Strong normalization for System F by HOAS on top of FOAS

Andrei Popescu[1] Elsa L. Gunter[2] Christopher J. Osborn[3]
1,2,3: *University of Illinois at Urbana-Champaign.* 1: *Corresponding author. Email* uuomul@yahoo.com

***Abstract*—We present a point of view concerning HOAS (Higher-Order Abstract Syntax) and an extensive exercise in HOAS along this point of view. The point of view is that HOAS can be soundly and fruitfully regarded as a *definitional extension* on top of FOAS (First-Order Abstract Syntax). As such, HOAS is not only an *encoding technique*, but also a *higher-order view of a first-order reality*. A rich collection of concepts and proof principles is developed inside the standard mathematical universe to give technical life to this point of view. The exercise consists of a new proof of Strong Normalization for System F. HOAS makes our proof considerably more direct than previous proofs. The concepts and results presented here have been formalized in the theorem prover Isabelle/HOL.**

***Keywords-* Higher-Order Abstract Syntax; System F; General-Purpose Framework; Isabelle/HOL**

I. Introduction

HOAS (*Higher-Order Abstract Syntax*) is a methodology for representing formal systems (typically, logical systems or static or dynamic semantics of programming languages or calculi), referred to as *object systems*, into a fixed suitably chosen logic, referred to as the *meta logic*. HOAS prescribes that the object system be represented in the meta logic so that variable-binding, substitution and inference mechanisms of the former be captured by corresponding mechanisms of the latter. HOAS originated in [36], [52], [31], [49] and has ever since been extensively developed in frameworks with a wide variety of features and flavors. We distinguish two main (overlapping) directions in these developments.

-(I) First, the employment of a chosen meta logic as a *pure logical framework*, used for defining object systems for the purpose of reasoning *inside those systems*. A standard example is higher-order logic (HOL) as the meta logic and first-order logic (FOL) as the object system. Thanks to affinities between the mechanisms of these two logics, one obtains an *adequate encoding* of FOL in HOL by merely declaring in HOL types and constants and stating the FOL axioms and rules as HOL axioms – then the mechanisms for building FOL deductions (including substitution, instantiation, etc.) are already present in the meta logic, HOL.

-(II) Second, the employment of the meta-logic to reason *about* the represented object systems, i.e., to represent not only the object systems, but also (some of) their *meta-theory*. (E.g., cut elimination is a property *about* Gentzen-style FOL, not expressible in a standard HOAS-encoding of FOL into HOL.) While direction (I) has been quasi-saturated by the achievement of quasi-maximally convenient logical frameworks (such Edinburgh LF [31] and generic Isabelle [49]), this second direction undergoes these days a period of active research. We distinguish two main approaches here:

-(II.a) The *HOAS-tailored framework approach* [61], [59], [38], [40], [65], [25], [1], [16], [53]. This is characterized by the extension of the pure logical frameworks as in (I) with meta-reasoning capabilities. The diad (object system, meta logic) from (I) becomes a triad: *object system*, *logical framework* where this system is specified, *meta-logical framework* where one can reason *about* the logical framework [51]. The challenge here is choosing suitable logical and meta-logical frameworks that allow for adequate HOAS encodings, as well as enough expressive meta-theoretic power. (The logical framework is typically chosen to be a weak logic, e.g., an intuitionistic logic or type system as in (I), or linear logic.)

Somewhat complementary to the above work on HOAS-tailored *meta-reasoning*, [60], [20] developed HOAS-tailored *recursive definition* principles in a logical framework distinguishing between a parametric and a primitive-recursive function space.

-(II.b) The *general-purpose framework approach* [19], [18], [7]. This approach employs a general-purpose setting for developing mathematics, such as ZF Set Theory, Calculus of Constructions, or HOL with Infinity, as the logical framework, with object-level bindings captured again by means of meta-level bindings, here typically functional bindings – this means that terms with bindings from the object system are denoted using *standard functions*. Here there is no need for the three-level architecture as in (II.a), since the chosen logical framework is already strong enough, meta-theoretic expressiveness not being a problem. However, the difficulty here is brought be the meta-level function space being wider than desired, containing so-called "exotic terms". Even after the function spaces are cut down to valid terms, adequacy is harder to prove than at (II.a), precisely because of the logic's expressiveness.

We advocate here a variant of the approach (II.b), highlighting an important feature, to our knowledge not yet explored in the HOAS literature: the capability to *internalize, and eventually automate, both the representation map and the adequacy proof*. We illustrate this point by an example. Say we wish to represent and reason about λ-calculus and its associated β-reduction (as we actually do in this paper). Therefore, the object system is an "independent" (Platonic, if you will) mathematical notion, given by a collection of items called λ-terms, with operators on them, among which the syntactic constructs, free variables and substitution, and with an inductively defined reduction relation.

In the HOAS-tailored framework approach, for representing this system one defines a corresponding collection of constants in the considered logical framework, say LF, and then does an informal (but rigorous) *pen and paper proof* of the fact that the syntax representation is adequate (i.e., the existence of a compositional bijection between the λ-terms and normal forms of LF terms of appropriate types) and of a corresponding fact for β-reduction [50].

In the general-purpose framework approach, one can define the original system itself (here λ-calculus) in the meta-logical framework (say, HOL_ω, Higher-Order Logic with Infinity) *in such a way that accepting this definition as conforming to the mathematical definition is usually not a problem* (for one who already accepts that HOL_ω is adequate for representing the mathematical universe (or part of it)), since the former definitions are typically almost verbatim renderings of the latter – in HOL_ω, one can define inductively the datatype of terms, perhaps define α-equivalence and factor to it, then define substitution, reduction, etc. Moreover, one can also define in HOL_ω a system that is a HOAS-style representation of (the original) λ-calculus, i.e.: define a new type of items, call them HOAS-terms, with operators corresponding to the syntactic constructs of the original terms, but dealing with bindings via higher-order operators instead. In particular, the constructor for λ-abstraction will have type (HOAS-terms $\looparrowright$ HOAS-terms) $\to$ HOAS-terms, where one may choose the type constructor $\looparrowright$ to yield a restricted function space, or the whole function space accompanied by a predicate to cut down the "junk", etc. Once these constructions are done, one may also define in HOL_ω the syntax representation map from λ-terms to HOAS-terms and prove adequacy. (And a corresponding effort yields the representation of λ-term reduction.) Now, if the above are performed in a theorem prover that implements HOL_ω, such as Isabelle/HOL, then HOAS-terms become a *formally certified* adequate representation of the (original) λ-terms, not available in the existent HOAS-tailored approaches. Moreover, in many cases the construction of the HOAS-terms, the proofs of their basic properties and the adequacy proof can be *automated*, being the same for all syntaxes with bindings.

One may argue that, on the other hand, the above HOAS-terms do not retain all the convenience of a genuine HOAS encoding. Thus, when various standard relations need to be defined on HOAS-terms, certain context-free clauses specific to the HOAS-tailored frameworks (within the so-called *HOAS-encoding of judgements*) are no longer available here. E.g., a rule like $\dfrac{\forall X.\, X : S \Rightarrow A\, X : T}{\mathsf{Lam}(A) : S \to T}$ (typing rule for λ-abstractions – $\Rightarrow$ is logical implication and A a map from terms to terms) cannot act as a definitional clause in HOL_ω for a typing relation $\_ : \_$, due to non-monotonicity. The short answer to this objection is agreeing that general-purpose frameworks do bring their expressiveness with the price of not allowing the cleanest possible HOAS. A longer answer is given in this paper, where we argue that developing suitable general-purpose-framework concepts accommodates non-monotonicity and impredicativity flavors that make "pure" HOAS so attractive.

This paper develops novel HOAS concepts and techniques pertaining to the *general-purpose framework approach*. Here, the general-purpose framework could be regarded as being the mathematical universe (given axiomatically by any standard formalization of mathematics). All the involved systems, including the original systems and their representations, dwell in this mathematical universe, and are thus discussed and related via standard-mathematics concepts and theorems. Our HOAS exercise here is a proof of the strong normalization result for System F [27].

Apart from this introduction, Sec. II recalling some syntax concepts, Sec. VI drawing conclusions and discussing related and future work, and the appendix giving more details on our Isabelle formalization, the paper has two main parts. In the first part, consisting of Secs. III and IV, we discuss some general HOAS techniques for representing syntax and inductively defined relations, illustrated on the λ-calculus and System F. The HOAS "representation" of the original first-order syntax will not be a representation in the usual sense (via defining a new (higher-order) syntax), but will take *a different view of the same syntax*. Let us call *abstractions* pairs (x, X) variable-term modulo α-equivalence.[1] Abstractions are therefore the arguments to which the λ-operator applies, as in $\lambda x.X$.[2] Under the higher-order view, abstractions A are no longer constructed by variable-term representatives, but are analyzed/"destructed" by applying them (as functions), via substitution, to terms. Namely, given a term X, $A \,\_\, X$, read "A applied to X", is defined to be $Y[X/x]$, where (x, Y) is any variable-term representative for A. This way, the space of abstractions becomes essentially a restricted function space from terms to terms, as in strong HOAS. (The [strong HOAS]–[weak HOAS] dichotomy is recalled in Sec. VI-C.) Although this change of view is as banal as possible, it meets its purpose: the role previously played by substitution now belongs to function-like application. The latter of course originates in substitution, but one can forget about its origin. In fact, one can (although is not required to!) also forget about the original first-order binding constructor and handle terms entirely by means of the new, higher-order destructor. Moving on to the discussion of recursive-definition principles for syntax, we perform an analysis of various candidates for the type of the recursive combinator, resulting notably in a novel "impredicative" principle in the spirit of (strong) HOAS.

Then we discuss HOAS representation of inductively defined relations, performed by a form of transliteration following some general patterns. These patterns are illustrated

[1]In this paper, we use lowercases for variables and uppercases for terms.

[2]In order to streamline our presentation, we prefer to regard an operator such as λ not as binding itself variables into its arguments, but rather as taking its arguments with their variables already bound, i.e., taking abstractions as arguments – see Sec. II.

by the case of the reduction and typing relation for System F, and it appears that a large class of systems (e.g., most of those from the monographs [10], [29], [42], [54]) can be handled along these lines. For typing, we also present a "purely HOAS" induction principle, not mentioning typing contexts. Once our formalization will be fully automated (see Sec. VI-B), it will have a salient advantage over previous HOAS approaches: adequacy will need *not* be proved by hand, but will follow automatically from general principles.

In the second part, Sec. V, we sketch a proof of strong normalization for System F within our HOAS framework. We make essential use of our aforementioned *definitional principle* and *typing-context-free induction principle* to obtain a general criterion for proving properties on typable terms (which is in principle applicable to properties other than strong normalization, including confluence and type preservation). From this criterion, we infer strong normalization for terms typable in the empty context. (Extending the result to arbitrary contexts is then immediate.) Unlike previous proofs [28], [64], [43], [26], [11], [6], [13], [37], [21], our proof does not employ *data or type environments* and *semantic interpretation of typing contexts* – a virtue of our setting, which is thus delivering the HOAS-prescribed service of *clearing the picture of inessential details*.

Isabelle formalization. For the formalization of the concepts and results presented in this paper (including the FOAS definitions of the systems, their HOAS representations and adequacy theorems, and the Strong Normalization theorem), we have chosen a particular general-purpose logic, namely HOL$_\omega$, implemented as Isabelle/HOL [48]. The formal scripts can be downloaded from [56]. The document SysF.pdf from that (zipped) folder contains a detailed presentation of the relevant theories. These theories can also be browsed in html format in the folder SysF_Browse. The section-wise structure of this paper reflects quite faithfully that of our Isabelle development, so that the reader should have no difficulty mapping one into the other. Moreover, the concrete syntax we use for our operators in Isabelle is almost identical to the one of the paper; the proofs, written (for the more complex facts) in the top-down Isar [47] style, are also fairly readable. (More details can be found in the appendix.) The above precautions allow us to focus our presentation on mathematics rather than on formalization, while still keeping an eye on formalization. As a side-effect, we hope to illustrate that the discussed "general mathematics" is formalizable in other general-purpose theorem provers besides that of our choice. (Though, admittedly, some extra care is required, if working in more constructive settings such as Coq [14], in order to have our definitions go through.)

Conventions and notations. While Isabelle distinguishes between types (as primitive items) and sets (as items inhabiting **bool**-functional types), we shall ignore this distinction here and refer to all the involved collections as *sets* (the reader can recognize the types though by the boldface fonts). We employ the lambda-abstraction, universal/existential quantification and implication symbols λ, $\forall$, $\exists$ and $\Rightarrow$ only in the meta-language of this paper, and *not* in the formal languages that we discuss. $A \to B$ is the A-to-B function space, and $\mathbf{P}(A)$ and $\mathbf{P}_{\neq\emptyset}(A)$ the powerset and $\mathbf{P}(A) \setminus \{\emptyset\}$, respectively. $\circ$ is functional composition. For $R \subseteq A \times A$, R^* is its reflexive-transitive closure. [] is the empty list and infixed "," is list concatenation.

II. The λ-calculus and System F recalled

The two systems are standardly defined employing First-Order Abstract Syntax (FOAS), modulo α-equivalence. We later refer to them as "the original systems", to contrast them with their HOAS representations.

A. The (untyped) λ-calculus

We fix an infinite set **var**, of *variables*, ranged over by x, y, z. The sets **term**, of *terms*, ranged over by X, Y, Z, and **abs**, of *abstractions*, ranged over by A, B, are given by: $X ::= \mathsf{InV}\ x \mid \mathsf{App}\ X\ Y \mid \mathsf{Lam}\ A \qquad A ::= x.X$ where we assume that, in $x.X$, x is bound in X, and *terms and abstractions are identified modulo the standardly induced notion of α-equivalence* (see, e.g., [12]). Therefore what we call "abstractions" and "terms" in this paper are *α-equivalence classes*. (Note: the operators App, Lam and $x.\_$ (for any fixed x) are well-defined on α-equivalence classes.) For convenience, we shall keep implicit the injective map InV : **var** $\to$ **term**, and pretend that **var** $\subseteq$ **term** (this omission will be performed directly for the syntax of System F below). An *environment* $\rho \in$ **env** is a finite-domain partial function from variables to terms. We write:
- fresh : **var** $\to$ **term** $\to$ **bool**, for the predicate indicating if a variable is fresh in a term ("fresh" meaning "non-free");
- $\_[\_]$: **term** $\to$ **env** $\to$ **term**, for the concurrent substitution on terms – namely, $X[\rho]$ is the term obtained from X by concurrently (and capture-avoiding-ly) substituting in X each variable x with the term $\rho(x)$ if $\rho(x)$ is defined.
- $\_[\_/\_]$: **term** $\to$ **term** $\to$ **var** $\to$ **term**, for unary substitution – namely, $X[Y/y]$ is the term obtained from X by (capture-avoiding-ly) substituting y with Y in X.

We employ the same notations for abstractions: fresh : **var** $\to$ **abs** $\to$ **bool**, $\_[\_]$: **abs** $\to$ **env** $\to$ **abs**, etc.

One-step β-reduction $\leadsto$: **term** $\to$ **term** $\to$ **bool** is given by:

$$\frac{\cdot}{\mathsf{App}\ (\mathsf{Lam}(x.Y))\ X \leadsto Y[X/x]}\ \textbf{(Beta)} \qquad \frac{X \leadsto Y}{\mathsf{Lam}(z.X) \leadsto \mathsf{Lam}(z.Y)}\ \textbf{(Xi)}$$

$$\frac{X \leadsto Y}{\mathsf{App}\ X\ Z \leadsto \mathsf{App}\ Y\ Z}\ \textbf{(AppL)} \qquad \frac{X \leadsto Y}{\mathsf{App}\ Z\ X \leadsto \mathsf{App}\ Z\ Y}\ \textbf{(AppR)}$$

X is called *strongly normalizing* if there is no infinite sequence $(X_n)_{n\in\mathbb{N}}$ with $X_0 = X$ and $\forall n.\ X_n \leadsto X_{n+1}$.

B. System F

We describe this system as a typing system for λ-terms without type annotations, in a Curry style (see [10]). Its syntax consists of two copies of the untyped λ-calculus syntax – one for data and one for types. More precisely, we fix two infinite sets, **dvar**, of *data variables* (*dvars* for short), ranged over by x, y, z, and **tvar**, of *type variables* (*tvars* for short), ranged over by *tx*, *ty*, *tz*. The sets **dterm** and **dabs**, of *data terms* and *abstractions* (*dterms* and *dabstractions* for short), ranged over by X, Y, Z and A, B, C, and **tterm** and **tabs**, of *type terms* and *abstractions* (*tterms* and *tabstractions* for short), ranged over by tX, tY, tZ and tA, tB, tC, are defined by the following grammars, again up to α-equivalence:

$$X \ ::= \ x \mid \mathsf{App}\ X\ Y \mid \mathsf{Lam}\ A \qquad A \ ::= \ x.X$$
$$tX \ ::= \ tx \mid \mathsf{Arr}\ tX\ tY \mid \mathsf{Al}\ tA \qquad tA \ ::= \ tx\,.\,tX$$

Above, App and Lam stand, as in Subsec II-A, for "application" and "lambda", while Arr and Al stand for "arrow" and the "for all" quantifier. Since dterms do not have type annotations, indeed both the abstract syntax of dterms and that of tterms are that of λ-calculus (from Subsec. II-A), just that for tterms we write Arr and Al instead of App and Lam.

All concepts and results from Subsec. II-A apply to either syntactic category, separately. Let **denv**, ranged over by ρ, be the set of data environments, and **tenv**, ranged over by ξ, that of type environments. For any items a and b, we may write $a : b$ for the pair (a, b). A *well-formed typing context* (*context* for short) $\Gamma \in$ **ctxt** is a list of pairs dvar-tterm, $x_1 : tX_1, \ldots, x_n : tX_n$, with the x_i's distinct. The homonymous predicates fresh : **dvar** $\rightarrow$ **ctxt** $\rightarrow$ **bool** and fresh : **tvar** $\rightarrow$ **ctxt** $\rightarrow$ **bool** (indicating if a dvar or a tvar is fresh for a context) are defined as expected: $\mathsf{fresh}\, y\, [] = \mathsf{True}$; $\mathsf{fresh}\, y\, (\Gamma, (x : tX)) = (\mathsf{fresh}\, y\, \Gamma \wedge y \neq x)$; $\mathsf{fresh}\ ty\ [] = \mathsf{True}$; $\mathsf{fresh}\ ty\ (\Gamma, (x : tX)) = (\mathsf{fresh}\ ty\ \Gamma \wedge \mathsf{fresh}\ ty\ tX)$.

The type inference relation $(\_ \vdash \_ : \_)$: **ctxt** $\rightarrow$ **dterm** $\rightarrow$ **tterm** $\rightarrow$ **bool** is defined inductively by the clauses:

$$\frac{\cdot}{\Gamma, x : tX \vdash x : tX}\ \textbf{(Asm)}\ [\mathsf{fresh}\ x\ \Gamma] \qquad \frac{\Gamma \vdash X : tX}{\Gamma, y : tY \vdash X : tX}\ \textbf{(Weak)}\ [\mathsf{fresh}\ y\ \Gamma]$$

$$\frac{\Gamma, x : tX \vdash Y : tY}{\Gamma \vdash \mathsf{Lam}(x.Y) : \mathsf{Arr}\ tX\ tY}\ \textbf{(ArrI)}\ [\mathsf{fresh}\ x\ \Gamma] \qquad \frac{\Gamma \vdash Y : tY}{\Gamma \vdash Y : \mathsf{Al}(tx.tY)}\ \textbf{(AlI)}\ [\mathsf{fresh}\ tx\ \Gamma]$$

$$\frac{\Gamma \vdash X : \mathsf{Arr}\ tY\ tZ \qquad \Gamma \vdash Y : tY}{\Gamma \vdash \mathsf{App}\ X\ Y : tZ}\ \textbf{(ArrE)} \qquad \frac{\Gamma \vdash Y : \mathsf{Al}(tx.tY)}{\Gamma \vdash Y : tY[tX/tx]}\ \textbf{(AlE)}$$

We write $\vdash X : tX$ for $[] \vdash X : tX$. X is called *typable* if $\Gamma \vdash X : tX$ for some Γ and tX.

III. HOAS VIEW OF SYNTAX

Here we present a HOAS approach to the *syntax* of calculi with bindings. We describe our approach for the paradigmatic particular case of the untyped λ-calculus (from Sec. II-A), but our discussion is easily generalizable to terms generated from any (possibly many-sorted) binding signature (as defined, e.g., in [23], [63]). We do *not* define a new higher-order syntax, but introduce higher-order operators on the original syntax – hence we speak of a *HOAS view* rather than of a *HOAS representation*.

A. Abstractions as functions

Throughout the rest of this section, we use the concepts and notations from Sec. II-A, and *not* the ones from Sec. II-B. Given $A \in$ **abs** and $X \in$ **term**, the *functional application of A to X*, written $A\,\_\,X$, is defined to be $Y[X/x]$ for any x and Y s.t. $A = (x.Y)$. (The choice of (x, Y) is easily seen to be immaterial.) The operator $\_$ is extensional, qualifying the set of abstractions as a *restricted term-to-term function space*, and preserves freshness. Thus, abstractions are no longer regarded as pairs var-term up to α-equivalence, but as functions, in the style of HOAS. Under this higher-order view, abstractions can be destructed by application, as opposed to constructed by means of var-term representatives as in the original first-order view. But does the higher-order view suffice for the specification of relevant systems with bindings? I.e., can we do without "constructing" abstractions? Our answer is threefold:

-(1) Since the higher-order view does not change the first-order syntax, abstractions by representatives are still available if needed.

-(2) Many relevant systems with bindings employ the binding constructors within a particular style of interaction with substitution and scope extrusion (e.g., all variables appear either bound, or substituted, or [free in the hypothesis]) which makes the choice of binding representatives irrelevant. This phenomenon, to our knowledge not yet rigorously studied mathematically for a general syntax with bindings, is really the basis of most HOAS representations from the literature. In Sec. IV, we elaborate informally on what this phenomenon becomes in our setting.

-(3) The previous point argued that relevant systems *specifications* can do without constructing abstractions. Now, w.r.t. *proofs* of meta-theoretic properties, one may occasionally need to perform case-analysis and induction *on abstractions*. HOAS-style case-analysis and induction are discussed below, after we introduce 2-abstractions.

B. 2-abstractions

These are for abstractions what abstractions are for terms. 2-abstractions $\mathcal{A} \in$ **abs2** are defined as pairs $x.A$ var-abstraction up to α-equivalence (just like abstractions are pairs var-term up to α). (Alternatively, they can be regarded as triples $x.y.Z$, with $x, y \in$ **var** and $Z \in$ **term**, again up to α.) Next we define two application operators for 2-abstractions. If $\mathcal{A} \in$ **abs**2 and $X \in$ **term**, then $\mathcal{A}\,\_1\,X$ and $\mathcal{A}\,\_2\,X$ are the following elements of **abs**:

- $\mathcal{A}\,\_1\,X = A[X/x]$, where x, A are s.t. $\mathcal{A} = (x.A)$;
- $\mathcal{A}\,\_2\,X = (y.(Z[X/x]))$, where y, Z are s.t. $y \neq x$, $\mathsf{fresh}\ y\ X$ and $\mathcal{A} = (y.(x.Z))$.

(Again, the choice of representatives is immaterial.) Thus, essentially, 2-abstractions are regarded as 2-argument functions and applied correspondingly.

Now we can define homonymous syntactic operations for abstractions lifting those for terms:

- InV : **var** $\rightarrow$ **abs**, by $\mathsf{InV}\ x = (y.x)$, where y is s.t. $y \neq x$;
- App : **abs** $\rightarrow$ **abs** $\rightarrow$ **abs**, by $\mathsf{App}\ A\ B = (z.\,(\mathsf{App}\ X\ Y))$, where z, X, Y are s.t. $A = (z.X)$ and $B = (z.Y)$.
- Lam : **abs**2 $\rightarrow$ **abs**, by $\mathsf{Lam}\ \mathcal{A} = (x.\,(\mathsf{Lam}\ A))$, where x, A are s.t. $\mathcal{A} = (x.A)$.

If we also define $\mathsf{id} \in \mathsf{Abs}$ to be $(x.x)$ for some x, we can case-analyze abstractions by the above four (complete and non-overlapping) constructors. Moreover, functional application verifies the expected exchange law $(\mathcal{A}\_1\,X)\_Y = (\mathcal{A}\_2\,Y)\_X$ and commutes with abstraction versus terms constructors, e.g., $(\mathsf{Lam}\ \mathcal{A})\_X = \mathsf{Lam}(\mathcal{A}\_1\,X)$.

C. Induction principles for syntax

The following is the natural principle for terms under the HOAS view. Notice that it requires the use of abstractions.

Prop 1: Let $\varphi : \textbf{term} \to \textbf{bool}$ be s.t. the following hold:
(i) $\forall x.\ \varphi\ x$. **(ii)** $\forall X, Y.\ \varphi\ X \wedge \varphi\ Y \Rightarrow \varphi(\mathsf{App}\ X\ Y)$.
(iii) $\forall A.\ (\forall x. \varphi(A\_x)) \Rightarrow \varphi(\mathsf{Lam}\ A)$. Then $\forall X.\ \varphi\ X$.

Likewise, a HOAS induction principle for abstractions requires the use of 2-abstractions. The 2-place application in the inductive hypothesis for Lam in Prop. 2 offers "permutative" flexibility for when reasoning about multiple bindings – the proof of Prop. 13 from Sec. V illustrates this.

Prop 2: Let $\varphi : \textbf{abs} \to \textbf{bool}$ be s.t. the following hold:
(i) $\varphi\,\mathsf{id}$. **(ii)** $\forall x.\ \varphi(\mathsf{InV}\,x)$.
(iii) $\forall A, B.\ \varphi\,A \wedge \varphi\,B \Rightarrow \varphi(\mathsf{App}\,A\,B)$.
(iv) $\forall \mathcal{A}.\ (\forall x.\ \varphi(\mathcal{A}\_1\,x) \wedge \varphi(\mathcal{A}\_2\,x)) \Rightarrow \varphi(\mathsf{Lam}\ \mathcal{A})$.
Then $\forall A.\ \varphi\ A$.

D. Recursive definition principles for syntax

This is known as a delicate matter in HOAS. One would like that, given any set $\mathcal{C}$, a map $H : \textbf{term} \to \mathcal{C}$ be determined by a choice of the operations $\mathsf{cInV} : \textbf{var} \to \mathcal{C}$, $\mathsf{cApp} : \mathcal{C} \to \mathcal{C} \to \mathcal{C}$, and cLam (whose type we do not yet specify) via the conditions:
(I) $H\,x = \mathsf{cInV}\ x$. **(II)** $H(\mathsf{App}\,X\,Y) = \mathsf{cApp}\,(H\ X)\,(H\,Y)$.
(III) An equation (depending on the type of cLam) with $H(\mathsf{Lam}\ A)$ on the left.
(We only discuss *iteration*, and not general recursion.)

Candidates for the type of the operator cLam are:
(1) $\mathsf{cLam} : (\textbf{term} \to \mathcal{C}) \to \mathcal{C}$, suggesting the equation $H(\mathsf{Lam}\ A) = \mathsf{cLam}(\lambda X.\,H(A\_X))$ – this is problematic as a definitional clause, due to its impredicativity;
(2) A weak-HOAS-like [18] variable-restriction of (1), namely, $\mathsf{cLam} : (\textbf{var} \to \mathcal{C}) \to \mathcal{C}$, yielding the equation
(III$_w$): $H(\mathsf{Lam}\ A) = \mathsf{cLam}(\lambda x.\,H(A\_x))$
and a recursive principle:

Prop 3: There exists a unique map $H : \textit{Term} \to \mathcal{C}$ s.t. equations (I), (II), and (III$_w$) hold.
(3) $\mathsf{cLam} : (\mathcal{C} \to \mathcal{C}) \to \mathcal{C}$. Then there is no apparent way of defining the equation (III) in terms of Lam and cLam without parameterizing by valuations/environments in $\textbf{var} \to \mathcal{C}$, and thus getting into first-order "details" (at least not in a standard setting such as ours – but see [60], [20] for an elegant solution within a modal typed λ-calculus).
(4) A "flattened" version (collapsing some type information) of both (1) and (3), namely, $\mathsf{cLam} : \mathbf{P}_{\neq\emptyset}(\mathcal{C}) \to \mathcal{C}$. This may be regarded as obtained by requiring the operator from (1) or (3) to depend only on the image of its arguments in $\textbf{term} \to \mathcal{C}$ or $\mathcal{C} \to \mathcal{C}$, respectively. The natural associated (valuation-independent) condition (III) would be $H(\mathsf{Lam}\ A) = \mathsf{cLam}(\{H(A\_X).\ X \in \textbf{term}\})$.

Unfortunately, this condition is still too strong to guarantee the existence of H. But interestingly, if we have enough variables, the existence of a compositional map holds:

Prop 4: Assume $\mathsf{card}(\textbf{var}) \geq \mathsf{card}(\mathcal{C})$ and let $\mathsf{cApp} : \mathcal{C} \to \mathcal{C} \to \mathcal{C}$ and $\mathsf{cLam} : \mathbf{P}_{\neq\emptyset}(\mathcal{C}) \to \mathcal{C}$ (where card is the cardinal operator). Then there exists $H : \textbf{term} \to \mathcal{C}$ s.t.:
(I) $H(\mathsf{App}\ X\ Y) = \mathsf{cApp}\ (H\ X)\ (H\ Y)$ for all X, Y.
(II) $H(\mathsf{Lam}\,A) = \mathsf{cLam}(\{H(A\_X).\ X \in \textbf{term}\})$ for all A.

Prop. 4 is looser than a definition principle, since it does not state uniqueness of H. In effect, it is a "loose definition" principle, which makes no commitment to the choice of interpreting the variables. (Though it can be proved that H is uniquely determined by its action on variables. As a trivial example, the identity function on terms is uniquely identified by its action on variables and by equations (I) and (II). Other functions, such as term-depth, do *not* fall into the cardinality hypothesis of this proposition, but of course can be defined using Prop. 3.) Note the "impredicative" nature of equation (II): it "defines" H on $\mathsf{Lam}\,A$ in terms of the "HOAS-components" of A, where a "HOAS component" is a result of applying A (as a function) to a term X and can of course be larger than A. This proposition can be useful in situations where the existence of a compositional map is the only relevant aspect, allowing to take a shortcut from the first-order route of achieving compositionality through interpretation in environments – our proof of Strong Normalization from Sec. V takes advantage of this.

Conclusion: While the above preparations for HOAS on top of FOAS do require some work, this work is uniformly applicable to any (statically-scoped) syntax with bindings, hence automatable. Moreover, once this definitional effort is finished, one can forget about the definitions and work entirely in the comfortable HOAS setting (meaning: no more α-representatives, variable capture, etc.), as illustrated next.

IV. HOAS REPRESENTATION OF INFERENCE

This section deals with the HOAS representation of inductively defined relations on syntax, such as typing and reduction. Given an inductively defined relation on the first-order syntax employing the first-order operators, we *transliterate* it through our HOAS view, roughly as follows:
(I) abstractions constructed by terms with explicit dependencies become "plain" abstractions (used as functions);
(II) terms with implicit dependencies become abstractions applied to the parameter they depend on;
(III) substitution becomes functional application;
(IV) unbound arbitrary variables become arbitrary terms;
(V) scope extrusion is handled by universal quantification.
(We explain and illustrate these as we go through the examples, where the informal notions of implicit and explicit dependency will also be clarified.)

Our presentation focuses on a particular example, the typing and reduction of System F, but the reader can notice that the approach is rather general, covering a large class of reduction and type systems.

At this point, the reader should recall the definitions and notations pertaining to System F from Sec. II-B. Notations, in a nutshell: lowercases x, y, z for dvars, uppercases X, Y, Z for dterms, uppercases A, B for dabstractions, calligraphic uppercases $\mathcal{A}, \mathcal{B}$ for 2-dabstractions; for the type versions of the above, we prefix everything by "t". All the discussion from Sec. III duplicates for the two copies of the λ-calculus that make the syntax of System F. In particular, we have data-abstraction-lifted operators $\mathsf{App} : \mathbf{dabs} \to \mathbf{dabs} \to \mathbf{dabs}$, $\mathsf{Lam} : \mathbf{dabs2} \to \mathbf{dabs}$, etc. (where $\mathbf{dabs2}$ is the set of data 2-abstractions).

A. Representation of reduction

We define $\rightsquigarrow\!\!\!\!\!\twoheadrightarrow : \mathbf{dterm} \to \mathbf{dterm} \to \mathbf{bool}$ inductively:

$$\frac{\cdot}{\mathsf{App}\ (\mathsf{Lam}\ A)\ X \rightsquigarrow\!\!\!\!\!\twoheadrightarrow A\,\_\,X}\ \textbf{(HBeta)} \qquad \frac{\forall Z.\ A\,\_\,Z \rightsquigarrow\!\!\!\!\!\twoheadrightarrow B\,\_\,Z}{\mathsf{Lam}\ A \rightsquigarrow\!\!\!\!\!\twoheadrightarrow \mathsf{Lam}\ B}\ \textbf{(HXi)}$$

$$\frac{X \rightsquigarrow\!\!\!\!\!\twoheadrightarrow Y}{\mathsf{App}\ X\ Z \rightsquigarrow\!\!\!\!\!\twoheadrightarrow \mathsf{App}\ Y\ Z}\ \textbf{(HAppL)} \qquad \frac{X \rightsquigarrow\!\!\!\!\!\twoheadrightarrow Y}{\mathsf{App}\ Z\ X \rightsquigarrow\!\!\!\!\!\twoheadrightarrow \mathsf{App}\ Z\ Y}\ \textbf{(HAppR)}$$

Adequacy of the reduction representation is contained in:

Prop 5: The following are equivalent:
(1) $X \rightsquigarrow Y$. **(2)** $X \rightsquigarrow\!\!\!\!\!\twoheadrightarrow Y$. **(3)** $\forall \rho \in \mathbf{denv}.\ X[\rho] \rightsquigarrow\!\!\!\!\!\twoheadrightarrow Y[\rho]$.

Remember that our HOAS representation dwells in the same universe as the original system, i.e., both the original relation $\rightsquigarrow$ and the representation relation $\rightsquigarrow\!\!\!\!\!\twoheadrightarrow$ act on the same syntax – they only differ *intensionally* in the way their definition manipulates this syntax: the former through bindings and substitution, the latter through abstractions-as-functions and function application. Looking for the incarnations of the general HOAS-transliteration patterns (I)-(V) listed at the beginning of this section, we find that:
- The definition of $\rightsquigarrow\!\!\!\!\!\twoheadrightarrow$ is obtained by modifying in $\rightsquigarrow$ *only the clauses involving binding and substitution*: (Beta), (Xi);
- In (Beta) and (Xi), $\mathsf{Lam}(x.Y)$, $\mathsf{Lam}(z.X)$ and $\mathsf{Lam}(z.Y)$ become $\mathsf{Lam}\ A$, $\mathsf{Lam}\ A$ and $\mathsf{Lam}\ B$, according to (I);
- In (Beta), $Y[X/x]$ becomes $A\,\_\,X$, according to (III);
- In (Xi), *regarded as applied backwards*, we have the extrusion of the scope of z, as z is bound in the conclusion and free in the hypothesis – by pattern (V), this brings universal quantification over an arbitrary term Z in the hypothesis, as well as the acknowledgement of an implicit dependency on z (now having become Z) in the X and Y from the hypothesis, making them become, by (II), abstractions applied to the implicit parameter, $A\,\_\,Z$ and $B\,\_\,Z$.
(Note that this example does not illustrate pattern (IV), since all variables appearing in the definition of $\rightsquigarrow$ are bound.)[3]

The infinitary clause (HXi) from the definition of $\rightsquigarrow\!\!\!\!\!\twoheadrightarrow$ (whose premise quantifies over all dterms Z) is convenient when proving that $\rightsquigarrow\!\!\!\!\!\twoheadrightarrow$ *is included in* another relation, as it makes a very strong induction hypothesis, much stronger than that given by (Xi) for $\rightsquigarrow$. This is also true for rule inversion, where from $\mathsf{Lam}\ A \rightsquigarrow\!\!\!\!\!\twoheadrightarrow \mathsf{Lam}\ B$ we can infer a good deal of information compared to the first-order case. However, when proving that $\rightsquigarrow\!\!\!\!\!\twoheadrightarrow$ *includes* a certain relation, it appears that a HOAS clause matching (Xi) more closely may help. Such a clause can be extracted from (Xi):

Prop 6: $\rightsquigarrow\!\!\!\!\!\twoheadrightarrow$ is closed under the following rule:

$$\frac{\mathsf{fresh}\ z\ A \quad \mathsf{fresh}\ z\ B \quad A\ z \rightsquigarrow\!\!\!\!\!\twoheadrightarrow B\ z}{\mathsf{Lam}\ A \rightsquigarrow\!\!\!\!\!\twoheadrightarrow \mathsf{Lam}\ B}\ \textbf{(HXi')}$$

Note that (HXi') is stronger than (HXi) (but stronger as a rule means weaker as an induction-principle clause). A rule such as (HXi') should be viewed as a facility to descend, if necessary, from the HOAS altitude into "some details" (here, a freshness side-condition). This fits into our goal of encouraging HOAS definitions and proofs, while also allowing access to details on a by-need basis.

Since, by Prop. 5, the relations $\rightsquigarrow$ and $\rightsquigarrow\!\!\!\!\!\twoheadrightarrow$ coincide, hereafter we shall use only the symbol "$\rightsquigarrow\!\!\!\!\!\twoheadrightarrow$".

B. Representation of inference

A *HOAS context* (*Hcontext* for short) $\Delta \in \mathbf{Hctxt}$ is a list of pairs in $\mathbf{dterm} \times \mathbf{tterm}$, $X_1 : tX_1, \ldots, X_n : tX_n$. Note that $\mathbf{ctxt} \subseteq \mathbf{Hctxt}$. For Hcontexts, freshness, $\mathsf{fresh} : \mathbf{dvar} \to \mathbf{Hctxt} \to \mathbf{bool}$ and $\mathsf{fresh} : \mathbf{tvar} \to \mathbf{Hctxt} \to \mathbf{bool}$, and substitution, $\_\,[\_\,,\_\,] : \mathbf{Hctxt} \to \mathbf{tenv} \to \mathbf{denv} \to \mathbf{Hctxt}$ are defined as expected: $\mathsf{fresh}\, y\, [] = \mathsf{True}$; $\mathsf{fresh}\, y\, (\Delta, (X : tX)) = (\mathsf{fresh}\ y\ \Delta \wedge \mathsf{fresh}\ y\ X)$; $\mathsf{fresh}\ ty\ [] = \mathsf{True}$; $\mathsf{fresh}\ ty\ (\Delta, (x : tX)) = (\mathsf{fresh}\ ty\ \Delta \wedge \mathsf{fresh}\ ty\ tX)$; $[]\,[\xi, \rho] = []$; $(\Delta, (X : tX))\,[\xi, \rho] = (\Delta[\xi, \rho],\ (X[\rho] : tX[\xi]))$.

We represent type inference by the relation $(\_ \Vdash \_ : \_) : \mathbf{Hctxt} \to \mathbf{dterm} \to \mathbf{tterm} \to \mathbf{bool}$, called *HOAS typing* (*Htyping* for short):

$$\frac{\cdot}{\Delta, X : tX \Vdash X : tX}\ \textbf{(HAsm)} \qquad \frac{\Delta \Vdash X : tX}{\Delta, Y : tY \Vdash X : tX}\ \textbf{(HWeak)}$$

$$\frac{\forall X.\ \Delta, X : tX \Vdash A\,\_\,X : tY}{\Delta \Vdash \mathsf{Lam}\ A : \mathsf{Arr}\ tX\ tY}\ \textbf{(HArrI)} \qquad \frac{\forall tX.\ \Delta \Vdash Y : tA\,\_\,tX}{\Delta \Vdash Y : \mathsf{Al}\ tA}\ \textbf{(HAlI)}$$

$$\frac{\Delta \Vdash X : \mathsf{Arr}\ tY\ tZ \quad \Delta \Vdash Y : tY}{\Delta \Vdash \mathsf{App}\ X\ Y : tZ}\ \textbf{(HArrE)} \qquad \frac{\Delta \Vdash Y : \mathsf{Al}\ tA}{\Delta \Vdash Y : tA\,\_\,tX}\ \textbf{(HAlE)}$$

Prop 7: (Adequacy) The following are equivalent:
(1) $\Gamma \vdash X : A$.
(2) $\Gamma \Vdash X : A$. (Note: contexts are particular Hcontexts.)
(3) $\Gamma[\xi, \rho] \Vdash X[\rho] : A[\xi]$ for all $\xi \in \mathbf{tenv}$ and $\rho \in \mathbf{denv}$.

It follows that $\Vdash$ is a *conservative extension* (from contexts to Hcontexts) of $\vdash$. Thus, unlike with reduction, our HOAS representation of typing, $\Vdash$, does *not* manipulate the same items as the original relation $\vdash$, but *extends* the domain – essentially, the new domain is the closure of the original domain under substitution. Hereafter we write $\Vdash$ for either relation, but still have Γ range over $\mathbf{ctxt}$ and Δ over $\mathbf{Hctxt}$.

The only pattern from (I)-(V) exhibited by our HOAS-transliteration of typing that is not already present in the one for reduction is (IV), shown in the transliterations of (Asm), (Weak) and (ArrI) – there, we have the variables x and y becoming terms X and Y in (HAsm) (HWeak) and (HArrI). At (ArrI), (IV) is used in combination with (V), because x is also extruded back from the conclusion to the hypothesis,

[3]What we discuss here, in the context of the aforementioned patterns, are not the inductively defined relations, but the inductive definitions themselves; and what we loosely refer to as "variables" and "terms" appearing in these definitions are really variable and term meta-variables.

thus becoming in the hypothesis of (HArrI) a universally quantified term X. Another phenomenon not exhibited by reduction is the presence of freshness side-conditions (in the original system), whose effect is to *prevent dependencies* – e.g., the side-condition $\mathsf{fresh}\ y\ \Gamma$ from (Weak) says that Γ does not depend on x, meaning that, when transliterating (Weak) into (HWeak), (II) is not applicable to Γ. (Otherwise, to represent this we would need Hcontext-abstractions!)

Note that $\leadsto\!\!\!\twoheadrightarrow$ and $\rightsquigarrow$ coincide, while $\Vdash$ is only a conservative extension of $\vdash$ – this is because our HOAS transliteration method *always closes under substitution*, and $\rightsquigarrow$, unlike $\vdash$, is already closed. The presence of unbound variables in the first-order definition, requiring modification (IV), is a precise indicator of non-closedness.

C. Induction principle for type inference

By definition, $\Vdash$ offers an induction principle: If a relation R : **Hctxt** $\rightarrow$ **dterm** $\rightarrow$ **tterm** $\rightarrow$ **bool** is closed under the rules defining $\Vdash$, then $\forall \Delta, X, tX.\ \Delta \Vdash X : tX \Rightarrow R\ \Delta\ X\ tX$.

A HOAS technique should ideally do away (whenever possible) not only with the explicit reference to bound variables and substitution, but with the explicit reference to inference (judgment) contexts as well. Our inductive definition of Htyping achieves the former, but not the latter. Now, trying to naively eliminate contexts in a "truly HOAS" fashion, replacing, e.g., the rule (HArrI) with something like:

$$\frac{\forall X.\ \textit{typeOf}\ X\ tX \Rightarrow \textit{typeOf}\ (A \_ X)\ tY}{\textit{typeOf}\ (\mathsf{Lam}\ A)\ (\mathsf{Arr}\ tX\ tY)} \quad (*)$$

in an attempt to define *non-hypothetic typing* (i.e., typing in the empty context) directly as a binary relation *typeOf* between dterms and tterms, we hit two well-known problems:
-(I) The contravariant position of $\textit{typeOf}(X, tX)$ prevents the clause (*) from participating at a valid inductive definition.
-(II) Even if we "compromise" for a non-definitional (i.e., axiomatic) approach, but would like to retain the advantages of working in a standard logic, then (*) is likely to *not be sound*, i.e., not capture correctly the behavior of the original system. Indeed, in a classical logic it would allow one to type any $\mathsf{Lam}\,A$ to a type $\mathsf{Arr}\ tX\ tY$ for some non-inhabited type tX. Moreover, even we restrict ourselves to an intuitionistic setting, we still need to be very careful with (and, to some extent, make compromises on) the foundations of the logic in order for axioms like (*) to be sound. This is because, while the behavior of the intuitionistic connectives accommodates such axioms adequately, other mechanisms pertaining to recursive definitions are not a priori guaranteed to preserve adequacy – see [33], [39].

So what can one make of a clause such as (*) in a framework with meta-reasoning capabilities? As already discussed in the introduction, the HOAS-tailored framework's solution is axiomatic: (*) would be an axiom in a logic L (hosting the representation of the object system), with L itself is viewed as an object by the meta-logic; in the meta-logic then, one can perform proofs by induction on derivations in L. Thus, HOAS-tailored frameworks solve the problems with (*) by stepping one level up to a meta-logic. Previous work in general-purpose frameworks, after several experiments, eventually proposed similar solutions, either of directly interfering with the framework axiomatically [45] or of employing the mentioned intermediate logic L [44].

Our own solution has an entirely different flavor, and does not involve traveling between logics and/or postulating axioms, but stays in this world (the same mathematical universe where all the development has taken place) and sees what this world has to offer: it turns out that clauses such as (*) are "backwards sound", in the sense that any relation satisfying them will include the empty-context Htyping relation. This yields "context-free" induction:

Prop 8: Assume θ : **dterm** $\rightarrow$ **tterm** $\rightarrow$ **bool** s.t.:

$$\frac{\forall X.\ \theta\ X\ tX\ \Rightarrow\ \theta\ (A \_ X)\ tY}{\theta\ (\mathsf{Lam}\ A)\ (\mathsf{Arr}\ tX\ tY)}\ \textbf{(ArrI}_\theta\textbf{)} \qquad \frac{\forall tX.\ \theta\ Y\ (tA \_ tX)}{\theta\ Y\ (\mathsf{Al}\ tA)}\ \textbf{(AlI}_\theta\textbf{)}$$

$$\frac{\theta\ Y\ (\mathsf{Arr}\ tX\ tZ) \qquad \theta\ X\ tX}{\theta\ (\mathsf{App}\ Y\ X)\ tZ}\ \textbf{(ArrE}_\theta\textbf{)} \qquad \frac{\theta\ Y\ (\mathsf{Al}\ tA)}{\theta\ Y\ (tA \_ tX)}\ \textbf{(AlE}_\theta\textbf{)}$$

Then $\Vdash X : tX$ implies $\theta\ X\ tX$ for all X, tX.
Proof sketch. Take R : **Hctxt** $\rightarrow$ **dterm** $\rightarrow$ **tterm** $\rightarrow$ **bool** to be $R\ \Delta\ X\ tX = ((\forall (Y : tY) \in \Delta.\ \theta\ Y\ tY) \Rightarrow \theta\ X\ tX)$. Then R satisfies the clause that define $\Vdash$, hence, in particular, for all X, tX, $\Vdash X : tX$ implies $R\ []\ X\ tX$, i.e., $\theta\ X\ tX$. ■

Viewing relations as nondeterministic functions, we can rephrase Prop. 8 in a manner closer to the intuition of types as sets of data, with a *logical predicate* [62] flavor:

Prop 8 (rephrased): Assume θ : **dterm** $\rightarrow$ $\mathbf{P}$(**tterm**) s.t.:

$$\frac{\forall X.\ X \in \theta\ tX\ \Rightarrow\ (A \_ X) \in \theta\ tY}{(\mathsf{Lam}\ A) \in \theta\ (\mathsf{Arr}\ tX\ tY)}\ \textbf{(ArrI}_\theta\textbf{)} \qquad \frac{\forall tX.\ Y \in \theta\ (tA \_ tX)}{Y \in \theta\ (\mathsf{Al}\ tA)}\ \textbf{(AlI}_\theta\textbf{)}$$

$$\frac{Y \in \theta\ (\mathsf{Arr}\ tX\ tZ) \qquad X \in \theta\ tX}{(\mathsf{App}\ Y\ X) \in \theta\ tZ}\ \textbf{(ArrE}_\theta\textbf{)} \qquad \frac{Y \in \theta\ (\mathsf{Al}\ tA)}{Y \in \theta\ (tA \_ tX)}\ \textbf{(AlE}_\theta\textbf{)}$$

Then $\Vdash X : tX$ implies $X \in \theta\ tX$ for all X, tX.

V. The HOAS principles at work

In this section we give a proof of strong normalization for System F within our HOAS representation using the developed definitional and proof machinery.

Remember that when introducing System F in Sec. II-B we fixed *infinite* sets of type and data variables, **dvar** and **tvar**, without making other assumption about their cardinalities. But now we commit to such an assumption, asking that we have much more type variables than data variables, namely, that **tvar** has a cardinality greater than or equal to that of $\mathbf{P}$(**dvar**). (This assumption is needed for obtaining a compositional map via Prop. 4.) One can easily see that this assumption does not affect the generality of the result, since once strong normalization has been proved for *some* fixed infinite cardinalities of the variable sets, then it can be inferred that it holds for *any* other infinite cardinalities – moreover, this also seems to be the case for most of the interesting properties considered for typing systems in the literature. Note also that this cardinality assumption has an intuitive reading in Cantorian set theory: think of types as sets of data, identify types with tterms and data with dterms; then, saying that $\mathsf{card}(\textbf{tvar}) = \mathsf{card}(\mathbf{P}(\textbf{dvar}))$ is the same as saying that $\mathsf{card}(\mathbf{tterm}) = \mathsf{card}(\mathbf{P}(\mathbf{dterm}))$, i.e., that types are indeed (in bijection with) sets of data.

A. An effective proof principle for typable terms

Before going after a proof of a particular property of System F, we first analyze how we could hypothetically employ our HOAS machinery in a potential proof. Many important properties of typed λ-calculi state something about the typable terms, with the statement possibly depending on the type. I.e., for a family $(G_{tX})_{tX} \in \prod_{tX \in \mathbf{tterm}} \mathbf{P}(\mathbf{dterm})$ (viewed as a **tterm**-sorted predicate), one would like to prove that $\Vdash X : tX$ implies $X \in G_{tX}$ for all X, tX. E.g.:
- Strong normalization: $G_{tX} = \{X.\, X \text{ strongly normalizing}\}$.
- Type preservation: $G_{tX} = \{X.\, \forall X'.\, X \rightsquigarrow X' \Rightarrow\ \Vdash X' : tX\}$.
- Church-Rosser: $G_{tX} = \{X.\, \forall Y_1, Y_2.\, X \rightsquigarrow Y_1 \wedge X \rightsquigarrow Y_1 \Rightarrow (\exists Z.\, Y_1 \rightsquigarrow Z \wedge Y_2 \rightsquigarrow Z)\}$.

One may also wish to prove the more general versions of these properties, which consider contexts as well. We call a subset $K \subseteq \mathbf{ctxt}$ *essentially context-free* if $[] \in K$ implies $K = \mathbf{ctxt}$. We think of K as a property on contexts. If the property is true (i.e., $K = \mathbf{ctxt}$), then it is trivially essentially context-free. The notion of essential context-freeness is thus only interesting for properties whose truth has not been established yet, and it says that it suffices to prove such properties for empty contexts only. The adverb "essentially" suggests that the effort of proving $K = \mathbf{ctxt}$ from $[] \in K$ is negligible, at least compared to that of proving $[] \in K$. This is the case of strong normalization, and also of type preservation, which is fortunate, since our HOAS induction principle (Prop. 8) is only applicable to empty-context versions of properties.

How would one go about proving that $\Vdash X : tX$ implies $X \in G_{tX}$ for all X? In order not to cramp the ideas with (meta)type dependencies that require extra notation but do not bring extra insight, we shall assume that all G_{tX}'s are equal,[4] i.e., that we start with a subset $G \subseteq \mathbf{dterm}$ and the question is: How would we go about proving that $\Vdash X : tX$ implies $X \in G$ for all X? Our "first reflex" is of course to use the HOAS-induction principle from Prop. 8, that is, search for $\theta : \mathbf{tterm} \to \mathbf{P}(\mathbf{dterm})$ with $Im(\theta) \subseteq G$, i.e., $\theta : \mathbf{tterm} \to \mathbf{P}(G)$, satisfying the clauses from there. Then Prop. 4 suggests a HOAS-recursive definition of θ. Interestingly, after some investigation, we are naturally led to a general criterion justifiable by the combination of HOAS induction and recursion (in what follows, we let Zs range over lists of terms and take $\mathsf{AppL} : \mathbf{dterm} \to \mathbf{List}(\mathbf{dterm}) \to \mathbf{dterm}$ to be defined by $\mathsf{AppL}\ X\ [] = X$ and $\mathsf{AppL}\ X\ (Z, Zs) = \mathsf{AppL}\ (\mathsf{App}\ X\ Z)\ Zs$; moreover, given a list Zs and a set G, we loosely write $Zs \subseteq G$ to indicate that all terms from Zs are in G):

Prop 9: Assume that $G \subseteq \mathbf{dterm}$ s.t. the following hold:

$$\frac{Zs \subseteq G}{\mathsf{AppL}\ y\ Zs \in G}\ (\mathbf{VCl}^G) \qquad \frac{\forall x.\ \mathsf{App}\ Y\ x \in G}{Y \in G}\ (\mathbf{AppCl}^G)$$

$$\frac{X \in G \quad Zs \subseteq G \quad \mathsf{AppL}\ (A\,\_\,X)\ Zs \in G}{\mathsf{AppL}\ (\mathsf{App}\ (\mathsf{Lam}\ A)\ X)\ Zs \in G}\ (\mathbf{Cl}^G)$$

Then $\Vdash X : A$ implies $X \in G$ for all X, A.

Proof sketch. Consider the following clauses, expressing potential properties of subsets $S \subseteq \mathbf{dterm}$ (assumed universally quantified over all the other parameters):
- (VCl^S): if $Zs \subseteq G$, then $\mathsf{AppL}\ y\ Zs \in S$;
- (Cl^S): if $X \in G$, $Zs \subseteq G$ and $\mathsf{AppL}\ (A\,\_\,X)\ Zs \in S$, then $\mathsf{AppL}\ (\mathsf{App}\ (\mathsf{Lam}\ A)\ X)\ Zs \in S$.

Let $\mathcal{C} = \{S \subseteq G.\ (\mathrm{VCl}^S) \text{ and } (\mathrm{Cl}^S) \text{ hold}\}$. We define $\mathsf{cArr} : \mathcal{C} \to \mathcal{C} \to \mathcal{C}$ and $\mathsf{cAl} : \mathbf{P}_{\neq\emptyset}(\mathcal{C}) \to \mathcal{C}$ by $\mathsf{cArr}\ S_1\ S_2 = \{Y.\, \forall X \in S_1.\, \mathsf{App}\ Y\ X \in S_2\}$ and $\mathsf{cAl}\ K = \bigcap K$.

By Prop. 4, there exists a map $\theta : \mathbf{tterm} \to \mathcal{C}$ that commutes with cArr and cAl, i.e.:
-(I) $\theta(\mathsf{Arr}\ tX\ tZ) = \{Y.\, \forall X \in \theta\ tX.\, \mathsf{App}\ Y\ X \in \theta\ tZ\}$.
-(II) $\theta(\mathsf{Al}\ tA) = \bigcap_{tX \in \mathbf{tterm}} \theta(tA\,\_\,tX)$.

Now, (II) is precisely the conjunction of the clauses (AlI_θ) and (AlE_θ) from Prop. 8 (rephrased), while the left-to-right inclusion part of (I) is a rephrasing of (ArrE_θ). Finally, (AlE_θ) holds because (Cl^S) holds for all $S \in \mathcal{C}$. Thus, the hypotheses of Prop. 8 (rephrased) are satisfied by $\theta : \mathbf{tterm} \to \mathcal{C}$ (regarded as a map in $\mathbf{tterm} \to \mathbf{P}(\mathbf{dterm})$). Hence, $\forall X, tX.\ \Vdash X : tX \Rightarrow X \in \theta\, tX$. And since $\forall tX.\, \theta\ tX \subseteq G$, we get $\forall X, tX.\ \Vdash X : tX \Rightarrow X \in G$. ■

We call a subset $G \subseteq \mathbf{dterm}$ *type-closed* (terminology taken from [41]) if it satisfies the hypotheses of Prop. 9.

B. Proof of strong normalization for System F

We let $\mathcal{SN}$ be the set of all strongly normalizing dterms.

Prop 10: (Strong Normalization) If $\Gamma \Vdash X : tX$, then $X \in \mathcal{SN}$.

Proof. Let $K_{SN} \subseteq \mathbf{ctxt}$ be $\{\Gamma.\ \forall X, tX.\ \Gamma \Vdash X : tX \Rightarrow X \in \mathcal{SN}\}$. It suffices to check: **(1)** K_{SN} is essentially context-free; **(2)** $\mathcal{SN}$ is type-closed. Indeed, by Prop. 9, (2) would ensure $[] \in K_{SN}$, hence, with (1), we would have $K_{SN} = \mathbf{ctxt}$. (1) and (2) are treated next. ■

Prop 11: K_{SN} is essentially context-free.

Prop 12: $\mathcal{SN}$ is type-closed.

The (very simple) proof of Prop. 11 is a mere rephrasing of an argument using the original syntax that reduces, for the two dterms Y and $\mathsf{Lam}(x.Y)$, well-typedness of the former to well-typedness of the latter and termination of the latter to termination of the former. On the other hand, the proof of Prop. 12 requires a tedious case analysis that mirrors that of the original proof (so here our HOAS approach does not bring any improvement). The latter proposition employs the following lemma, whose proof occasions the usage of the argument-permutative induction from Prop. 2:

Prop 13: If $X \rightsquigarrow^* X'$, then $A\,\_\,X \rightsquigarrow^* A\,\_\,X'$.

Proof. First, we note that $\mathsf{fresh}\ z\ A \wedge A\,\_\,z \rightsquigarrow^* A'\,\_\,z \Rightarrow \mathsf{Lam}\ A \rightsquigarrow^* \mathsf{Lam}\ A'$, from which we get
$(\forall z.\ A\,\_\,z \rightsquigarrow^* A'\,\_\,z) \Rightarrow \mathsf{Lam}\ A \rightsquigarrow^* \mathsf{Lam}\ A'$ (**)

Now, we employ the principle from Prop. 2, performing induction on A. For the only interesting case, assume A has

[4]But the type-dependent case could be handled along the same lines.

the form $\mathsf{Lam}\ \mathcal{A}$. We know from IH that $\forall z.\ (\mathcal{A}\_1\, z)\_X \rightsquigarrow^* (\mathcal{A}\_1\, z)\_X' \ \wedge\ (\mathcal{A}\_2\, z)\_X \rightsquigarrow^* (\mathcal{A}\_2\, z)\_X'$. The second conjunct gives $\forall z.(\mathcal{A}\_1\, X)\_z \rightsquigarrow^* (\mathcal{A}\_1\, X')\_z$, hence, with (**), $\mathsf{Lam}(\mathcal{A}\_1\, X) \rightsquigarrow^* \mathsf{Lam}(\mathcal{A}\_1\, X')$, i.e., $(\mathsf{Lam}\,\mathcal{A})\_X \rightsquigarrow^* (\mathsf{Lam}\,\mathcal{A})\_X'$. (We also used the exchange and commutation laws from Sec. III-B.) ■

The above proof reveals an interesting phenomenon: in a HOAS setting, where bindings are kept implicit and substitution is mere function application, in some proofs one may need to perform a permutation of the "placeholders" for function application (requiring 2-abstractions), whereas in a first-order framework one would be able to proceed more directly. Indeed, consider a first-order version of Prop. 13, stating that $\rightsquigarrow^*$ is substitutive: $X \rightsquigarrow^* X'$ implies $Y[X/x] \rightsquigarrow^* Y[X'/x]$. Its proof goes by induction on Y, treating the case of abstraction as follows: Assume $Y = \mathsf{Lam}(z, Z)$. By Barendregt's variable convention (made rigorous by work [67], [66] using Nominal Logic [55]), we may assume z fresh for x, X, X'. By IH, $Z[X/x] \rightsquigarrow^* Z[X'/x]$. By (Xi) (iterated), $\mathsf{Lam}(z.(Z[X/x])) \rightsquigarrow^* \mathsf{Lam}(z.(Z[X'/x]))$, hence (since z is fresh), $\mathsf{Lam}(z.Z)[X/x] \rightsquigarrow^* \mathsf{Lam}(z.Z)[X'/x]$, as desired.

The proof of the first-order version of the fact is more direct than that of the HOAS version because under the first-order view a term Y allows substitution *at any position*, i.e., at any of its free variables, while under the HOAS view an abstraction A has only *one particular position* "prepared" for substitution. Our definitional framework accommodates both the first-order and the HOAS proofs, since *the object syntax is the same*, being only subjected to two distinct views.

C. Our proof in the context of existing proofs

The first proof of strong normalization for System F was given in Girard's Ph.D. thesis [27], the very place where (a Church-typed version of) the system was introduced. All the proofs that followed employed in one way or another Girard's original idea of *reducibility candidates*, in later papers by different authors called (under slightly different technical conditions) *saturated sets* – Sec. 11 in [26] gives an overview. Variations in these proofs include the employment of terms that may or may not bear type annotations and technical adjustments on the "candidates". Our own proof follows Girard's idea as well, but brings a twofold improvement over previous proofs:

(1) It delves more directly into the heart of the problem – our *general-purpose* HOAS induction principle[5] expressed by Prop. 8 "invites" one to seek a notion of candidate.

(2) It does away with the notions of *typing context*, and *type* or *data environment*, which are employed in *all* the previous proofs as "auxiliaries" to the main proof idea. Indeed, previous proofs define a variant of our type evaluation map θ (required to apply Prop. 8) that is *parameterized by type environments*, i.e., by maps from tvars to tterms. Instead, we employ our compositionality criterion (Prop. 4) to obtain a lightweight, non-parameterized θ directly, verifying what is known as "Girard's trick" (namely, proving that it has its image in the set of candidates) in a more transparent fashion. Then, previous proofs define a notion of semantic deduction in contexts, universally quantifying over type environments and/or data environments, and prove the typing relation sound w.r.t. it – this step is *not* required by our proof; more precisely, this routine issue of logical soundness has been recognized as a general phenomenon pertaining to HOAS and has already been dealt with in the proof of Prop. 4.

On the formalization side, we are only aware of the LEGO [2] formalization from [6], and of the ATS [16] formalization from [21], both following [28]. The former uses de Bruijn encoding of the syntax, while the latter employs LF-style, axiomatic HOAS for data terms and de Bruijn indices for type terms. It appears that potential ATS variants of some of our results (mainly Props. 8 and 4) could have been used to "HOASify" (and simplify) the proof from [21] – in particular, our employment of Prop. 4 seems to answer the following question raised in loc. cit., on page 120: "[can one] prove strong normalization using a higher-order representation for types[?]". On the other hand, due to the partly axiomatic approach, the adequacy of the HOAS representation from loc. cit. (i.e., variants of our Props. 5 and 7) cannot be formally established in that setting.

VI. Conclusions, related work and future work

One purpose of this paper was to insist on, and bring technical evidence for, the advantage of using a general-purpose framework for HOAS, or, in other words, to employ HOAS within standard mathematics. We showed that our general-purpose framework offers access to some of the HOAS advanced conveniences, such as impredicative and context-free representations of (originally context-based) type systems. Another purpose was to bring, via an extensive HOAS exercise, more evidence to a belief seemingly shared by the whole HOAS community (beyond the large variety of proposed technical solutions), but not yet sustained by many examples in the literature (apart from those from [9]): that a HOAS representation of a system is in principle able not only to allow hassle-free manipulation and study of a system, but also to actually *shed more light on the deep properties of a system*. We believe that our general-purpose HOAS machinery does simplify and clarify the setting and justification of a notoriously hard result in type theory.

A. Future work – generalization

The constructions and results from Sec. III can be straightforwardly generalized to an arbitrary many-sorted syntax with bindings. Moreover, the constructions and adequacy proofs from Sec. IV seem to work for a large class of inductively defined inference systems in whose clauses the

[5]"General-purpose", in that it is *not* an ad hoc principle aimed at proving the particular strong normalization result, but a general one derived by mere syntactic analysis of the typing system; analogous principles are available for a large class of typing systems.

migration of variables between scopes satisfies a few general conditions, allowing the sound application of transformations (I)-(V) discussed in Sec. IV. We are currently working on determining such suitably general conditions.

B. Future work – full automation

Although our results have been formalized, we have not yet taken full advantage of the ample possibilities for automatically building the HOAS machinery. We are currently implementing (the general versions of) the results presented in Secs. III and IV as a definitional package in Isabelle/HOL. Our system will require the user to give a *binding signature* and a number of *inference system specifications* on the terms of this signatures (for various desired relations: typing, subtyping, type generality, reduction, etc.). From the binding signature, the system will produce the terms (one Isabelle type of terms for each syntactic category), as well as all the standard operators on them (substitution, free variables etc.) and prove the standard lemmas about these. (Currently, we have the underlying FOAS machinery formalized for an *arbitrary binding signature* (see the document FOAS.pdf from [56]), but need to write some template proofs for the simple facts required for instantiating this signature based on user input.) From the inference system specifications, the system will produce the actual inductive definitions of the intended relations. Then the system will construct, along the lines of this paper, the HOAS view of syntax (defining new higher-order operators on terms and proving their properties) and the representation of inference, which will be automatically proved adequate. General versions of the propositions in this paper's Secs. III and IV shall also be proved (automatically). All in all, based on a very compact input from the user our system will produce: **(i)** the intended object system with all its basic first-order constructions; **(ii)** a HOAS representation formally certified as adequate.

We also plan to engage our system in formalizing other extensive case studies, including the POPLmark challenge [4] and formalizing the meta-theory of π-like calculi [58].

It would be interesting to investigate to which extent different theorem provers could support (and perhaps simplify) the formal development of our results. For instance, Coq would free us from having to write template proofs, as the general results for an arbitrary syntax with bindings could be stated over families of types, thus being directly instantiable to particular syntaxes. On the other hand, our employment of the Hilbert choice operator in Isabelle in the crucial definition of application (as being substitution, for *some* representatives) is not directly supported by Coq.

C. More related work

There is a very extensive literature on the subject of syntax representation in general and on HOAS in particular. We only mention some works most directly relevant here. The HOAS-tailored framework approach yielded several theorem provers and functional programming environments (some of them already mature and with an extensive case-study record), including several extensions of LF – Twelf [5], Delphin [1], ATS [16], Beluga [53] – and Abella [3], a HOAS-specialized prover based on definitional reflection. On the other hand, the Hybrid package [7], written in Isabelle/HOL, is a successful realization of the general-purpose framework approach. Later versions of this system [44], [46], [22] also import the three-level architecture idea from the HOAS-tailored framework approach. Our context-free induction principle from Prop. 8 captures the (non-inductive) open-world situation from a HOAS-tailored setting while avoiding the need for an exotic logic or for a "third-party" logic.

Another standard classification of HOAS approaches is in *weak* versus *strong* HOAS. Both capture object-level bindings by meta-level *functional* bindings; "weak" refers to the considered functions mapping *variables* to terms, while "strong" refers to these functions mapping *terms* to terms. Weak HOAS approaches are taken in [18], [34], [57], [30], including in category-theoretic form (with a denotational-semantics flavor) in [23], [33], [8], [24]. Our work in this paper, the above HOAS-tailored approaches, as well as [19], the work on Hybrid [7], [44], [46], [22], as well as parametric HOAS [17], parametricity-based HOAS [35],[6] and de-Bruijn-mixed-HOAS [32], fall within strong HOAS. In weak HOAS, some of the convenience is lost, since substitution of terms for variables is not mere function application, as in strong HOAS. On the other hand, weak HOAS is is easier to define directly inductively. However, as illustrated in this paper and in previous work [19], [7], in a general-purpose setting having strong HOAS (perhaps on top of weak HOAS as in [19], or directly on top of the first-order syntax as here) is only a matter of some *definitional* work. Because variables are particular terms, strong HOAS can accommodate weak induction and recursion principles, and in fact in most situations only such weak principles are available due to the need of well-foundedness – Prop. 1 (similar to an axiom postulated in the Theory of Contexts [34], [15] and to a fact proved by Hybrid [7]), as well as our permutative induction for 2-abstractions expressed in Prop. 2, are examples of "weak" principles within strong HOAS. To our knowledge, our Prop. 4 is the first genuinely "strong" (albeit restricted) compositionality principle for syntax interpretation within general-purpose frameworks.

Acknowledgements. We thank the reviewers for their insightful comments and suggestions.

References

[1] Delphin. http://cs-www.cs.yale.edu/homes/carsten/delphin.

[2] LEGO. http://www.dcs.ed.ac.uk/home/lego.

[3] Abella Theorem prover, 2009. http://abella.cs.umn.edu/.

[4] The POPLmark challenge, 2009. http://fling-l.seas.upenn.edu/ plclub/cgi-bin/poplmark/.

[6]The import of the notion of parametricity into HOAS was apparently pioneered by [60], [20].

[5] The Twelf Project, 2009. http://twelf.plparty.org/.

[6] T. Altenkirch. A formalization of the strong normalization proof for System F in LEGO. In *TLCA*, pages 13–28, 1993.

[7] S. Ambler, R. L. Crole, and A. Momigliano. Combining Higher Order Abstract Syntax with tactical theorem proving and (co)induction. In *TPHOLs*, pages 13–30, 2002.

[8] S. J. Ambler, R. L. Crole, and A. Momigliano. A definitional approach to primitive recursion over Higher Order Abstract Syntax. In *MERLIN*, 2003.

[9] A. Avron, F. Honsell, I. A. Mason, and R. Pollack. Using typed λ-calculus to implement formal systems on a machine. *J. of Aut. Reasoning*, 9(3):309–354, 1992.

[10] H. Barendregt. Introduction to generalized type systems. *J. Funct. Program.*, 1(2):125–154, 1991.

[11] H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*. Oxford University Press, 1992.

[12] H. P. Barendregt. *The Lambda Calculus*. North-Holland, 1984.

[13] M. Berger, K. Honda, and N. Yoshida. Genericity and the pi-calculus. *Acta Inform.*, 42(2):83–141, 2005.

[14] Y. Bertot and P. Casteran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

[15] A. Bucalo, F. Honsell, M. Miculan, I. Scagnetto, and M. Hofmann. Consistency of the theory of contexts. *J. Funct. Program.*, 16(3):327–372, 2006.

[16] C. Chen and H. Xi. Combining programming with theorem proving. In *ICFP*, pages 66–77, 2005.

[17] A. J. Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *ICFP*, pages 143–156, 2008.

[18] J. Despeyroux, A. P. Felty, and A. Hirschowitz. Higher-order abstract syntax in Coq. In *TLCA*, pages 124–138, 1995.

[19] J. Despeyroux and A. Hirschowitz. Higher-Order Abstract Syntax with induction in Coq. In *LPAR*, pages 159–173, 1994.

[20] J. Despeyroux and P. Leleu. Recursion over objects of functional type. *Mathematical Structures in Computer Science*, 11(4):555–572, 2001.

[21] K. Donnelly and H. Xi. A formalization of strong normalization for simply-typed lambda-calculus and system F. *Electron. Notes Theor. Comput. Sci.*, 174(5):109–125, 2007.

[22] A. P. Felty and A. Momigliano. Hybrid: A definitional two-level approach to reasoning with Higher-Order Abstract Syntax. *CoRR*, abs/0811.4367, 2008.

[23] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding (extended abstract). In *LICS*, pages 193–202, 1999.

[24] M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the π-calculus. In *LICS*, pages 43–54, 1996.

[25] A. Gacek, D. Miller, and G. Nadathur. Combining generic judgments with recursive definitions. In F. Pfenning, editor, *LICS*, pages 33–44, June 2008.

[26] J. Gallier. On Girard's candidats de reductibilite. In *Logic and Computer Science*, pages 123–203. Academic Press, 1990.

[27] J.-Y. Girard. Une extension de l'interpretation de Gödel a l'analyse, et son application a l'elimination des coupure dans l'analyse et la theorie des types. In *2nd Scandinavian Logic Symposium*, pages 63–92. 1971.

[28] J.-Y. Girard. *Proofs and Types*. Cambridge University Press, 1989.

[29] C. A. Gunter. *Semantics of Programming Languages. Structures and Techniques*. The MIT Press, 1992.

[30] E. L. Gunter, C. J. Osborn, and A. Popescu. Theory support for weak Higher Order Abstract Syntax in Isabelle/HOL. In *LFMTP*, pages 12–20, 2009.

[31] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *LICS*, pages 194–204. IEEE, Computer Society Press, 1987.

[32] J. Hickey, A. Nogin, X. Yu, and A. Kopylov. Mechanized meta-reasoning using a hybrid HOAS/de Bruijn representation and reflection. In *ICFP*, pages 172–183, 2006.

[33] M. Hofmann. Semantical analysis of higher-order abstract syntax. In *LICS*, page 204, 1999.

[34] F. Honsell, M. Miculan, and I. Scagnetto. An axiomatic approach to metareasoning on nominal algebras in HOAS. In *ICALP*, pages 963–978, 2001.

[35] D. J. Howe. Higher-order abstract syntax in classical higher-order logic. In *LFMTP*, pages 1–11, 2009.

[36] G. P. Huet and B. Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Inf.*, 11:31–55, 1978.

[37] R. Loader. Normalization by calculation. Unpublished note, 1995. http://homepages.ihug.co.nz/ suckfish/papers/normal.pdf.

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

[39] R. C. McDowell. *Reasoning in a logic with definitions and induction*. PhD thesis, University of Pennsylvania, 1997.

[40] D. Miller and A. Tiu. A proof theory for generic judgments. *ACM Transactions on Computational Logic*, 6(4):749–783, 2005.

[41] J. C. Mitchell. A type-inference approach to reduction properties and semantics of polymorphic expressions (summary). In *LISP and Functional Programming*, pages 308–319, 1986.

[42] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[43] J. C. Mitchell and A. R. Meyer. Second-order logical relations (extended abstract). In *CLP*, pages 225–236, 1985.

[44] A. Momigliano and S. Ambler. Multi-level meta-reasoning with higher-order abstract syntax. In *FoSSaCS*, pages 375–391, 2003.

[45] A. Momigliano, S. J. Ambler, and R. L. Crole. A comparison of formalizations of the meta-theory of a language with variable bindings in isabelle. Technical report, Supplemental Proceedings of TPHOL'01, 2001.

[46] A. Momigliano, A. J. Martin, and A. P. Felty. Two-level Hybrid: A system for reasoning using Higher-Order Abstract Syntax. *Electron. Notes Theor. Comput. Sci.*, 196:85–93, 2008.

[47] T. Nipkow. Structured Proofs in Isar/HOL. In *TYPES*, pages 259–278, 2003.

[48] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer, 2002.

[49] L. C. Paulson. The foundation of a generic theorem prover. *J. Autom. Reason.*, 5(3):363–397, 1989.

[50] F. Pfenning. Logical frameworks. In *Handbook of Automated Reasoning*. Elsevier Science, 1999.

[51] F. Pfenning. Logical frameworks - a brief introduction. In *Paris Colloqvium on Programming*, pages 137–166. 2002.

[52] F. Pfenning and C. Elliot. Higher-order abstract syntax. In *PLDI*, pages 199–208, 1988.

[53] B. Pientka. Beluga: Programming with dependent types, contextual data, and contexts. In *FLOPS*, pages 1–12, 2010.

[54] B. C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[55] A. M. Pitts. Nominal logic: A first order theory of names and binding. In *TACS*, pages 219–242, 2001.

[56] A. Popescu. HOAS on top of FOAS formalized in Isabelle/HOL. Tech. Rep., Univ. of Illinois at Urbana-Champaign, 2010. http://hdl.handle.net/2142/15449.

[57] C. Röckl and D. Hirschkoff. A fully adequate shallow embedding of the [pi]-calculus in Isabelle/HOL with mechanized syntax analysis. *J. Funct. Program.*, 13(2):415–451, 2003.

[58] D. Sangiorgi and D. Walker. *The π-calculus. A theory of mobile processes*. Cambridge, 2001.

[59] C. Schurmann. *Automating the meta-theory of deductive systems*. PhD thesis, Carnegie Mellon University, 2000.

[60] C. Schurmann, J. Despeyroux, and F. Pfenning. Primitive recursion for higher-order abstract syntax. *Theor. Comput. Sci.*, 266(1-2):1–57, 2001.

[61] C. Schurmann and F. Pfenning. Automated theorem proving in a simple meta-logic for LF. In *CADE*, pages 286–300, 1998.

[62] R. Statman. Logical relations and the typed lambda-calculus. *Information and Control*, 65(2/3):85–97, 1985.

[63] Y. Sun. An algebraic generalization of Frege structures - binding algebras. *Theor. Comput. Sci.*, 211(1-2):189–232, 1999.

[64] W. Tait. A realizability interpretation of the theory of species. In *Logic Colloquium*, pages 240–251. Springer, 1975.

[65] A. Tiu. *A Logical Framework for Reasoning about Logical Specifications*. PhD thesis, Penn State University, 2004.

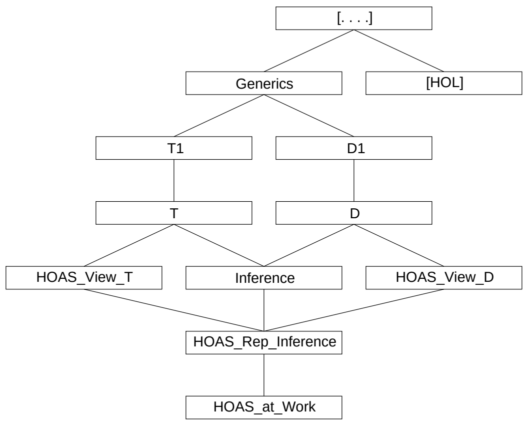
[66] C. Urban, S. Berghofer, and M. Norrish. Barendregt's variable convention in rule inductions. In *CADE*, pages 35–50, 2007.

[67] C. Urban and M. Norrish. A formal treatment of the Barendregt variable convention in rule inductions. In *MERLIN*, pages 25–32, 2005.

APPENDIX

More details on the formalization

The diagram in the figure shows the relevant part of our theory structure in Isabelle. In fact, the part consisting of the theories D and T and the ones below them matches faithfully the structure of this paper and is conceptually self-contained.



FOAS

The binding first-order preliminaries required by our HOAS approach have been formalized for an arbitrary (possibly infinitary) many-sorted syntax with bindings (not shown in the diagram). While we believe this has some interest in its own right, here we have neither the space nor the need to discuss it. (But the interested reader can look at the document FOAS.pdf from [56].)

We have instantiated these general results to the syntax of System F. Theory D1 performs the instantiation to dterms. (It consists of many simple facts about transiting back and forth between the abstract notions of an input to a binding operator (of specified arity) and the concrete binding arguments of this syntax. The inference of these simple but tedious facts will be soon automated.) The reader does *not* have to look into D1 in order to comprehend our HOAS formalization. Everything required w.r.t. the FOAS of dterms is contained in theory D, which accumulates all the relevant FOAS facts on the syntactic constructs, fresh variables, substitution, as well as swapping (the latter not discussed in the paper). As mentioned, this very large collection of first-order facts comes from a corresponding one on an arbitrary syntax with bindings and will soon be completely automated as an Isabelle package. The roles played by D1 and D for dterms are played by theories T1 and T for tterms.

Thus, theories D and T correspond to the first part of the paper's Sec. II. The notations we used in these theories match the ones from the paper, save for some variations required to please the Isabelle parser. E.g., $Y[X/x]$ from the paper becomes $Y\#[X./x]$ in the scripts. The reader may ignore these extra symbols and focus on "the shape" of the notation. Next we refer to the theory D, but the discussion applies to T as well. The reader may legitimately wonder how is one to be sure that we are talking about *the* syntax of System F, i.e., that it has been correctly defined, without looking into our generic theory of syntax with bindings. The answer is provided by the first two sections of D, containing properties that characterize the types of dterms and dabstractions together with the operations on them *uniquely, up to isomorphism* – in a nutshell: (almost-)injectivity and completeness of the syntactic constructs, together with the induction principle, determine the types uniquely, and the simplification rules for freshness, substitution and swapping act like definitional clauses. The theory Inference defines beta-reduction and typing for System F – therefore the theory trio (D,T,Inference) matches this paper's Sec. II.

HOAS

Again, here the proof development matches faithfully the sectionwise (and, to some extent, also the subsectionwise) structure of the paper.

The theories HOAS_View_D and HOAS_View_T formalize the HOAS view of the syntax of dterms and tterms, respectively, thus matching Sec. III. Next we refer to HOAS_View_D only (since HOAS_View_T is similar). The definitions of abstraction application and the other operators given in the paper informally and claimed to be independent of representatives are first given in Isabelle by picking *some* representatives. E.g. the operators varOfAbs and termOfAbs pick *together* a representative (x, Y) for an abstraction A, and then A_X is defined to be $Y[X/x]$; then, "the real definition", not committing to any particular such pair, is stated as a lemma: "$A\_X = Y[X/x]$ for *all* x, Y such that $A = (x.Y)$". (Note that in the scripts we write Dabs x Y for $(x.Y)$.) While the induction principles from Sec. III-C are rendered verbatim in the scripts, the formalizations of the recursive definition principles from Sec. III-D have a slightly different form, reflecting Isabelle's distinction between a type and a set. E.g., to obtain a flexible Isabelle version of Prop. 4, we have the domain C from there represented not merely by a type, but by a type c together with a "well-structured-ness" predicate cWls : $c \rightarrow$ **bool**. Then a compositional map H as in Prop. 4 is called there a "HOAS-morphism"; the existence of such a map is stated in the scripts as Th. ex_HOASmorph, and then rephrased as Th. ex_comp_, which matches Prop. 4 more closely.

The theory HOAS_Rep_Inference formalizes the HOAS representation of inference, discussed in Sec. IV The three subsections of this theory match those of Sec. IV. Our HOAS inference employs infinitary inductive clauses, but these are unproblematic in Isabelle, both definition-wise and proof-wise (of course, it is their uniformity that makes them unproblematic proof-wise). While in the paper we assume **ctxt** $\subseteq$ **Hctxt**, in the formalization we have an explicit injection asHctxt : **ctxt** $\rightarrow$ **Hctxt**, and a predicate isCtxt : **Hctxt** $\rightarrow$ **bool** representing its image. As a "psychological" note, Isabelle figures out automatically the proof of Prop. 8 once we indicate the relation R, while for the human mind this is somewhat difficult to grasp, as is any statement whose justification involves implications nested on the left, as in $(\varphi \Rightarrow \chi) \Rightarrow \psi$. (This is also part of the difficulty of comprehending Girard's proof technique, and, more generally, the method of logical relations.)

The theory HOAS_at_Work formalizes the strong normalization proof, corresponding to this paper's Sec. V. Here is the content of this theory. First we prove the type-closedness criterion, Prop. 9. Then we prove Prop. 13 – we actually give two alternative proofs of this, reflecting the paper's discussion following Prop. 13. Then we make further preparations for the proof of Prop. 12 in terms of some variations of the notion of reduction-simulation. Finally, we prove strong normalization, via verbatim renderings of Props. 12 and 11.

Our Isabelle scripts can be downloaded from [56]. The document SysF.pdf from that (zipped) folder contains a detailed presentation of the relevant theories. These theories can also be browsed in html format in the folder SysF_Browse (note that the browsable format shows also all the background (FOAS) theories needed for our HOAS work).

Here is a list of further differences between the paper and the Isabelle scripts:

- Isabelle uses $\Rightarrow$ for function space and $\longrightarrow$ and $\Longrightarrow$ for logical implication. It also uses $\forall$, $\bigwedge$ (the latter also written !!) for universal quantification. (There are differences between the two Isabelle versions of implications and universal quantifications, but they can be ignored by the reader.)
- Isabelle uses :: for membership to a type, and $\in$ (also written :) for membership to a set. We have ignored this distinction in the paper.
- Prefix $\#$ indicates swapping or substitution on terms, and \$ and $\%$ the same operation on abstractions and environments, respectively.)
- freshAbs and freshEnv (instead of fresh) are used for the freshness operators on abstractions and environments, respectively.
- Dabs x X and Dabs2 x A (instead of $x.X$ and $x.A$) are used for the first-order dabstraction and 2-dabstraction constructs.
- Similarly, Tabs x *tX* and Tabs2 x *tA* (instead of $x.$*tX* and $x.$*tA*) are used for the first-order tabstraction and 2-tabstraction constructs.
- In theories T and T1, since there is no overlap (yet) with data items, we do not prefix the variable names by "t".
- In the Isabelle scripts we have three kinds of notations for substitutions: arbitrary substitution in environments, $X[\rho]$, unary substitution ("usubst") $X[Y/y]$, and variable-for-variable unary substitution ("vusubst") $X[x//y]$; we also have (variable-for-variable) swapping, written $X[x \wedge y]$.
- While the paper keeps some injections implicit, in Isabelle we represent them explicitly:
  - dInV : **dvar** $\rightarrow$ **dterm**, the injection of dvars as dterms;
  - tInV : **tvar** $\rightarrow$ **tterm**, the injection of tvars as tterms;
  - asHctxt : **ctxt** $\rightarrow$ **Hctxt**, the injection of contexts as Hcontexts;
  - isCtxt : **Hctxt** $\rightarrow$ **bool**, the predicate checking if an Hcontexts is (the image of) a **ctxt**;