Partiality and Recursion in Higher-Order Logic

Łukasz Czajka

Institute of Informatics, University of Warsaw
Banacha 2, 02-097 Warszawa, Poland
lukaszcz@mimuw.edu.pl

Abstract. We present an illative system $\mathcal{I}_s$ of classical higher-order logic with subtyping and basic inductive types. The system $\mathcal{I}_s$ allows for direct definitions of partial and general recursive functions, and provides means for handling functions whose termination has not been proven. We give examples of how properties of some recursive functions may be established in our system. In a technical appendix to the paper we prove consistency of $\mathcal{I}_s$. The proof is by model construction. We then use this construction to show conservativity of $\mathcal{I}_s$ over classical first-order logic. Conservativity over higher-order logic is conjectured, but not proven.

1 Introduction

We present an illative λ-calculus system $\mathcal{I}_s$ of classical higher-order logic with subtyping and basic inductive types. Being illative means that the system is a combination of higher-order logic with the *untyped* λ-calculus. It therefore allows for unrestricted recursive definitions directly, including definitions of possibly non-terminating partial functions. We believe that this feature of $\mathcal{I}_s$ makes it potentially interesting as a logic for an interactive theorem prover intended to be used for program verification.

In order to ensure consistency, most popular proof assistants allow only total functions, and totality must be ensured by the user, either by very precise specifications of function domains, restricting recursion in a way that guarantees termination, explicit well-foundedness proofs, or other means. There are various indirect ways of dealing with general recursion in popular theorem provers based on total logics. There are also many non-standard logics allowing partial functions directly. We briefly survey some related work in Sect. 5.

In Sect. 2 we introduce the system $\mathcal{I}_s$. Our approach builds on the old tradition of illative combinatory logic [1,2,3]. This tradition dates back to early inconsistent systems of Shönfinkel, Church and Curry proposed in the 1920s and the 1930s [2]. However, after the discovery of paradoxes most logicians abandoned this approach. A notable exception was Haskell Curry and his school, but not much progress was made in establishing consistency of illative systems strong enough to interpret traditional logic. Only in the 1990s some first-order illative system were shown consistent and complete for traditional first-order logic [1,4]. The system $\mathcal{I}_s$, in terms of the features it provides, may be considered an extension of the illative system $\mathcal{I}_\omega$ from [3]. We briefly discuss the relationship between $\mathcal{I}_s$ and $\mathcal{I}_\omega$ in Sect. 5.

Because $\mathcal{I}_s$ is based on the untyped λ-calculus, its consistency is obviously open to doubt. In an appendix we give a proof by model construction of consistency of $\mathcal{I}_s$. Unfortunately, the proof is too long to fit within the page limits of a conference paper. In Sect. 3 we give a general overview of the proof. The model construction is similar to the one from [3] for the traditional illative system $\mathcal{I}_\omega$. It is extended and adapted to account for additional features of $\mathcal{I}_s$. To our knowlege $\mathcal{I}_s$ is the first higher-order illative system featuring subtypes and some form of induction, for which there is a consistency proof.

In Sect. 4 we provide examples of proofs in $\mathcal{I}_s$ indicating possible applications of our approach to the problem of dealing with partiality, non-termination and general recursion in higher-order logic. We are mainly interested in partiality arising from non-termination of non-well-founded recursive definitions.

For lack of space we omit proofs of the lemmas and theorems we state. The proofs of non-trivial results are in technical appendices to this paper. The appendices may be found in [5].

2 The Illative System

In this section we present the system $\mathcal{I}_s$ of illative classical higher-order logic with subtyping and derive some of its basic properties.

Definition 1. The system $\mathcal{I}_s$ consists of the following.

- A countably infinite set of variables $V_s = \{x, y, z, \ldots\}$ and a set of constants Σ_s.
- The set of sorts $\mathcal{S} = \{\mathrm{Type}, \mathrm{Prop}\}$.
- The set of *basic inductive types* $\mathcal{T}_I$ is defined inductively by the rule: if $\iota_{1,1}, \ldots, \iota_{1,n_1}, \ldots, \iota_{m,1}, \ldots, \iota_{m,n_m} \in \mathcal{T}_I \cup \{\star\}$ then
$$\mu(\langle \iota_{1,1}, \ldots, \iota_{1,n_1} \rangle, \ldots, \langle \iota_{m,1}, \ldots, \iota_{m,n_m} \rangle) \in \mathcal{T}_I$$
where $m \in \mathbb{N}_+$ and $n_1, \ldots, n_m \in \mathbb{N}$.
- We define the sets of *constructors* $\mathcal{C}$, *destructors* $\mathcal{D}$, and *tests* $\mathcal{O}$ as follows. For each $\iota \in \mathcal{T}_I$ of the form
$$\iota = \mu(\langle \iota_{1,1}, \ldots, \iota_{1,n_1} \rangle, \ldots, \langle \iota_{m,1}, \ldots, \iota_{m,n_m} \rangle) \in \mathcal{T}_I$$
where $\iota_{i,j} \in \mathcal{T}_I \cup \{\star\}$, the set $\mathcal{C}$ contains m distinct constants $c_1^\iota, \ldots, c_m^\iota$. The number n_i is called the *arity* of c_i^ι, and $\langle \iota_{i,1}, \ldots, \iota_{i,n_i} \rangle$ is its *signature*. With each $c_i^\iota \in \mathcal{C}$ of arity n_i we associate n_i distinct destructors $d_{i,1}^\iota, \ldots, d_{i,n_i}^\iota \in \mathcal{D}$ and one test $o_i^\iota \in \mathcal{O}$. When we use the symbols c_i^ι, o_i^ι and $d_{i,j}^\iota$ we implicitly assume that they denote the constructors, tests and destructors associated with ι. When it is clear from the context which type ι is meant, we use the notation $\iota_{i,j}^*$ for $\iota_{i,j}$ if $\iota_{i,j} \neq \star$, or for ι if $\iota_{i,j} = \star$.
- The set of $\mathcal{I}_s$-*terms* $\mathbb{T}$ is defined by the following grammar.
$$\begin{aligned}\mathbb{T} ::= {} & V_s \mid \Sigma_s \mid \mathcal{S} \mid \mathcal{C} \mid \mathcal{D} \mid \mathcal{O} \mid \mathcal{T}_I \mid \lambda V_s \,.\, \mathbb{T} \mid (\mathbb{T}\mathbb{T}) \mid \mathrm{Is} \mid \mathrm{Subtype} \mid \mathrm{Fun} \mid \\ & \forall \mid \vee \mid \bot \mid \epsilon \mid \mathrm{Eq} \mid \mathrm{Cond}\end{aligned}$$
We assume application associates to the left and omit spurious brackets.

- We identify α-equivalent terms, i.e. terms differing only in the names of bound variables are considered identical. We use the symbol $\equiv$ for identity of terms up to α-equivalence. We also assume that all bound variables in a term are distinct from the free variables, unless indicated otherwise.[1]
- In what follows we use the abbreviations:

$$\begin{aligned}
t_1 : t_2 &\equiv \mathrm{Is}\, t_1\, t_2 \\
\{x : \alpha \mid \varphi\} &\equiv \mathrm{Subtype}\, \alpha\, (\lambda x \,.\, \varphi) \\
\alpha \to \beta &\equiv \mathrm{Fun}\, \alpha\, \beta \\
\forall x : \alpha \,.\, \varphi &\equiv \forall\, \alpha\, (\lambda x \,.\, \varphi) \\
\forall x_1, \ldots, x_n : \alpha \,.\, \varphi &\equiv \forall x_1 : \alpha \,.\, \ldots \forall x_n : \alpha \,.\, \varphi \\
\varphi \supset \psi &\equiv \forall x : \{y : \mathrm{Prop} \mid \varphi\} \,.\, \psi \ \text{ where } x, y \notin FV(\varphi, \psi) \\
\neg\varphi &\equiv \varphi \supset \bot \\
\top &\equiv \bot \supset \bot \\
\varphi \vee \psi &\equiv \vee\varphi\psi \\
\varphi \wedge \psi &\equiv \neg(\neg\varphi \vee \neg\psi) \\
\exists x : \alpha \,.\, \varphi &\equiv \neg\forall x : \alpha \,.\, \neg\varphi
\end{aligned}$$

We assume that $\neg$ has the highest precedence.
- The system $\mathcal{I}_s$ is given by the following rules and axioms, where Γ is a finite set of terms, $t, \varphi, \psi, \alpha, \beta$, etc. are arbitrary terms. The notation Γ, φ is a shorthand for $\Gamma \cup \{\varphi\}$. We use Greek letters φ, ψ, etc. to highlight that a term is to be intuitively interpreted as a proposition, and we use α, β, etc. when it is to be interpreted as a type, but there is no a priori syntactic distinction. All judgements have the form $\Gamma \vdash t$ where Γ is a set of terms and t a term. In particular, $\Gamma \vdash t : \alpha$ is a shorthand for $\Gamma \vdash \mathrm{Is}\, t\, \alpha$.

Axioms

1: $\Gamma, \varphi \vdash \varphi$
2: $\Gamma \vdash \mathrm{Eq}\, t\, t$
3: $\Gamma \vdash \mathrm{Prop} : \mathrm{Type}$
4: $\Gamma \vdash \iota : \mathrm{Type}$ for $\iota \in \mathcal{T}_I$
5: $\Gamma \vdash o_i^\iota(c_i^\iota t_1 \ldots t_{n_i})$ if $c_i^\iota \in \mathcal{C}$ has arity n_i
6: $\Gamma \vdash \neg(o_i^\iota(c_j^\iota t_1 \ldots t_{n_j}))$ if $i \neq j$ and $c_j^\iota \in \mathcal{C}$ has arity n_j
7: $\Gamma \vdash \mathrm{Eq}\, (d_{i,k}^\iota(c_i^\iota t_1 \ldots t_{n_i}))\, t_k$ for $k = 1, \ldots, n_i$, if $c_i^\iota \in \mathcal{C}$ has arity n_i
$\bot_t$: $\Gamma \vdash \bot : \mathrm{Prop}$
c: $\Gamma \vdash \forall p : \mathrm{Prop} \,.\, p \vee \neg p$
β: $\Gamma \vdash \mathrm{Eq}\, ((\lambda x \,.\, t_1) t_2)\, (t_1[x/t_2])$

Rules

$$\forall_i : \frac{\Gamma \vdash \alpha : \mathrm{Type} \qquad \Gamma, x : \alpha \vdash \varphi \qquad x \notin FV(\Gamma, \alpha)}{\Gamma \vdash \forall x : \alpha \,.\, \varphi}$$

[1] So e.g. in the axiom β the free variables of t_2 do not become bound in $t_1[x/t_2]$.

$$\forall_e : \frac{\Gamma \vdash \forall x : \alpha \,.\, \varphi \qquad \Gamma \vdash t : \alpha}{\Gamma \vdash \varphi[x/t]}$$

$$\forall_t : \frac{\Gamma \vdash \alpha : \mathrm{Type} \qquad \Gamma, x : \alpha \vdash \varphi : \mathrm{Prop} \qquad x \notin FV(\Gamma, \alpha)}{\Gamma \vdash (\forall x : \alpha \,.\, \varphi) : \mathrm{Prop}}$$

$$\exists_i : \frac{\Gamma \vdash \alpha : \mathrm{Type} \qquad \Gamma \vdash t : \alpha \qquad \Gamma \vdash \varphi[x/t]}{\Gamma \vdash \exists x : \alpha \,.\, \varphi}$$

$$\exists_e : \frac{\Gamma \vdash \exists x : \alpha \,.\, \varphi \qquad \Gamma, x : \alpha, \varphi \vdash \psi \qquad x \notin FV(\Gamma, \psi, \alpha)}{\Gamma \vdash \psi}$$

$$\vee_{i1} : \frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi} \qquad\qquad \vee_{i2} : \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}$$

$$\vee_e : \frac{\Gamma \vdash \varphi_1 \vee \varphi_2 \qquad \Gamma, \varphi_1 \vdash \psi \qquad \Gamma, \varphi_2 \vdash \psi}{\Gamma \vdash \psi}$$

$$\vee_t : \frac{\Gamma \vdash \varphi : \mathrm{Prop} \qquad \Gamma \vdash \psi : \mathrm{Prop}}{\Gamma \vdash (\varphi \vee \psi) : \mathrm{Prop}}$$

$$\wedge_{e1} : \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi} \qquad\qquad \wedge_{e2} : \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}$$

$$\supset_{t2} : \frac{\Gamma \vdash (\varphi \supset \psi) : \mathrm{Prop}}{\Gamma \vdash \varphi : \mathrm{Prop}} \qquad\qquad \bot_e : \frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}$$

$$\to_i : \frac{\Gamma \vdash \alpha : \mathrm{Type} \qquad \Gamma, x : \alpha \vdash t : \beta \qquad x \notin FV(\Gamma, \alpha, \beta)}{\Gamma \vdash (\lambda x \,.\, t) : \alpha \to \beta}$$

$$\to_e : \frac{\Gamma \vdash t_1 : \alpha \to \beta \qquad \Gamma \vdash t_2 : \alpha}{\Gamma \vdash t_1 t_2 : \beta} \quad \to_t : \frac{\Gamma \vdash \alpha : \mathrm{Type} \qquad \Gamma \vdash \beta : \mathrm{Type}}{\Gamma \vdash (\alpha \to \beta) : \mathrm{Type}}$$

$$s_i : \frac{\Gamma \vdash \{x : \alpha \mid \varphi\} : \mathrm{Type} \qquad \Gamma \vdash t : \alpha \qquad \Gamma \vdash (\lambda x \,.\, \varphi)t \qquad x \notin FV(\alpha)}{\Gamma \vdash t : \{x : \alpha \mid \varphi\}}$$

$$s_e : \frac{\Gamma \vdash t : \{x : \alpha \mid \varphi\}}{\Gamma \vdash \varphi[x/t]} \qquad\qquad s_{et} : \frac{\Gamma \vdash t : \{x : \alpha \mid \varphi\}}{\Gamma \vdash t : \alpha}$$

$$s_t : \frac{\Gamma \vdash \alpha : \mathrm{Type} \qquad \Gamma, x : \alpha \vdash \varphi : \mathrm{Prop} \qquad x \notin FV(\alpha)}{\Gamma \vdash \{x : \alpha \mid \varphi\} : \mathrm{Type}}$$

$$\epsilon_i : \frac{\Gamma \vdash \exists x : \alpha \,.\, \top}{\Gamma \vdash (\epsilon\alpha) : \alpha} \qquad\qquad p_i : \frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi : \mathrm{Prop}}$$

$$c_1 : \frac{\Gamma \vdash \varphi}{\Gamma \vdash \mathrm{Eq}\,(\mathrm{Cond}\,\varphi\, t_1\, t_2\,)\, t_1} \qquad\qquad c_2 : \frac{\Gamma \vdash \neg\varphi}{\Gamma \vdash \mathrm{Eq}\,(\mathrm{Cond}\,\varphi\, t_1\, t_2\,)\, t_2}$$

$$c_3 : \frac{\Gamma, \varphi \vdash \mathrm{Eq}\, t_1\, t_1' \qquad \Gamma \vdash \varphi : \mathrm{Prop}}{\Gamma \vdash \mathrm{Eq}\,(\mathrm{Cond}\,\varphi\, t_1\, t_2\,)\,(\mathrm{Cond}\,\varphi\, t_1'\, t_2\,)}$$

$$c_4 : \frac{\Gamma, \neg\varphi \vdash \mathrm{Eq}\, t_2\, t_2' \qquad \Gamma \vdash \varphi : \mathrm{Prop}}{\Gamma \vdash \mathrm{Eq}\,(\mathrm{Cond}\,\varphi\, t_1\, t_2\,)\,(\mathrm{Cond}\,\varphi\, t_1\, t_2'\,)}$$

$$c_5 : \frac{\Gamma \vdash \varphi : \mathrm{Prop}}{\Gamma \vdash \mathrm{Eq}\,(\mathrm{Cond}\,\varphi\, t\, t\,)\, t}$$

$$\text{eq} : \frac{\Gamma \vdash \varphi \qquad \Gamma \vdash \mathrm{Eq}\, \varphi\, \varphi'}{\Gamma \vdash \varphi'} \qquad \text{eq-sym} : \frac{\Gamma \vdash \mathrm{Eq}\, t_1\, t_2}{\Gamma \vdash \mathrm{Eq}\, t_2\, t_1}$$

$$\text{eq-trans} : \frac{\Gamma \vdash \mathrm{Eq}\, t_1\, t_2 \qquad \Gamma \vdash \mathrm{Eq}\, t_2\, t_3}{\Gamma \vdash \mathrm{Eq}\, t_1\, t_3}$$

$$\text{eq-cong-app} : \frac{\Gamma \vdash \mathrm{Eq}\, t_1\, t_1' \qquad \Gamma \vdash \mathrm{Eq}\, t_2\, t_2'}{\Gamma \vdash \mathrm{Eq}\, (t_1 t_2)\, (t_1' t_2')}$$

$$\text{eq-}\lambda\text{-}\xi : \frac{\Gamma \vdash \mathrm{Eq}\, t\, t' \qquad x \notin FV(\Gamma)}{\Gamma \vdash \mathrm{Eq}\, (\lambda x \,.\, t)\, (\lambda x \,.\, t')}$$

$$i_i^{\iota} : \frac{\begin{array}{c}\Gamma, x_1 : \iota_{i,1}^*, \ldots, x_{n_i} : \iota_{i,n_i}^*, t x_{j_{i,1}}, \ldots, t x_{j_{i,k_i}} \vdash t(c_i^{\iota} x_1 \ldots x_{n_i}) \\ \text{for } i = 1, \ldots, m\end{array}}{\Gamma \vdash \forall x : \iota \,.\, t x}$$

where $x, x_1, \ldots, x_{n_i} \notin FV(\Gamma, t)$, $c_1^{\iota}, \ldots, c_m^{\iota} \in \mathcal{C}$ are all constructors associated with $\iota \in \mathcal{T}_I$, and $j_{i,1}, \ldots, j_{i,k_i}$ is an increasing sequence of all indices $1 \leq j \leq n_i$ such that $\iota_{i,j} = \star$

$$i_t^{\iota,k} : \frac{\Gamma \vdash t_j : \iota_{k,j}^* \text{ for } j = 1, \ldots, n_k}{\Gamma \vdash (c_k^{\iota} t_1 \ldots t_{n_k}) : \iota}$$

For an arbitrary set of terms Γ, we write $\Gamma \vdash_{\mathcal{I}_s} \varphi$ if there exists a finite subset $\Gamma' \subseteq \Gamma$ such that $\Gamma' \vdash \varphi$ is derivable in the system $\mathcal{I}_s$. We drop the subscript when irrelevant or obvious from the context.

Lemma 1. *If $\Gamma \vdash \varphi$ then $\Gamma, \psi \vdash \varphi$.*

Lemma 2. *If $\Gamma \vdash \varphi$ then $\Gamma[x/t] \vdash \varphi[x/t]$, where $\Gamma[x/t] = \{\psi[x/t] \mid \psi \in \Gamma\}$.*

2.1 Representing Logic

The inference rules of $\mathcal{I}_s$ may be intuitively justified by appealing to an informal many-valued semantics. A term t may be true, false, or something entirely different ("undefined", a program, a natural number, a type, ...). By way of an example, we explain an informal meaning of some terms:

- t : Prop is true iff t is true or false,
- α : Type is true iff α is a type,
- $t : \alpha$ is true iff t has type α, assuming α is a type,
- $\forall x : \alpha.\varphi$ is true iff α is a type and for all t of type α, $\varphi[x/t]$ is true,
- $\forall x : \alpha.\varphi$ is false iff α is a type and there exists t of type α such that $\varphi[x/t]$ is false,
- $t_1 \vee t_2$ is true iff t_1 is true or t_2 is true,
- $t_1 \vee t_2$ is false iff t_1 is false and t_2 is false,

- $t_1 \supset t_2$ is true iff t_1 is false or both t_1 and t_2 are true,
- $t_1 \supset t_2$ is false iff t_1 is true and t_2 is false,
- $\neg t$ is true iff t is false,
- $\neg t$ is false iff t is true.

Obviously, $\Gamma \vdash t$ is then (informally) interpreted as: for all possible substitution instances Γ^*, t^* of Γ, t, [2] if all terms in Γ^* are true, then the term t^* is also true.

Note that the logical connectives are "lazy", e.g. for $t_1 \vee t_2$ to be true it suffices that t_1 is true, but t_2 need not have a truth value at all – it may be something else: a program, a type, "undefined", etc. This laziness allows us to omit many restrictions which would otherwise be needed in inference rules, and would thus make the system less similar to ordinary logic.

The following rules may be derived in $\mathcal{I}_s$.

$$\supset_i\colon \frac{\Gamma \vdash \varphi : \mathrm{Prop} \qquad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \supset \psi} \qquad \supset_e\colon \frac{\Gamma, \varphi \vdash \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}$$

$$\supset_t\colon \frac{\Gamma \vdash \varphi : \mathrm{Prop} \qquad \Gamma, \varphi \vdash \psi : \mathrm{Prop}}{\Gamma \vdash (\varphi \supset \psi) : \mathrm{Prop}} \qquad \wedge_i : \frac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}$$

Note that in general the elimination rules for $\wedge$ and the rules for $\exists$ cannot be derived from the rules for $\vee$ and $\forall$, because we would not be able to prove the premise φ : Prop when trying to apply the rule $\supset_i$. It is instructive to try to derive these rules and see where the proof breaks down.

In $\mathcal{I}_s$ the only non-standard restriction in the usual inference rules for logical connectives is the additional premise $\Gamma \vdash \varphi$: Prop in the rule $\supset_i$. It is certainly unavoidable, as otherwise Curry's paradox may be derived (see e.g. [1,2]). However, we have standard classical higher-order logic if we restrict to terms of type Prop, in the sense that the natural deduction rules then become identical to the rules of ordinary logic. This is made more precise in Sect. 3 where a sound translation from a traditional system of higher-order logic into $\mathcal{I}_s$ is described.

Note that we have the law of excluded middle only in the form $\forall p : \mathrm{Prop} \,.\, p \vee \neg p$. Adding $\Gamma \vdash \varphi \vee \neg\varphi$ as an axiom for an arbitrary term φ gives an inconsistent system.[3]

It is well-known (see e.g. [6, Chapter 11]) that in higher-order logic all logical connectives may be defined from $\forall$ and $\supset$. One may therefore wonder why we take $\vee$ and $\bot$ as primitive. The answer is that if we defined the connectives from $\forall$ and $\supset$, then the inference rules that could be derived for them would need to contain additional restrictions.

2.2 Equality, Recursive Definitions and Extensionality

It is well-known (see e.g. [7, Chapters 2, 6]) that since untyped λ-terms are available together with the axiom β and usual rules for equality, any set of

[2] To be more precise, for every possible substitution of terms for the free variables of Γ, t we perform this substitution on Γ, t, denoting the result by Γ^*, t^*.

[3] By defining (see the next subsection) $\varphi = \neg\varphi$ one could then easily derive $\bot$ using the rule $\vee_e$ applied to $\varphi \vee \neg\varphi$.

equations of the form $\{z_i x_1 \ldots x_m = \Phi_i(z_1, \ldots, z_n, x_1, \ldots, x_m) \mid i = 1, \ldots, n\}$ has a solution for $z_1, \ldots, z_n$, where $\Phi_i(z_1, \ldots, z_n, x_1, \ldots, x_m)$ are arbitrary terms with the free variables listed. In other words, there exist terms $t_1, \ldots, t_n$ such that for any terms $s_1, \ldots, s_m$ we have $\vdash \mathrm{Eq}\ (t_i s_1 \ldots s_m)\ (\Phi_i(t_1, \ldots, t_n, s_1, \ldots, s_m))$ for each $i = 1, \ldots, n$.

We will often define terms by such equations. In what follows we freely use the notation $t_1 = t_2$ for $\vdash \mathrm{Eq}\, t_1\, t_2$, or for $\Gamma \vdash \mathrm{Eq}\, t_1\, t_2$ when it is clear which Γ is meant. We use $t_1 = t_2 = \ldots = t_n$ to indicate that $\mathrm{Eq}\, t_i\, t_{i+1}$ may be derived for $i = 1, \ldots, n-1$. We also write a term of the form $\mathrm{Eq}\, t_1\, t_2$ as $t_1 = t_2$.

In $\mathcal{I}_s$ there is no rule for typing the equality Eq. One consequence is that $\vdash \neg(\mathrm{Eq}\, t_1\, t_2)$ cannot be derived for any terms t_1, t_2.[4] For this reason Eq is more like a meta-level notion of equality.

Definition 2. Leibniz equality Eql is defined as:

$$\mathrm{Eql} \equiv \lambda\alpha\lambda x\lambda y.\forall p : \alpha \to \mathrm{Prop}\,.\, px \supset py$$

As with $=$, we often write $t_1 =_\alpha t_2$ to denote $\vdash \mathrm{Eql}\, \alpha\, t_1\, t_2$ or $\Gamma \vdash \mathrm{Eql}\, \alpha\, t_1\, t_2$, or write $t_1 =_\alpha t_2$ instead of $\mathrm{Eql}\, \alpha\, t_1\, t_2$.

Lemma 3. *If* $\Gamma \vdash \alpha : \mathrm{Type}$ *then*

- $\Gamma \vdash \forall x, y : \alpha\,.\,(x =_\alpha y) : \mathrm{Prop}$,
- $\Gamma \vdash \forall x : \alpha\,.\,(x =_\alpha x)$,
- $\Gamma \vdash \forall x, y : \alpha\,.\,(x =_\alpha y) \supset (y =_\alpha x)$,
- $\Gamma \vdash \forall x, y, z : \alpha\,.\,(x =_\alpha y) \wedge (y =_\alpha z) \supset (x =_\alpha z)$.

The system $\mathcal{I}_s$, as it is stated, is intensional with respect to Leibniz equality. We could add the rules

$$e_f : \frac{\Gamma \vdash \alpha : \mathrm{Type} \qquad \Gamma \vdash \beta : \mathrm{Type}}{\forall f_1, f_2 : \alpha \to \beta\,.\,(\forall x : \alpha\,.\, f_1 x =_\beta f_2 x) \supset (f_1 =_{\alpha\to\beta} f_2)}$$

$$e_b : \frac{\Gamma \vdash \varphi_1 \supset \varphi_2 \qquad \Gamma \vdash \varphi_2 \supset \varphi_1}{\Gamma \vdash \varphi_1 = \varphi_2}$$

to obtain an extensional variant $e\mathcal{I}_s$ of $\mathcal{I}_s$. The system $e\mathcal{I}_s$ is still consistent – the model we construct for $\mathcal{I}_s$ validates the above rules.

Lemma 4. $\vdash_{e\mathcal{I}_s} \forall x, y : \mathrm{Prop}\,.\,(x =_{\mathrm{Prop}} y) \supset (x = y)$

2.3 Induction and Natural Numbers

The system $\mathcal{I}_s$ incorporates basic inductive types. In accordance with the terminology from [8], an inductive type is basic if its constructors have no functional arguments. This class of inductive types includes most simple commonly used inductive types, e.g. natural numbers, lists, finite trees.

[4] We mean this in a precise sense. This follows from our model construction.

Lemma 5. *If $c_i^\iota \in \mathcal{C}$ of arity n_i has signature $\langle \iota_1, \ldots, \iota_{n_i} \rangle$ then $\vdash_{\mathcal{I}_s} c_i^\iota : \iota_1^* \to \ldots \to \iota_{n_i}^* \to \iota$.*

Lemma 6. $\vdash_{\mathcal{I}_s} o_i^\iota : \iota \to \mathrm{Prop}$ *and* $\vdash_{\mathcal{I}_s} \forall x : \iota \,.\, o_i^\iota x \supset (d_{i,j}^\iota x : \iota_{i,j}^*)$

Lemma 7. *If $\iota \in \mathcal{T}_I$ then $\vdash_{\mathcal{I}_s} \forall x, y : \iota \,.\, x =_\iota y \supset x = y$.*

We may define the type of natural numbers by $\mathrm{Nat} \equiv \mu(\langle\rangle, \langle\star\rangle)$. We use the abbreviations: $0 \equiv c_1^{\mathrm{Nat}}$ (zero), $\mathfrak{o} \equiv o_1^{\mathrm{Nat}}$ (test for zero), $\mathfrak{s} \equiv c_2^{\mathrm{Nat}}$ (successor) and $\mathfrak{p} \equiv \lambda x \,.\, \mathrm{Cond}\,(\mathfrak{o}x)\,0\,(d_{2,1}^{\mathrm{Nat}} x)$ (predecessor). The rules i_i^{Nat} and $i_t^{\mathrm{Nat},k}$ become:

$$n_i : \frac{\Gamma \vdash t0 \qquad \Gamma, x : \mathrm{Nat}, tx \vdash t(\mathfrak{s}x) \qquad x \notin FV(\Gamma, t)}{\Gamma \vdash \forall x : \mathrm{Nat} \,.\, tx}$$

$$n_t^1 : \frac{}{\Gamma \vdash 0 : \mathrm{Nat}} \qquad\qquad n_t^2 : \frac{\Gamma \vdash t : \mathrm{Nat}}{\Gamma \vdash (\mathfrak{s}t) : \mathrm{Nat}}$$

To simplify the exposition, we discuss some properties of our formulation of inductive types using the example of natural numbers. Much of what we say applies to other basic inductive types, with appropriate modifications.

The rule n_i is an induction principle for natural numbers. An important property of this induction principle is that it places no restrictions on t. This allows us to prove by induction on natural numbers properties of terms about which nothing is known beforehand. In particular, we do not need to know whether t has a β-normal form in order to apply the rule n_i to it. In contrast, an induction principle of the form e.g.

$$n_i' : \ \forall f : \mathrm{Nat} \to \mathrm{Prop} \,.\, ((f0 \wedge (\forall x : \mathrm{Nat} \,.\, fx \supset f(\mathfrak{s}x))) \supset \forall x : \mathrm{Nat} \,.\, fx)$$

would be much less useful, because to apply it to a term t we would have to prove $t : \mathrm{Nat} \to \mathrm{Prop}$ *beforehand*. Examples of the use of the rule n_i for reasoning about possibly nonterminating general recursive programs are given in Sect. 4.

The operations $+$, $-$, $\cdot$, $<$ and $\leq$, usually used in infix notation, may be defined by recursive equations. It is possible to derive all Peano axioms.

Lemma 8. *The following terms are derivable in $\mathcal{I}_s$:*

– $\forall x, y : \mathrm{Nat} \,.\, (x + y) : \mathrm{Nat}$, $\forall x, y : \mathrm{Nat} \,.\, (x - y) : \mathrm{Nat}$, $\forall x, y : \mathrm{Nat} \,.\, (x \cdot y) : \mathrm{Nat}$,
– $\forall x, y : \mathrm{Nat} \,.\, (x \leq y) : \mathrm{Prop}$, $\forall x, y : \mathrm{Nat} \,.\, (x < y) : \mathrm{Prop}$.

The next theorem shows that any function for which there exists a measure on its arguments, which may be shown to decrease with every recursive call in each of a finite number of exhaustive cases, is typable in our system.

Theorem 1. *Suppose $\Gamma \vdash \forall x_1 : \alpha_1 \ldots \forall x_n : \alpha_n \,.\, \varphi_1 \vee \ldots \vee \varphi_m$, $\Gamma \vdash \alpha_j : \mathrm{Type}$ for $j = 1, \ldots, n$, and for $i = 1, \ldots, m$: $\Gamma \vdash \forall x_1 : \alpha_1 \ldots \forall x_n : \alpha_n \,.\, t_i : \beta \to \ldots \to \beta$ where β occurs $k_i + 1$ times, $\Gamma \vdash \forall x_1 : \alpha_1 \ldots \forall x_n : \alpha_n \,.\, t_{i,j,k} : \alpha_k$ for $j = 1, \ldots, k_i$, $k = 1, \ldots, n$, $x_1, \ldots, x_n \notin FV(f, \alpha_1, \ldots, \alpha_n, \beta)$ and*

$$\Gamma \vdash \forall x_1 : \alpha_1 \ldots \forall x_n : \alpha_n . \varphi_i \supset (f x_1 \ldots x_n = t_i(f t_{i,1,1} \ldots t_{i,1,n}) \ldots (f t_{i,k_i,1} \ldots t_{i,k_i,n})).$$

If there is a term g such that $\Gamma \vdash g : \alpha_1 \to \ldots \to \alpha_n \to \mathrm{Nat}$ and for $i = 1, \ldots, m$

$$\Gamma \vdash \forall x_1 : \alpha_1 \ldots \forall x_n : \alpha_n . \varphi_i \supset (((f x_1 \ldots x_n) : \beta) \vee ((g t_{i,1,1} \ldots t_{i,1,n}) < (g x_1 \ldots x_n) \wedge \ldots \wedge (g t_{i,k_i,1} \ldots t_{i,k_i,n}) < (g x_1 \ldots x_n)))$$

where $x_1, \ldots, x_n \notin FV(g)$, then $\Gamma \vdash f : \alpha_1 \to \ldots \to \alpha_n \to \beta$.

3 Conservativity and Consistency

In this section we show a sound embedding of ordinary classical higher-order logic into $\mathcal{I}_s$, which we also conjecture to be complete. We have a completeness proof only for a restriction of this embedding to first-order logic. We also give a brief overview of the model construction used to establish consistency of $\mathcal{I}_s$.

First, let us define the system CPREDω of classical higher-order logic.

– The *types* of CPREDω are given by $\mathcal{T} ::= o \mid \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$ where $\mathcal{B}$ is a specific finite set of base types. The type o is the type of propositions.
– The set of terms of CPREDω of type τ, denoted T_τ, is defined as follows:
  - $V_\tau, \Sigma_\tau \subseteq T_\tau$,
  - if $t_1 \in T_{\sigma\to\tau}$ and $t_2 \in T_\sigma$ then $t_1 t_2 \in T_\tau$,
  - if $x \in V_{\tau_1}$ and $t \in T_{\tau_2}$ then $\lambda x : \tau_1 . t \in T_{\tau_1 \to \tau_2}$,
  - if $\varphi, \psi \in T_o$ then $\varphi \supset \psi \in T_o$,
  - if $x \in V_\tau$ and $\varphi \in T_o$ then $\forall x : \tau . \varphi \in T_o$,

  where for each $\tau \in \mathcal{T}$ the set V_τ is a set of variables and Σ_τ is a set of constants. We assume that the sets V_τ and Σ_σ are all pairwise disjoint. We write x_τ for a variable $x_\tau \in V_\tau$. Terms of type o are *formulas*.
– The system CPREDω is given by the following rules and axioms, where Δ is a finite set of formulas, φ, ψ are formulas.

  Axioms
  - $\Delta, \varphi \vdash \varphi$
  - $\Delta \vdash \forall p : o . ((p \supset \bot) \supset \bot) \supset p$ where $\bot \equiv \forall p : o . p$

  Rules

$$\supset_i^P : \frac{\Delta, \varphi \vdash \psi}{\Delta \vdash \varphi \supset \psi} \qquad \supset_e^P : \frac{\Delta \vdash \varphi \supset \psi \quad \Delta \vdash \varphi}{\Delta \vdash \psi}$$

$$\forall_i^P : \frac{\Delta \vdash \varphi}{\Delta \vdash \forall x_\tau . \varphi}\ x_\tau \notin FV(\Delta) \qquad \forall_e^P : \frac{\Delta \vdash \forall x_\tau . \varphi}{\Delta \vdash \varphi[x_\tau / t]}\ t \in T_\tau$$

$$\mathrm{conv}^P : \frac{\Delta \vdash \varphi \qquad \varphi =_\beta \psi}{\Delta \vdash \psi}$$

In CPREDω, we define Leibniz equality in type $\tau \in \mathcal{T}$ by

$$t_1 =_\tau t_2 \equiv \forall p : \tau \to o \,.\, pt_1 \supset pt_2.$$

The system CPREDω is intensional. An extensional variant E-CPREDω may be obtained by adding the following axioms for all $\tau, \sigma \in \mathcal{T}$:

$$e_f^P : \forall f_1, f_2 : \tau \to \sigma \,.\, (\forall x : \tau \,.\, f_1 x =_\sigma f_2 x) \supset (f_1 =_{\tau \to \sigma} f_2)$$
$$e_b^P : \forall \varphi_1, \varphi_2 : o \,.\, ((\varphi_1 \supset \varphi_2) \wedge (\varphi_2 \supset \varphi_1)) \supset (\varphi_1 =_o \varphi_2)$$

For an arbitrary set of formulas Δ we write $\Delta \vdash_S \varphi$ if φ is derivable from a subset of Δ in system S.

We now define a mapping $\lceil - \rceil$ from types and terms of CPREDω to terms of $\mathcal{I}_s$, and a mapping $\Gamma(-)$ from sets of terms of CPREDω to sets of terms of $\mathcal{I}_s$. We assume $\mathcal{B} \subseteq \Sigma_s$, $\Sigma_\tau \subseteq \Sigma_s$ and $V_\tau \subseteq V_s$ for $\tau \in \mathcal{T}$.

- $\lceil \tau \rceil = \tau$ for $\tau \in \mathcal{B}$,
- $\lceil o \rceil = \mathrm{Prop}$,
- $\lceil \tau_1 \to \tau_2 \rceil = \lceil \tau_1 \rceil \to \lceil \tau_2 \rceil$ for $\tau_1, \tau_2 \in \mathcal{T}$,
- $\lceil c \rceil = c$ if $c \in \Sigma_\tau$ for some $\tau \in \mathcal{T}$,
- $\lceil x \rceil = x$ if $x \in V_\tau$ for some $\tau \in \mathcal{T}$,
- $\lceil t_1 t_2 \rceil = \lceil t_1 \rceil \lceil t_2 \rceil$,
- $\lceil \lambda x : \tau \,.\, t \rceil = \lambda x \,.\, \lceil t \rceil$,
- $\lceil \varphi \supset \psi \rceil = \lceil \varphi \rceil \supset \lceil \psi \rceil$,
- $\lceil \forall x : \tau \,.\, \varphi \rceil = \forall x : \lceil \tau \rceil \,.\, \lceil \varphi \rceil$.

By $\lceil \Delta \rceil$ we denote the image of $\lceil - \rceil$ on Δ. The set $\Gamma(\Delta)$ is defined to contain:

- $x : \lceil \tau \rceil$ for all $\tau \in \mathcal{T}$ and all $x \in FV(\Delta)$ such that $x \in V_\tau$,
- $c : \lceil \tau \rceil$ for all $\tau \in \mathcal{T}$ and all $c \in \Sigma_\tau$,
- $\tau : \mathrm{Type}$ for all $\tau \in \mathcal{B}$,
- $y : \tau$ for all $\tau \in \mathcal{B}$ and some $y \in V_\tau$ such that $y \notin FV(\Delta)$.

Theorem 2. *If $\Delta \vdash_{\mathrm{CPRED}\omega} \varphi$ then $\Gamma(\Delta, \varphi), \lceil \Delta \rceil \vdash_{\mathcal{I}_s} \lceil \varphi \rceil$. The same holds if we change CPREDω to E-CPREDω and $\mathcal{I}_s$ to $e\mathcal{I}_s$.*

The above theorem shows that $\mathcal{I}_s$ may be considered an extension of ordinary higher-order logic, obtained by relaxing typing requirements on allowable λ-terms. Type-checking is obviously undecidable in $\mathcal{I}_s$, but the purpose of types in illative systems is not to have a decidable method for syntactic correctness checks, but to provide general means for classifying terms into various categories.

Conjecture 1. If $\Gamma(\Delta, \varphi), \lceil \Delta \rceil \vdash_{\mathcal{I}_s} \lceil \varphi \rceil$ then $\Delta \vdash_{\mathrm{CPRED}\omega} \varphi$. The same holds if we change CPREDω to E-CPREDω and $\mathcal{I}_s$ to $e\mathcal{I}_s$.

We prove this conjecture only for first-order logic. The system of classical first-order logic (FOL) is obtained by restricting CPREDω in obvious ways.

Theorem 3. *If $\mathcal{I} = \mathcal{I}_s$ or $\mathcal{I} = e\mathcal{I}_s$ then: $\Delta \vdash_{\mathrm{FOL}} \varphi$ iff $\Gamma(\Delta, \varphi), \lceil \Delta \rceil \vdash_{\mathcal{I}} \lceil \varphi \rceil$.*

Theorem 4. *The systems $\mathcal{I}_s$ and $e\mathcal{I}_s$ are consistent, i.e. $\not\vdash_{\mathcal{I}_s} \bot$ and $\not\vdash_{e\mathcal{I}_s} \bot$.*

We now give an informal overview of the model construction. To simplify the exposition we pretend $\mathcal{I}_s$ allows only function types. Other types add some technicalities, but the general idea of the construction remains the same.

An $\mathcal{I}_s$-model is defined as a λ-model (see e.g. [7, Chapter 5]) with designated elements interpreting the constants of $\mathcal{I}_s$, satisfying certain conditions. By $[\![t]\!]^{\mathcal{M}}$ we denote the interpretation of the $\mathcal{I}_s$-term t in a model $\mathcal{M}$, and $\cdot$ is the application operation in the model. The conditions imposed on an $\mathcal{I}_s$-model express the meaning of each rule of $\mathcal{I}_s$ according to the intuitive semantics. For instance, we have the condition:

($\forall_\top$) for $a \in \mathcal{M}$, if $[\![\mathrm{Is}]\!]^{\mathcal{M}} \cdot a \cdot [\![\mathrm{Type}]\!]^{\mathcal{M}} = [\![\top]\!]^{\mathcal{M}}$ and for all $c \in \mathcal{M}$ such that $[\![\mathrm{Is}]\!]^{\mathcal{M}} \cdot c \cdot a = [\![\top]\!]^{\mathcal{M}}$ we have $b \cdot c = [\![\top]\!]^{\mathcal{M}}$ then $[\![\forall]\!]^{\mathcal{M}} \cdot a \cdot b = [\![\top]\!]^{\mathcal{M}}$.

We show that the semantics based on $\mathcal{I}_s$-models is sound for $\mathcal{I}_s$. Then it suffices to construct a non-trivial $\mathcal{I}_s$-model to establish consistency of $\mathcal{I}_s$. The model will in fact satisfy additional conditions corresponding to the rules e_f and e_b, so we obtain consistency of $e\mathcal{I}_s$ as well.

The model is constructed as the set of equivalence classes of a certain relation $\overset{*}{\Leftrightarrow}$ on the set of so called semantic terms. A semantic term is a well-founded tree whose leaves are labelled with variables or constants, and whose internal nodes are labelled with $\cdot$, λx or $\mathrm{A}\tau$. For semantic terms with the roots labelled with $\cdot$ and λx we use the notation $t_1 t_2$ and $\lambda x.t$. A node labelled with $\mathrm{A}\tau$, where τ is a set of constants, "represents" the statement: for all $c \in \tau$, tc is true. Such a node has one child for each $c \in \tau$. The relation $\overset{*}{\Leftrightarrow}$ is defined as the equivalence relation generated by a certain reduction relation $\Rightarrow$ on semantic terms. The relation $\Rightarrow$ will satisfy[5]: $(\lambda x.t_1)t_2 \Rightarrow t_1[x/t_2]$, $\vee\top t \Rightarrow \top$, $\vee\bot\bot \Rightarrow \bot$, etc. The question is how to define $\Rightarrow$ for $\forall t_1 t_2$ so that the resulting structure satisfies ($\forall_\top$). One could try closing $\Rightarrow$ under the rule: if $\mathrm{Is}\, t_1\, \mathrm{Type} \overset{*}{\Rightarrow} \top$ and for all t such that $\mathrm{Is}\, t\, t_1 \overset{*}{\Rightarrow} \top$ we have $t_2 t \overset{*}{\Rightarrow} \top$, then $\forall t_1 t_2 \Rightarrow \top$. However, there is a negative reference to $\Rightarrow$ here, so the definition would not be monotone, and we would not necessarily reach a fixpoint. This is a major problem. We need to know the range of all quantifiers beforehand. However, the range (i.e. the set of all semantic terms t such that $t_1 t \overset{*}{\Rightarrow} \top$) depends on the definition of $\Rightarrow$, so it is not at all clear how to achieve this.

Fortunately, it is not so difficult to analyze a priori the form of types of $\mathcal{I}_s$. Informally, if $t : \mathrm{Type}$ is true, then t corresponds to a set in $\mathcal{T}$, where $\mathcal{T}$ is defined as follows, ignoring subtypes and inductive types.

– $\mathrm{Bool} \in \mathcal{T}$ where $\mathrm{Bool} = \{\top, \bot\}$.
– If $\tau_1, \tau_2 \in \mathcal{T}$ then $\tau_2^{\tau_1} \in \mathcal{T}$, where $\tau_2^{\tau_1}$ is the set of all set-theoretic functions from τ_1 to τ_2.

We take the elements of $\mathcal{T}$ and $\bigcup \mathcal{T} \setminus \mathrm{Bool}$ as fresh constants, i.e. they may occur as constants in semantic terms. The elements of $\bigcup \mathcal{T}$ are *canonical constants*. If

[5] Substitution is defined for semantic terms in an obvious way, avoiding variable capture.

$c \in \tau_2^{\tau_1}$ and $c_1 \in \tau_1$ then we write $\mathcal{F}(c)(c_1)$ instead of $c(c_1)$ to avoid confusion with the semantic term cc_1. We then define a relation $\succ$ satisfying:

- $c \succ c$ for a canonical constant c,
- if $c \in \tau_2^{\tau_1}$ and for all $c_1 \in \tau_1$ there exists a semantic term t' such that $tc_1 \overset{*}{\Rightarrow} t' \succ \mathcal{F}(c)(c_1)$, then $t \succ c$.

Intuitively, $t \succ c \in \tau$ holds if c "simulates" t in type τ, i.e. t behaves exactly like c in every context where a term of type τ is "expected".

The relation $\Rightarrow$ is then defined by transfinite induction in a monotone way. It will satisfy e.g.:

- if $t \succ c \in \tau \in \mathcal{T}$ then $\mathrm{Is}\, t\, \tau \Rightarrow \top$,
- if $t \succ c_1 \in \tau_1$ and $c \in \tau_2^{\tau_1}$ then $ct \Rightarrow \mathcal{F}(c)(c_1)$,
- $\mathrm{Fun}\, \tau_1\, \tau_2 \Rightarrow \tau_2^{\tau_1}$,
- $\forall \tau t \Rightarrow t'$ where the label at the root of t' is $\mathrm{A}\tau$, and for each $c \in \tau$, t' has a child tc,
- $t \Rightarrow \top$ if the label of the root of t is $\mathrm{A}\tau$, and all children of t are labelled with $\top$,
- if $t_c \overset{*}{\Rightarrow} t'_c$ for all $c \in \tau \in \mathcal{T}$, the label of the root of t is $\mathrm{A}\tau$, and $\{t_c \mid c \in \tau\}$ is the set of children of t, then $t \Rightarrow t'$, where the label of the root of t' is $\mathrm{A}\tau$ and $\{t'_c \mid c \in \tau\}$ is the set of children of t'.

We removed negative references to $\Rightarrow$, but it is not easy to show that the resulting model satisfies the required conditions. Two key properties established in the correctness proof are: 1. $\Rightarrow$ has the Church-Rosser property, and 2. if $t_2 \succ c$ and $t_1 c \overset{*}{\Rightarrow} d \in \{\top, \bot\}$ then $t_1 t_2 \overset{*}{\Rightarrow} d$. The second property shows that quantifying over only canonical constants of type τ is in a sense equivalent to quantifying over all terms of type τ. This is essential for establishing e.g. the condition $(\forall_\top)$.

Both properties have intricate proofs. Essentially, the proofs show certain commutation and postponement properties for $\Rightarrow$, $\succ$ and other auxiliary relations. The proofs proceed by induction on lexicographic products of various ordinals and other parameters associated with the relations and terms involved.

4 Partiality and General Recursion

In this section we give some examples of proofs in $\mathcal{I}_s$ of properties of functions defined by recursion. For lack of space, we give only informal indications of how formal proofs may be obtained, assuming certain basic properties of operations on natural numbers. The transformation of the given informal arguments into formal proofs in $\mathcal{I}_s$ is not difficult. Mostly complete formal proofs may be found in a technical appendix.

Example 1. Consider a term subp satisfying the following recursive equation:

$$\mathrm{subp} = \lambda i j \,.\, \mathrm{Cond}\, (i =_{\mathrm{Nat}} j)\, 0\ ((\mathrm{subp}\, i\ (j+1)) + 1)\ .$$

If $i \geq j$ then $\mathrm{subp}\, i\, j = i - j$. If $i < j$ then $\mathrm{subp}\, i\, j$ does not terminate. An appropriate specification for subp is $\forall i, j : \mathrm{Nat} \,.\, (i \geq j) \supset (\mathrm{subp}\, x = i - j)$.

Let $\varphi(y) = \forall i : \mathrm{Nat} \,.\, \forall j : \mathrm{Nat} \,.\, (i \geq j \supset y =_{\mathrm{Nat}} i - j \supset \mathrm{subp}\, i\, j = i - j)$. We show by induction on y that $\forall y : \mathrm{Nat} \,.\, \varphi(y)$.

First note that under the assumptions $y : \mathrm{Nat}$, $i : \mathrm{Nat}$, $j : \mathrm{Nat}$ it follows from Lemma 8 that $(i \geq j) : \mathrm{Prop}$ and $(y =_{\mathrm{Nat}} i - j) : \mathrm{Prop}$. Hence, whenever $y : \mathrm{Nat}$, to show $i \geq j \supset y =_{\mathrm{Nat}} i - j \supset \mathrm{subp}\, i\, j = i - j$ it suffices to derive $\mathrm{subp}\, i\, j = i - j$ under the assumptions $i \geq j$ and $y =_{\mathrm{Nat}} i - j$. By Lemma 7 the assumption $y =_{\mathrm{Nat}} i - j$ may be weakened to $y = i - j$.

In the base step it thus suffices to show $\mathrm{subp}\, i\, j = i - j$ under the assumptions $i : \mathrm{Nat}$, $j : \mathrm{Nat}$, $i \geq j$, $i - j = 0$. From $i - j = 0$ we obtain $\mathfrak{o}(i - j)$, so $j \geq i$. From $i \geq j$ and $i \leq j$ we derive $i =_{\mathrm{Nat}} j$. Then $\mathrm{subp}\, i\, j = i - j$ follows by simple computation (i.e. by applying rules for Eq and appropriate rules for the conditional).

In the inductive step we have $\varphi(y)$ for $y : \mathrm{Nat}$ and we need to obtain $\varphi(\mathfrak{s}y)$. It suffices to show $\mathrm{subp}\, i\, j = i - j$ under the assumptions $i : \mathrm{Nat}$, $j : \mathrm{Nat}$ and $\mathfrak{s}y = i - j$. Because $\mathfrak{s}y \neq_{\mathrm{Nat}} 0$ we have $i \neq_{\mathrm{Nat}} j$, hence $\mathrm{subp}\, i\, j = \mathfrak{s}(\mathrm{subp}\, i\, (\mathfrak{s}j))$ follows by computation. Using the inductive hypothesis we now conclude $\mathrm{subp}\, i\, (\mathfrak{s}j) = i - (\mathfrak{s}j)$, and thus $\mathrm{subp}\, i\, (\mathfrak{s}j) =_{\mathrm{Nat}} i - (\mathfrak{s}j)$ by reflexivity of $=_{\mathrm{Nat}}$ on natural numbers. Then it follows by properties of operations on natural numbers that $\mathfrak{s}(\mathrm{subp}\, i\, (\mathfrak{s}j)) =_{\mathrm{Nat}} i - j$. By Lemma 7 we obtain the thesis.

We have thus completed an inductive proof of $\forall y : \mathrm{Nat} \,.\, \varphi(y)$. Now we use this formula to derive $\mathrm{subp}\, i\, j = i - j$ under the assumptions $i : \mathrm{Nat}$, $j : \mathrm{Nat}$, $i \geq j$. Then it remains to apply $\supset_i$ and $\forall_i$ twice.

In the logic of PVS [9] one may define subp by specifying its domain precisely using predicate subtypes and dependent types, somewhat similarly to what is done here. However, an important distinction is that we do not require a domain specification to be a part of the definition. Because of this, we may easily derive $\varphi \equiv \forall i, j : \mathrm{Nat} \,.\, ((\mathrm{subp}\, i\, j = i - j) \vee (\mathrm{subp}\, j\, i = j - i))$. This is not possible in PVS because the formula φ translated to PVS generates false proof obligations [9].

Example 2. The next example is a well-known "challenge" posed by McCarthy:

$$f(n) = \mathrm{Cond}\, (n > 100)\, (n - 10)\, (f(f(n + 11)))$$

For $n \leq 101$ we have $f(n) = 91$, which fact may be proven by induction on $101 - n$. This function is interesting because of its use of nested recursion. Termination behavior of a nested recursive function may depend on its functional behavior, which makes reasoning about termination and function value interdependent. Below we give an indication of how a formal proof of $\forall n : \mathrm{Nat} \,.\, n \leq 101 \supset f(n) = 91$ may be derived in $\mathcal{I}_s$. Lemma 8 is used implicitly with implication introduction.

Let $\varphi(y) \equiv \forall n : \mathrm{Nat} \,.\, n \leq 101 \supset 101 - n \leq y \supset f(n) = 91$. We prove $\forall y : \mathrm{Nat} \,.\, \varphi(y)$ by induction on y. In the base step we need to prove $f(n) = 91$ under the assumptions $n : \mathrm{Nat}$, $n \leq 101$ and $101 - n \leq y = 0$. We have $n =_{\mathrm{Nat}} 101$, hence $n = 101$, and the thesis follows by simple computation.

In the inductive step we distinguish three cases: 1. $n+11>101$ and $n<101$, 2. $n+11>101$ and $n\geq 101$, 3. $n+11\leq 101$. We need to prove $f(n)=91$ under the assumptions of the inductive hypothesis $y:\mathrm{Nat}, \forall m:\mathrm{Nat}\,.\,m\leq 101\supset 101-m\leq y\supset f(m)=91$, and of $n:\mathrm{Nat}$, $n\leq 101$ and $101-n\leq(\mathfrak{s}y)$.

We treat only the third case, other cases being similar. From $101-n\leq s(y)$ we infer $101-(n+11)\leq y$. Since $n+11\leq 101$ we conclude by the inductive hypothesis that $f(n+11)=91$. Because $n+11\leq 101$, so $n\leq 100$, and by definition we infer $f(n)=f(f(n{+}11))=f(91)$. Now we simply compute $f(91)=f(f(102))=f(92)=f(f(103))=\ldots=f(100)=f(f(111))=f(101)=91$ (i.e. we apply rules for Eq and Cond an appropriate number of times).

This concludes the inductive proof of $\forall y:\mathrm{Nat}\,.\,\forall n:\mathrm{Nat}\,.\,n\leq 101\supset 101-n\leq y\supset f(n)=91$. Having this it is easy to show $\forall n:\mathrm{Nat}\,.\,n\leq 101\supset f(n)=91$.

Note that the computation of $f(91)$ in the inductive step relies on the fact that in our logic values of functions may always be computed for specific arguments, regardless of what we know about the function.

5 Related Work

In this section we discuss the relationship between $\mathcal{I}_s$ and the traditional illative system $\mathcal{I}_\omega$. We also briefly survey some approaches to dealing with partiality and general recursion in proof assistants. A general overview of the literature relevant to this problem may be found in [10].

5.1 Relationship with Systems of Illative Combinatory Logic

In terms of the features provided, the system $\mathcal{I}_s$ may be considered an extension of $\mathcal{I}_\omega$ from [3], which is a direct extension of $\mathcal{I}\Xi$ from [1] to higher-order logic. The ideas behind $\mathcal{I}_\omega$ date back to [11], or even earlier as far as the general form of inference rules is concerned.

However, there are some technical differences between $\mathcal{I}_s$ and traditional illative systems. For one thing, traditional systems strive to use as few constants and rules as possible. For instance, $\mathcal{I}_\omega$ has only two primitive constants, disregarding constants representing base types. Because of this in $\mathcal{I}_\omega$ e.g. $\mathrm{Is}=\lambda xy\,.\,yx$ and $\mathrm{Prop}=\lambda x\,.\,\mathrm{Type}(\lambda y.x)$, using the notation of the present paper. Moreover, the names of the constants and the notations employed are not in common use today. We will not explain these technicalities in any more detail. The reader may consult [2,1,3] for more information on illative combinatory logic.

Below we briefly describe a system $\mathcal{I}'_\omega$ which is a variant of $\mathcal{I}_\omega$ adapted to our notation. It differs somewhat from $\mathcal{I}_\omega$, mostly by taking more constants as primitive. The terms of $\mathcal{I}'_\omega$ are those of $\mathcal{I}_s$, except that we do not allow subtypes, inductive types, Eq, Cond, $\vee$, $\bot$ and ϵ. There are also additional constants: ω (the type of all terms), ε (the empty type) and $\supset$. The axioms are: $\Gamma,\varphi\vdash\varphi$, $\Gamma\vdash\mathrm{Prop}:\mathrm{Type}$, $\Gamma\vdash\varepsilon:\mathrm{Type}$, $\Gamma\vdash\omega:\mathrm{Type}$, $\Gamma\vdash t:\omega$. The rules are: $\forall_i$, $\forall_e$, $\forall_t$, $\supset_i$, $\supset_e$, $\supset_t$, $\rightarrow_i$, $\rightarrow_e$, $\rightarrow_t$, p_i, and the rules:

$$\text{conv}: \frac{\Gamma \vdash \varphi \qquad \varphi =_\beta \psi}{\Gamma \vdash \psi} \qquad\qquad \varepsilon_\bot : \frac{\Gamma \vdash t : \varepsilon}{\Gamma \vdash \bot}$$

$$\to_p: \frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma, x : \alpha \vdash ((tx) : \beta) : \text{Prop} \qquad x \notin FV(\Gamma, t)}{\Gamma \vdash (t : \alpha \to \beta) : \text{Prop}}$$

5.2 Partiality and Recursion in Proof Assistants

Perhaps the most common way of dealing with recursion in interactive theorem provers is to impose certain syntactic restrictions on the form of recursive definitions so as to guarantee well-foundedness. Well-foundedness of definitions is then checked by a built-in automatic syntactic termination checker. Some systems, e.g. ACL2 or PVS, pass the task of proving termination to the user. Such systems require that a well-founded relation or a measure be given with each recursive function definition. Then the system generates so called proof obligations, to be shown by the user, which state that the recursive calls are made on smaller arguments.

The method of restricting possible forms of recursive definitions obviously works only for total functions. If a function does not in fact terminate on some elements of its specified domain, then it cannot be introduced by a well-founded definition. One solution is to use a rich type system, e.g. dependent types combined with predicate subtyping, to precisely specify function domains so as to rule out the arguments on which the function does not terminate. This approach is adopted by PVS [9].

A different approach to dealing with partiality and general recursion is to use a special logic which allows partial functions directly. Systems adopting this approach are often based on variants of the logic of partial terms of Beeson [12,13]. For instance, the IMPS interactive theorem prover [14] uses Farmer's logic PF of partial functions [15], which is essentially a variant of the logic of partial terms adapted to higher-order logic.

The above gives only a very brief overview. There are many approaches to the problem of partiality and general recursion in proof assistants, most of which we didn't mention. We do not attempt here to provide a detailed comparison with a multitude of existing approaches or give in-depth arguments in favor of our system. For such arguments to be entirely convincing, they would need to be backed up by extensive experimentation in proving properties of sizable programs using our logic. No such experimentation has been undertaken. In contrast, our interest is theoretical.

6 Conclusion

We have presented a system $\mathcal{I}_s$ of classical higher-order illative λ-calculus with subtyping and basic inductive types. A distinguishing characteristic of $\mathcal{I}_s$ is that it is based on the untyped λ-calculus. Therefore, it allows recursive definitions of potentially non-terminating functions directly. The inference rules of $\mathcal{I}_s$ are formulated in a way that makes it possible to apply them even when some of the terms used in the premises have not been proven to belong to any type. Additionally, our system may be considered an extension of ordinary higher-order logic, obtained by relaxing the typing restrictions on allowable λ-terms. We believe these facts alone make it relevant to the problem of partiality and recursion in proof assistants, and the system at least deserves some attention.

References

1. Barendregt, H., Bunder, M.W., Dekkers, W.: Systems of illative combinatory logic complete for first-order propositional and predicate calculus. Journal of Symbolic Logic 58(3), 769–788 (1993)
2. Seldin, J.P.: The logic of Church and Curry. In: Gabbay, D.M., Woods, J. (eds.) Logic from Russell to Church. Handbook of the History of Logic, vol. 5, pp. 819–873. North-Holland (2009)
3. Czajka, Ł.: Higher-order illative combinatory logic. Journal of Symbolic Logic (2013), `http://arxiv.org/abs/1202.3672` (accepted)
4. Dekkers, W., Bunder, M.W., Barendregt, H.: Completeness of the propositions-as-types interpretation of intuitionistic logic into illative combinatory logic. Journal of Symbolic Logic 63(3), 869–890 (1998)
5. Czajka, Ł.: Partiality and recursion in higher-order logic. Technical report, University of Warsaw (2013), `http://arxiv.org/abs/1210.2039`
6. Sørensen, M.H., Urzyczyn, P.: Lectures on the Curry-Howard isomorphism. Studies in Logic and the Foundations of Mathematics, vol. 149. Elsevier, Amsterdam (2006)
7. Barendregt, H.P.: The lambda calculus: Its syntax and semantics. Revised edn. North Holland (1984)
8. Blanqui, F., Jouannaud, J., Okada, M.: Inductive-data-type systems. Theoretical Computer Science 272(1), 41–68 (2002)
9. Rushby, J., Owre, S., Shankar, N.: Subtypes for specifications: Predicate subtyping in PVS. IEEE Transactions on Software Engineering 24(9) (1998)
10. Bove, A., Krauss, A., Sozeau, M.: Partiality and recursion in interactive theorem provers: An overview. Mathematical Structures in Computer Science (2012) (to appear)
11. Bunder, M.W.: Predicate calculus of arbitrarily high finite order. Archive for Mathematical Logic 23(1), 1–10 (1983)
12. Feferman, S.: Definedness. Erkenntnis 43, 295–320 (1995)
13. Beeson, M.J.: Proving programs and programming proofs. In: Marcus, R.B., Dorn, G., Weingartner, P. (eds.) Logic, Methodology and Philosophy of Science VII, pp. 51–82. North-Holland (1986)
14. Farmer, W.M., Guttman, J.D., Thayer, F.J.: IMPS: An interactive mathematical proof system. Journal of Automated Reasoning 11(2), 213–248 (1993)
15. Farmer, W.M.: A partial functions version of Church's simple theory of types. Journal of Symbolic Logic 55(3), 1269–1291 (1990)