x)$, then $t(s)$ is a well-defined term of the same type as $t(x)$.*

**Proposition 2.** *(Uniqueness of Types) If both $t : F$ and $t : F'$ are well-typed terms, then $F \equiv F'$.*

**Theorem 1.** *(Internalization Property for $\boldsymbol{\lambda}^\infty$) Let $\boldsymbol{\lambda}^\infty$ derive*

$$A_1, A_2, \dots, A_m \vdash B.$$

*Then one can build a well-defined term $t(x_1, x_2, \dots, x_m)$ with fresh variables $\boldsymbol{x}$ such that $\boldsymbol{\lambda}^\infty$ also derives*

$$x_1 {:} A_1, x_2 {:} A_2, \dots, x_m {:} A_m \vdash t(x_1, x_2, \dots, x_m) {:} B.$$

*Proof.* We increment $n$ at every node of the derivation $A_1, A_2, \dots, A_m \vdash B$. The base case is obvious. We will check the most principal step clause **($\boldsymbol{\lambda}$)** leaving the rest as an exercise. Let the last step of a derivation be

$$\frac{\Gamma, y_n {:} y_{n-1} {:} \dots {:} y_1 {:} A \vdash t_n {:} t_{n-1} {:} \dots {:} t_1 {:} B}{\Gamma \vdash \lambda^n y_n.t_n {:} \lambda^{n-1} y_{n-1}.t_{n-1} {:} \dots {:} \lambda y_1.t_1 {:} (A \to B)} \,.$$

By the induction hypothesis, for some term $s(\boldsymbol{x}, x_{m+1})$ of fresh variables $\boldsymbol{x}, x_{m+1}$

$$\boldsymbol{x} {:} \Gamma, x_{m+1} {:} y_n {:} y_{n-1} {:} \dots {:} y_1 {:} A \vdash s(\boldsymbol{x}, x_{m+1}) {:} t_n {:} t_{n-1} {:} \dots {:} t_1 {:} B.$$

Apply the rule **($\boldsymbol{\lambda}$)** for $n+1$ to obtain

$$\boldsymbol{x} {:} \Gamma \vdash \lambda^{n+1} x_{m+1}.s {:} \lambda^n y_n.t_n {:} \lambda^{n-1} y_{n-1}.t_{n-1} {:} \dots {:} \lambda y_1.t_1 {:} (A \to B),$$

and put $t(x_1, x_2, \dots, x_m) = \lambda^{n+1} x_{m+1}.s(\boldsymbol{x}, x_{m+1})$.

## 2.2 Reductions in the $\boldsymbol{\lambda}^\infty$Calculus

**Definition 1.** *For $n = 1, 2, \dots$, the redexes for $\boldsymbol{\lambda}^\infty$ and their contracta (written as $t \rhd t'$, for $t$ a redex, $t'$ a contractum of $t$) are:*

- $(\lambda^n x_n.t_n) \circ^n s_n : \dots : (\lambda x_1.t_1) \circ s_1 : F \;\rhd\; t_n[s_n/x_n] : \dots : t_1[s_1/x_1] : F$ *($\beta_n$-contraction)*
- $\boldsymbol{\pi}_i^n \mathbf{p}^n(t_0^n, t_1^n) {:} \boldsymbol{\pi}_i^{n-1} \mathbf{p}^{n-1}(t_0^{n-1}, t_1^{n-1}) {:} \dots {:} \boldsymbol{\pi}_i \mathbf{p}(t_0^1, t_1^1) {:} F \;\rhd$ $t_i^n : t_i^{n-1} : \dots : t_i^1 : F,\ i = 0, 1$
- $\Downarrow^n \Uparrow^n t_n : \Downarrow^{n-1} \Uparrow^{n-1} t_{n-1} : \dots : \Downarrow \Uparrow t_1 : F \rhd t_n : t_{n-1} : \dots : t_1 : F$

Before defining the reduction relation on reflective λ-terms (Definition 5) we will come with some motivations (Remarks 2 and 3).

*Remark 2.* The system should allow types of normalized terms to contain terms which are not normal. The object, then, is to normalize a term of any type, regardless of the redundancy of the type, and focus on eliminating redundancy in the term. It has been noted, for instance, that the statement of the normalization property for a term system, contains "redundancy", because it says that something which is not normal, is equivalent to a normal thing.

*Remark 3.* In the simply-typed and untyped $\lambda$-calculi, reduction of terms is defined as contraction of the redexes which occur as subterm occurrences of the term. For $\boldsymbol{\lambda}^\infty$, the situation is a little trickier. We want to be sure that, when reducing a term $t_n$ of type $t_{n-1}:\ldots:t_1:A$, we end up with a well-typed term $t'_n$ having type $t'_{n-1}:\ldots:t'_1:A$. I.e., we don't want the formula $A$ to change under reduction, while the intermediary terms $t_j$, may be allowed to change in a predictable way.

An example, illustrating the difficulties arising when we allow any subterm occurrence of a redex to contract, is the $\boldsymbol{\lambda}^\infty$-term

$$\dfrac{\dfrac{x_3:x_2:x_1:A \vdash \Uparrow^2 x_3:\Uparrow x_2:!x_1:x_1:A}{x_3:x_2:x_1:A \vdash \Downarrow^2\Uparrow^2 x_3:\Downarrow\Uparrow x_2:x_1:A}}{\vdash \lambda x_3.\Downarrow^2\Uparrow^2 x_3:(x_2:x_1:A \to \Downarrow\Uparrow x_2:x_1:A)}$$

Let us use a blend of type assertion notations $t^F$ and $t:F$ to show the type of the subterm $\Downarrow^2\Uparrow^2 x_3$ of the resulting term more explicitly.

$$[\lambda x_3.(\Downarrow^2\Uparrow^2 x_3)^{\Downarrow\Uparrow x_2:x_1:A}]^{(x_2:x_1:A\to\Downarrow\Uparrow x_2:x_1:A)}$$

In principle, the subterm $(\Downarrow^2\Uparrow^2 x_3)^{\Downarrow\Uparrow x_2:x_1:A}$ is a redex. However, from the proof theoretical point of view it would be a mistake to allow the whole term to reduce to

$$[\lambda x_3.x_3]^{(x_2:x_1:A\to x_2:x_1:A)}.$$

Indeed, the formula (type) $\tau$

$$x_2:x_1:A\to\Downarrow\Uparrow x_2:x_1:A$$

is a nontrivial formalized special case of so-called *subject extension.* We do not want to change this formula (type) in a normalization process to

$$x_2:x_1:A\to x_2:x_1:A$$

which is a trivial proposition. Speaking proof theoretically, while normalizing we are looking for a normal proof of a given proposition (here it is formula $\tau$) rather than changing a proposition itself in a process of finding a "better" proof of it. Abandoning $\tau$ in the normalization process would not allow us to find a natural normal inhabitant (normal proof) of $\tau$. The principal difficulty here is that the main term is "lower level" than the subterm occurrence being contracted. Accordingly, we develop a notion of subterm occurrences sitting properly in a term.

**Definition 2.** *For $t$ a well-defined $\boldsymbol{\lambda}^\infty$-term, $t$ is of level $n$ (written, $\ell ev(t)=n$) if the bottom rule in the derivation tree has superscript $n$.*

**Definition 3.** *For $t_0$ a subterm occurrence of a $\boldsymbol{\lambda}^\infty$-term $t$, with $\ell ev(t_0)=n$, we say that $t_0$ is properly embedded in $t$ ($t_0 \sqsubseteq_p t$) if there are no subterm occurrences in $t$ containing $t_0$ of the forms: $\lambda^m x.s$, $s\circ^m r$, $\mathbf{p}^m(s,r)$, $\boldsymbol{\pi}_i^m s$, $\Downarrow^m s$, with $m<n$*

Speaking vaguely, we require that redexes be properly embedded in all subterms except, may be, the "proof checkers" $\Uparrow^m s$. We will show below that this requirement suffices for type preservation, and strong normalization of $\boldsymbol{\lambda}^\infty$-terms.

We may now define reduction of $\boldsymbol{\lambda}^\infty$-terms.

**Definition 4.** **(a)** *For any well-defined $\boldsymbol{\lambda}^\infty$-terms $t$ and $t'$, $t$ one-step reduces to $t'$ ($t \succ_1 t'$) if $t$ contains a properly embedded subterm occurrence $s$ of the form of one of the redexes above, $s \rhd s'$, and $t' \equiv t[s'/s]$.*

**(b)** *The reduction relation $\succeq$ is the reflexive and transitive closure of $\succ_1$.*

Equipped with the above-defined notion of reduction, $\boldsymbol{\lambda}^\infty$ enjoys (suitably modified versions of) the most important properties of the simply typed lambda calculus. The elementary property of type preservation under reduction holds, in a modified form, namely, for $t : F$ a well-typed term, if $t \succ_1 t'$ then $t' : F'$ is well-typed, for a naturally determined type $F'$. More precisely,

**Proposition 3.** *(Type Preservation) Let $t_n$ be a well-typed term of level $n$, having type $t_{n-1} : \ldots : t_1 : A$. If $t_n \succ_1 t'_n$, then $t'_n$ is a well-typed level $n$ term. Furthermore, $t'_n$ has type $t'_{n-1} : \ldots : t'_1 : A$, with $t_i \succ_1 t'_i$ or $t_i \equiv t'_i$, for $1 \le i \le n-1$.*

*Proof.* $t_n \succ_1 t'_n$, by definition, only if there exists $s_n \sqsubseteq_p t_n, s_n \rhd s'_n$, and $t'_n \equiv t_n[s'_n/s_n]$. We proceed by induction on $|t_n|$, and on the number of subterm occurrences between $t_n$ and $s_n$, and construct a derivation in $\boldsymbol{\lambda}^\infty$ of the new term $t'_n$. The construction of this derivation is analogous to the construction of derivations in intuitionistic logic under detour reduction.

We get a form of subject reduction as an immediate consequence:

**Corollary 1.** *(Subject Reduction) $\Gamma \vdash \boldsymbol{t} : A \Rightarrow \Gamma \vdash \boldsymbol{t'} : A$, for $t_n \succ_1 t'_n$.*

*Remark 4.* It is easy to see that the derivation of $t'_n$ obtained in the proof above, is uniquely determined by $t'_n$, and the derivation of $t_n$. In this way, we associate to each reduction of terms, a corresponding reduction of derivations. Under this operation of simultaneous term and derivation reduction, we see how the internalization property of $\boldsymbol{\lambda}^\infty$ is a generalization of the Curry-Howard isomorphism. Namely, the square in Figure 1 commutes, where $\iota$ is the map given by the Internalization Property, and $\succ_1$ is the map given by one-step reduction.

*Example 2.* We may modify the term from the above example, to obtain a term with a properly embedded redex:

$$\cfrac{\cfrac{x_3 : x_2 : x_1 : A \vdash \Uparrow^2 x_3 : \Uparrow x_2 : !x_1 : x_1 : A}{x_3 : x_2 : x_1 : A \vdash \Downarrow^2 \Uparrow^2 x_3 : \Downarrow \Uparrow x_2 : x_1 : A}}{\vdash \lambda^2 x_3 . \Downarrow^2 \Uparrow^2 x_3 : \lambda x_2 . \Downarrow \Uparrow x_2 : (x_1 : A \to x_1 : A)}$$

$$\begin{tikzcd}
\begin{array}{c}\vdots\\ \mathcal{D}\\ \boldsymbol{t_n} : A\end{array} \arrow[r, "\iota"] \arrow[d, "\succ_1"] & \begin{array}{c}\vdots\\ \mathcal{D}_\iota\\ \iota(\boldsymbol{t_n}) : \boldsymbol{t_n} : A\end{array} \arrow[d, "\succ_1"] \\
\begin{array}{c}\vdots\\ \mathcal{D}'\\ \boldsymbol{t'_n} : A\end{array} \arrow[r, "\iota"] & \begin{array}{c}\vdots\\ \mathcal{D}'_\iota\\ \iota(\boldsymbol{t'_n}) : \boldsymbol{t'_n} : A\end{array}
\end{tikzcd}$$

**Fig. 1.** *Generalization of the Curry-Howard isomorphism*

The (properly embedded) redex $\Downarrow^2\Uparrow^2 x_3$ contracts to $x_3$, and the term derivation and type are modified accordingly:

$$\frac{x_3 : x_2 : x_1 : A \vdash x_3 : x_2 : x_1 : A}{\vdash \lambda^2 x_3.x_3 : \lambda x_2.x_2 : (x_1 : A \rightarrow x_1 : A)}$$

The result really is a normal proof of the statement inside the parentheses.

**Proposition 4.** *The reduction relation $\succeq$ is not confluent.*

*Proof.* By a counterexample. Consider the following well-defined terms (missing types can be easily recovered).

$\mathbf{p}^2(x_2, y_2) : \mathbf{p}(x_1, y_1) : (A \wedge B)$
$\lambda x_2.\mathbf{p}^2(x_2, y_2) : (x_1 : A \rightarrow \mathbf{p}(x_1, y_1) : (A \wedge B))$
$\lambda x_2.\mathbf{p}^2(x_2, y_2)) \circ z : \mathbf{p}(x_1, y_1) : (A \wedge B)$
$\lambda^2 y_2.[(\lambda x_2.\mathbf{p}^2(x_2, y_2)) \circ z] : \lambda y_1.\mathbf{p}(x_1, y_1) : (B \rightarrow (A \wedge B))$

If we call the above term $t_2 : t_1 : (B \rightarrow (A \wedge B))$,
then $t_2 \circ^2 \boldsymbol{\pi}_1^2\mathbf{p}^2(u_2, w_2) : t_1 \circ \boldsymbol{\pi}_1\mathbf{p}(u_1, w_1) : (A \wedge B)$ reduces to both:
(1) $t_2[\boldsymbol{\pi}_1^2\mathbf{p}^2(u_2, w_2)/y_2] : t_1[\boldsymbol{\pi}_1\mathbf{p}(u_1, w_1)/y_1] : (A \wedge B)$ and
(2) $t_2 \circ^2 u_2 : t_1 \circ u_1 : (A \wedge B)$,

which further reduces to
(3) $t_2[u_2/y_2] : t_1[u_1/y_1] : (A \wedge B)$.

And, there's no way to reduce (1) to (3), since $\boldsymbol{\pi}_1^2\mathbf{p}^2(u_2, w_2)$ isn't properly embedded in $t_2[\boldsymbol{\pi}_1^2\mathbf{p}^2(u_2, w_2)/y_2]$.

## 3 Strong Normalization of $\boldsymbol{\lambda}^\infty$-Terms

**Definition 5.** **(a)** *A reduction sequence is any sequence of terms* $t_0, \ldots, t_n \ldots$ *such that, for* $0 \le j < n$, $t_j \succ_1 t_{j+1}$. **(b)** *A term* $t$ *is strongly normalizing* ($t \in \mathbf{SN}$) *if every possible reduction sequence for* $t$ *is finite.* **(c)** *If* $t$ *is strongly normalizing,* $\nu(t) =$ *the least* $n$ *such that every reduction sequence for* $t$ *terminates in fewer than* $n$ *steps.*

We now show that all terms of $\boldsymbol{\lambda}^\infty$ are strongly normalizing. The proof follows the method of Tait (cf. [7]). This method defines a class of *reducible* terms by induction on type. We are concerned with types, but will ignore variations in type, *up to a point.*

**Definition 6.** *More precisely, a term* $t$ *is* $n$*-cushioned of type* $A$ *if* $t$ *has type* $t_{n-1} : \ldots : t_1 : A$, *where the* $t_i$ *are* $\boldsymbol{\lambda}^\infty$*-terms, for* $1 \le i \le n-1$. *Note that this is a weaker requirement than* $\ell ev(t) = n$. *Indeed,* $\ell ev(t) = n$ *implies* $t$ *is* $k$*-cushioned, for* $k \le n$, *and it could also be that* $t$ *is* $m$*-cushioned, for some* $m > n$.

For the proof, we'll define sets $\mathtt{RED}^n_T$ of reducible terms for $n$-cushioned terms of type $T$, by induction on $T$. We then prove some properties of these sets of terms, including the property that all terms in the sets are strongly normalizing. Finally, strong normalization of the calculus is established by showing that all $\boldsymbol{\lambda}^\infty$-terms are in a set $\mathtt{RED}^n_T$.

**Definition 7.** *If* $T$ *is a type, we define the depth of* $T$, $|T|$, *inductively:*

$|p| = 0$, *for* $p$ *an atomic proposition*
$|A \wedge B| = |A \to B| = max(|A|, |B|) + 1$
$|u : A| = |A| + 1$

**Definition 8.** *The sets* $\mathtt{RED}^n_T$ *of reducible* $n$*-cushioned terms of type* $T$ *are defined by induction on* $|T|$:

1. *For* $t$ *of atomic type* $p$, $t \in \mathtt{RED}^n_p$ *if* $t$ *is strongly normalizing.*
2. *For* $t$ *of type* $A \to B$, $t \in \mathtt{RED}^n_{A\to B}$ *if, for all* $s \in \mathtt{RED}^n_A$, $t \circ^n s \in \mathtt{RED}^n_B$.
3. *For* $t$ *of type* $A_0 \wedge A_1$, $t \in \mathtt{RED}^n_{A_0 \wedge A_1}$ *if* $\boldsymbol{\pi}^n_i t \in \mathtt{RED}^n_{A_i}$, *for* $i = 0, 1$.
4. *For* $t$ *of type* $u : A$, $t \in \mathtt{RED}^n_{u:A}$ *if* $\Downarrow^n t \in \mathtt{RED}^n_A$ *and* $t \in \mathtt{RED}^{n+1}_A$

**Proposition 5.** *For* $t$ *a well-typed term, if* $t$ *is strongly normalizing, then* $\Downarrow^n t$ *is too.*

*Proof.* If $t$ is not of the form $\Uparrow^n t_0$, then every reduction sequence for $\Downarrow^n t$ has less or equal to $\nu(t)$ terms in it. If $t$ is of the form $\Uparrow^n t_0$, then every reduction sequence for $\Downarrow^n t$ has length $\le \nu(t) + 1$.

**Definition 9.** *A term* $t$ *is neutral if* $t$ *is not of the form* $\lambda^n x.t_0$, $\mathbf{p}^n(t_0, t_1)$, *or* $\Uparrow^n t_0$.

**Proposition 6.** *For $T$ any type, all terms $t \in \mathtt{RED}^n_T$ have the following properties:*

**(CR 1)** *If $t \in \mathtt{RED}^n_T$, then $t$ is strongly normalizing.*
**(CR 2)** *If $t \in \mathtt{RED}^n_T$ and $t \succeq t'$, then $t' \in \mathtt{RED}^n_{T'}$, for*
$t : u_{n-1} : u_{n-2} : \ldots : u_1 : T \succeq t' : u'_{n-1} : u'_{n-2} : \ldots : u'_1 : T'$
**(CR 3)** *If $t$ is neutral, and $t' \in \mathtt{RED}^n_T$ for all $t'$ such that $t \succ_1 t'$, then $t \in \mathtt{RED}^n_T$*
**(CR 4)** *If $t \in \mathtt{RED}^n_T$, then $t \in \mathtt{RED}^{n-1}_{u_1:T}$, for $t : u_{n-1} : u_{n-2} : \ldots : u_1 : T$.*

*Proof.* Proof by induction on $|T|$.

**Step 1** $T$ is atomic, i.e. $T \equiv p$.
**(CR 1)** Trivial.
**(CR 2)** If $t$ is **SN**, and $t \succeq t'$, then $t'$ is **SN**, so $t'$ is reducible.
**(CR 3)** Let $M = max(\nu(t') \mid t \succ_1 t')$. There are finitely many $t'$ such that $t \succ_1 t'$, so $M$ is finite. And, $\nu(t) \leq M + 1$.
**(CR 4)** $t \in \mathtt{RED}^n_p$ yields $t$ is **SN** thus $\Downarrow^{n-1} t$ is **SN**, by above proposition. Therefore, $\Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_p$ and $t \in \mathtt{RED}^{n-1}_{u_1:p}$
**Step 2** $T \equiv (A \to B)$.
**(CR 1)** Let $x$ be a variable of type $A$. Then $t \circ^n x \in \mathtt{RED}^n_B$, and by induction, $t \circ^n x$ is **SN**. Since every reduction sequence for $t$ is embedded in a reduction sequence of $t \circ^n x$, $t$ is **SN**. This argument works since $t \succeq t'$ implies that the subterm occurrence being contracted is properly embedded in $t$, which means this subterm occurrence must have $lev < n + 1$. Under those conditions $t \succeq t'$ implies $t \circ^n x \succeq t' \circ^n x$.
**(CR 2)** Let $t \succeq t'$ for $t : u_{n-1} : u_{n-2} : \ldots : u_1 : T$, and take $s \in \mathtt{RED}^n_A$ for $s : w_{n-1} : w_{n-2} : \ldots : w_1 : A$. Then $t \circ^n s \in \mathtt{RED}^n_B$, and $t \circ^n s \succeq t' \circ^n s$. By induction, $t' \circ^n s$ is reducible. Since $s$ was arbitrary, $t'$ is reducible.
**(CR 3)** Let $t$ be neutral, and suppose that $t'$ is reducible, for all $t'$ such that $t \succ_1 t'$. Let $s \in \mathtt{RED}^n_A$. By induction, $s$ is **SN**. We prove, by induction on $\nu(s)$ that if $t \circ^n s \succ_1 (t \circ^n s)'$, $(t \circ^n s)'$ is reducible. There are two possibilities for $(t \circ^n s)'$:
1. $(t \circ^n s)'$ is $t' \circ^n s$, with $t \succ_1 t'$. Then $t'$ is reducible, so $t' \circ^n s$ is.
2. $(t \circ^n s)'$ is $t \circ^n s'$, with $s \succ_1 s'$. Then $s'$ is reducible, so $t \circ^n s'$ is.

By the induction hypothesis, $t \circ^n s$ is reducible, because it's neutral. Thus, $t$ is reducible.
**(CR 4)** We need to show that $\Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_{A \to B}$. Let $r \in \mathtt{RED}^{n-1}_A$. By induction, $r$ is **SN**; and by **(CR 1)**, $t$ is **SN**. Thus, by the above proposition, $\Downarrow^{n-1} t$ is **SN**, and we can induct on $\nu(r) + \nu(\Downarrow^{n-1} t)$. Base: $\nu(r) + \nu(\Downarrow^{n-1} t) = 0$ . Then $\Downarrow^{n-1} t \circ^{n-1} r$ is normal (and it is neutral), so by induction, $\Downarrow^{n-1} t \circ^{n-1} r \in \mathtt{RED}^{n-1}_B$. Induction: in one step, $\Downarrow^{n-1} t \circ^{n-1} r$ reduces to
• $(\Downarrow^{n-1} t)' \circ^{n-1} r$ - reducible by induction
• $\Downarrow^{n-1} t \circ^{n-1} r'$ - reducible by induction.
Hence, by induction hypothesis **(CR 3)**, $\Downarrow^{n-1} t \circ^{n-1} r \in \mathtt{RED}^{n-1}_B$ for all $r \in \mathtt{RED}^{n-1}_A$. Thus, $\Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_{A \to B}$, and $t \in \mathtt{RED}^{n-1}_{u_1:(A \to B)}$.
**Step 3** $T \equiv (A \wedge B)$.

**(CR 1)**–**(CR 3)** Similar to the previous step. Note that $t \succeq t'$ yields $\boldsymbol{\pi}_i^n t \succeq \boldsymbol{\pi}_i^n t'$, for $t : u_{n-1} : u_{n-2} : \ldots : u_1 : (A_0 \wedge A_1)$.

**(CR 4)** By **(CR 1)**, $t$ is **SN**. We need to show that $\Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_{A_0 \wedge A_1}$, i.e. that $\boldsymbol{\pi}_i^{n-1}(\Downarrow^{n-1} t) \in \mathtt{RED}^{n-1}_{A_i}$, for $i = 0, 1$. By induction on $\nu(\Downarrow^{n-1} t)$. Base $\nu(\Downarrow^{n-1} t) = 0$. Then $\boldsymbol{\pi}_i^{n-1}(\Downarrow^{n-1} t)$ is both neutral and normal. By I.H. **(CR 3)**, $\boldsymbol{\pi}_i^{n-1}(\Downarrow^{n-1} t) \in \mathtt{RED}^{n-1}_{A_i}$. Induction: in one step $\boldsymbol{\pi}_i^{n-1}(\Downarrow^{n-1} t)$ reduces to $\boldsymbol{\pi}_i^{n-1}(\Downarrow^{n-1} t')$ which is reducible, by I.H..

**Step 4** $T \equiv (u : A)$.

**(CR 1)** $t \in \mathtt{RED}^n_{u:A}$ yields $t \in \mathtt{RED}^{n+1}_A$, thus. by I.H., $t$ is **SN**.

**(CR 2)** Let $t \succeq t'$. By I.H. **(CR 2)**, $t' \in \mathtt{RED}^{n+1}_{A'}$. By I.H. **(CR 4)**, $t' \in \mathtt{RED}^n_{u':A'}$.

**(CR 3)** Suppose $t' \in \mathtt{RED}^n_{u':A'}$ for all $t'$ such that $t \succ_1 t'$. Then $t' \in \mathtt{RED}^{n+1}_{A'}$ for all $t'$ such that $t \succ_1 t'$. By I.H. **(CR 3)**, $t \in \mathtt{RED}^{n+1}_A$. By I.H. **(CR 4)**, $t \in \mathtt{RED}^n_{u:A}$.

**(CR 4)** Let $t \in \mathtt{RED}^n_{u:A}$. By **(CR 1)**, $t$ is **SN**. Thus, by Proposition 5, $\Downarrow^{n-1} t$ is **SN**. We need to show that $\Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_{u:A}$, i.e. that $\Downarrow^{n-1} t \in \mathtt{RED}^n_A$ and $\Downarrow^{n-1}\Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_A$. Both are easily shown by induction on $\nu(\Downarrow^{n-1} t)$ and on $\nu(\Downarrow^{n-1}\Downarrow^{n-1} t)$, respectively, by using **(CR 3)** inductively.

**Proposition 7.** *1. If, for all $u \in \mathtt{RED}^n_A$, $t[u/x] \in \mathtt{RED}^n_B$, then $\lambda^n x.t \in \mathtt{RED}^n_{A \to B}$.*
*2. If $t_0$, $t_1$ are reducible, then $\boldsymbol{p}^n(t_0, t_1)$ is reducible.*
*3. If $t \in \mathtt{RED}^n_{u:A}$, then $\Uparrow^n t \in \mathtt{RED}^n_{!u:u:A}$.*
*4. If $t \in \mathtt{RED}^n_A$ and $m < n$, then $\Uparrow^m t \in \mathtt{RED}^{n+1}_A$.*

*Proof.* For part (1): Let $u \in \mathtt{RED}^n_A$. We'll show that $(\lambda^n x.t) \circ^n u$ is reducible, by induction on $\nu(u) + \nu(t)$. $(\lambda^n x.t) \circ^n u \succ_1$:

- $t[u/x]$, which is reducible by hypothesis.
- $(\lambda^n x.t') \circ^n u$ with $t \succ_1 t'$. By induction, this is reducible.
- $(\lambda^n x.t) \circ^n u'$ with $u \succ_1 u'$. By induction, this is reducible.

Parts (2) and (3) are established similarly. Part 4 is by induction on $|A|$. The case $A$ is atomic is handled in the obvious way, implications and conjunctions are handled by induction on $\nu(t) + \nu(r)$ and $\nu(t)$ respectively. Finally, the case where $A$ is $u : B$ follows from the induction hypothesis.

**Proposition 8.** *Let $t$ be a term. Suppose the free variables of $t$ are among $x_1, \ldots, x_k$, of levels $n_1, \ldots, n_k$ and having types $U_1, \ldots, U_k$. Let $u_1, \ldots, u_k$ be terms with $u_1 \in \mathtt{RED}^{n_1}_{U_1}, \ldots, u_k \in \mathtt{RED}^{n_k}_{U_k}$. Then the term $t[u_1/x_1, \ldots, u_k/x_k]$ (written $t[\boldsymbol{u}/\boldsymbol{x}]$), is reducible.*

*Proof.* **Step 1** $t$ entered by **(Ax)**. Trivial.

**Step 2** Let $t$ be $\lambda^n y_n.t_n^0 : \lambda^{n-1} y_{n-1}.t_{n-1}^0 : \ldots : \lambda y_1.t_1^0 : (A \to B)$. By I.H., $t_n^0[v_n/y_n, \boldsymbol{u}/\boldsymbol{x}]$ is reducible (i.e., is in some reducibility class) for all $v_n \in \mathtt{RED}^m_T$, where $m$ is the level of $y_n$, and $T$ is $y_n$'s type (so $y_n$ has type $r_{m-1} : \ldots : r_1 : T$, and $m \geq n$). By **(CR 4)**, and the definition of the classes $\mathtt{RED}^k_{w:F}$, this implies that $t_n^0[v_n/y_n, \boldsymbol{u}/\boldsymbol{x}]$ is reducible for all $v_n \in \mathtt{RED}^n_A$. Now, using **(CR 4)** and the definition again, $t_n^0[v_n/y_n, \boldsymbol{u}/\boldsymbol{x}]$ is reducible implies that $t_n^0[v_n/y_n, \boldsymbol{u}/\boldsymbol{x}] \in \mathtt{RED}^n_B$. By Proposition 7, $\lambda^n y_n.t_n^0[\boldsymbol{u}/\boldsymbol{x}] (\equiv t[\boldsymbol{u}/\boldsymbol{x}])$ is reducible.

**Step 3** The other six cases are handled similarly, using either the definition of the reducibility classes or Proposition 7.

As a corollary, we have:

**Theorem 2.** *All well-defined terms of* $\boldsymbol{\lambda}^\infty$ *are strongly normalizable.*

## 4 Conclusion

1. Proof polynomials from the Logic of Proofs ([3]) represent the reflective idea in the combinatory terms format, with many postulated constant terms and three basic operations only. Such an approach turned out to be fruitful in proof theory, modal and epistemic logics, logics of knowledge, etc. Proof polynomials are capable of extracting explicit witnesses from any modal derivation and thus may be regarded as a logical basis for the quantitative theory of knowledge. On the other hand, such areas as typed programming languages, theorem provers, verification systems, etc., use the language of $\lambda$-terms rather than combinatory terms. Technically speaking, in the $\lambda$ format there are no postulated constants, but there are many (in our case infinitely many) operations on terms. Those operations create certain redundancies in terms, and the theory of eliminating those redundancies (i.e. the normalization process) plays an important role. Reflective $\lambda$-calculus $\boldsymbol{\lambda}^\infty$ is a universal superstructure over formal languages based on types and $\lambda$-terms. It offers a much richer collection of types, in particular the ones capable of encoding computations in types. We wish to think that a wide range of systems based on types and terms could make use of such a capability.

2. The very idea of reflective terms is native to both provability and computability. The extension of the Curry-Howard isomorphism offered by the reflective $\lambda$-calculus $\boldsymbol{\lambda}^\infty$ captures reflexivity in a uniform abstract way. This opens a possibility of finding more mutual connections between proofs and programs. In this paper we have built strongly normalizing reflective $\lambda$-terms that correspond to some proper subclass of proof polynomials. It is pivotal to study exact relations between $\boldsymbol{\lambda}^\infty$, Intuitionistic Modal Logic and the Intuitionistic Logic of Proofs. Such an investigation could bring important new rules to $\boldsymbol{\lambda}^\infty$, and eventually to programming languages.

3. Reflective $\lambda$-calculus is capable of internalizing its own reductions as well as its own derivations. If a term $t'{:}F'$ is obtained from $t{:}F$ by a reduction, then the corresponding derivation (well-formed term) $s:t:F$ can be transformed to a term $s':t':F'$. This takes care of the cases when, for example, an abstraction term is obtained not by the abstraction rule, but by a projection reduction. Such a step can be internalized in $\boldsymbol{\lambda}^\infty$ as well.

4. A more concise equivalent axiomatization of $\boldsymbol{\lambda}^\infty$ could probably be obtained by formalizing Internalization as an atomic rule of inference rather than a derived rule. This route is worth pursuing.

5. There are many natural systems of reductions for reflective $\lambda$-terms. For this paper we have picked one with well-embedded redexes on the basis of the

proof theoretical semantics. We conjecture that the system with unlimited reductions enjoys both strong normalization and confluence.

**Acknowledgment.** Sergei Artemov suggested the system $\boldsymbol{\lambda}^\infty$ and wrote sections 1, 2.1, 4. Jesse Alt made a major contribution to finding the proof of the strong normalization theorem $\boldsymbol{\lambda}^\infty$ and wrote most of sections 2.2, 3.

Special thanks are due to Robert Constable and Anil Nerode for supporting this research. The authors are also indebted to Volodya Krupski, Roma Kuznets, Alesha Nogin, Lena Nogina, and the anonymous referee for a reading different versions of this paper which led to valuable improvements.

## References

1. S. Artemov, "Operational Modal Logic," *Tech. Rep. MSI 95-29*, Cornell University, December 1995 `http://www.cs.cornell.edu/Info/People/artemov/MSI95-29.ps`
2. S. Artemov, "Uniform provability realization of intuitionistic logic, modality and lambda-terms", Electronic Notes on Theoretical Computer Science, vol. 23, No. 1. 1999 `http://www.elsevier.nl/entcs`
3. S. Artemov, "Explicit provability and constructive semantics", The *Bulletin for Symbolic Logic*, v.7, No. 1, pp. 1-36, 2001 `http://www.cs.cornell.edu/Info/People/artemov/BSL.ps`.
4. H. Barendregt, "Lambda calculi with types". In *Handbook of Logic in Computer Science, vol.2*, Abramsky, Gabbay, and Maibaum, eds., Oxford University Press, pp. 118-309, 1992
5. R. Constable, "Types in logic, mathematics and programming". In *Handbook in Proof Theory*, S. Buss, ed., Elsevier, pp. 683-786, 1998
6. D. de Jongh and G. Japaridze, "Logic of Provability", in S. Buss, ed., *Handbook of Proof Theory*, Elsevier, pp. 475-546, 1998
7. J.-Y. Girard, Y. Lafont, P. Taylor, *Proofs and Types*, Cambridge University Press, 1989.
8. V.N. Krupski, "Operational Logic of Proofs with Functionality Condition on Proof Predicate", Lecture Notes in Computer Science, v. 1234, *Logical Foundations of Computer Science' 97, Yaroslavl'*, pp. 167-177, 1997
9. V.N. Krupski, "The single-conclusion proof logic and inference rules specification", Annals of Pure and Applied Logic, v.110, No. 1-3, 2001 (to appear).
10. A. Nerode, "Applied Logic". In *The merging of Disciplines: New Directions in Pure, Applied, and Computational Mathematics*, R.E. Ewing, K.I. Gross, C.F. Martin, eds., Springer-Verlag, pp. 127-164, 1986
11. A. Troelstra, "Realizability". In *Handbook in Proof Theory*, S. Buss, ed., Elsevier, pp. 407-474, 1998
12. A.S. Troelstra and H. Schwichtenberg, *Basic Proof Theory*, Cambridge University Press, 1996.