RAIRO-Inf. Theor. Appl. **39** (2005) 423-453
DOI: 10.1051/ita:2005026

DENOTATIONAL ASPECTS OF UNTYPED NORMALIZATION BY EVALUATION *

Andrzej Filinski[1] and Henning Korsholm Rohde[2]

Abstract. We show that the standard normalization-by-evaluation construction for the simply-typed $\lambda_{\beta\eta}$-calculus has a natural counterpart for the untyped λ_{β}-calculus, with the central type-indexed logical relation replaced by a "recursively defined" *invariant relation*, in the style of Pitts. In fact, the construction can be seen as generalizing a computational-adequacy argument for an untyped, call-by-name language to normalization instead of evaluation.In the untyped setting, not all terms have normal forms, so the normalization function is necessarily partial. We establish its correctness in the senses of *soundness* (the output term, if any, is in normal form and β-equivalent to the input term); *identification* (β-equivalent terms are mapped to the same result); and *completeness* (the function is defined for all terms that do have normal forms). We also show how the semantic construction enables a simple yet formal correctness proof for the normalization algorithm, expressed as a functional program in an ML-like, call-by-value language. Finally, we generalize the construction to produce an infinitary variant of normal forms, namely *Böhm trees*. We show that the three-part characterization of correctness, as well as the proofs, extend naturally to this generalization.

Mathematics Subject Classification. 03B40, 06B35, 68N18, 68Q55.

Keywords and phrases. Normalization by evaluation, untyped λ-calculus, denotational semantics, functional programming, Böhm trees, computational adequacy.

* *An earlier version of this article appeared in the proceedings of FOSSACS 2004* [6]. *An extended version, with detailed proofs, is available as a technical report* [7].

[1] DIKU, Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark; andrzej@diku.dk

[2] BRICS†, Department of Computer Science, University of Aarhus, IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark; hense@brics.dk

† Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.

1. Introduction

1.1. Reduction-based and reduction-free normalization

Traditional accounts of term normalization are based on a directed notion of *reduction* (such as β-reduction), which can be applied anywhere within a term. A term is said to be a *normal form* if no reductions can be performed on it. If the reduction relation is confluent, normal forms are uniquely determined, so normalization is a (potentially partial) function on terms. Some terms (such as Ω) may not have normal forms at all; or a particular reduction strategy (such as normal-order reduction) may be required to guarantee arrival at a normal form when one exists; such a strategy is called *complete*. There is a very large body of work dealing with normalization in reduction-based settings.

However, in recent years, a rather different notion of normalization has emerged, so-called *reduction-free normalization*. As the name suggests, it is not based on a directed notion of reduction, but rather on an undirected notion of term *equivalence*. Equivalence may be defined as simply the reflexive-transitive-symmetric closure of an existing reduction relation, but it does not have to be: any congruence relation on terms may be used. The task is then to define a *normalization function* on terms, such that the output of the function is equivalent to the input, and such that any two equivalent terms are mapped to identical outputs [4].

For some notions of equivalence (such as β-convertibility of untyped lambda-terms), it is actually impossible to define a *computable*, total normalization function with both of these properties; we must thus accept that the normalization function may be partial. However, even in that case, we can impose a completeness constraint: if we have an independent syntactic characterization of acceptable *normal forms*, we can require that the function both produce terms in this form as output, and that it be defined on all terms equivalent to a normal form.

1.2. Normalization by evaluation

A particularly natural way of obtaining a reduction-free normalization function is known as *normalization by evaluation (NBE)*, based on the following idea: Suppose we can construct a denotational model of the term syntax (*i.e.*, such that equivalent terms have the same denotation), with the property that a syntactic representation of any normal-form term can be extracted from its denotation; such a model is called *residualizing*. Then the normalization function can be expressed simply as a compositional interpretation in the model, followed by extraction.

A priori, such a normalization function is not necessarily effectively computable. It can be given a computational interpretation if the denotational model is constructed in intuitionistic set theory [4], but this gets somewhat complicated for domain-theoretic models, especially those involving reflexive domains. In such cases, it is often easier to establish that the constructions are effective by showing that they can expressed as images of program terms in a language for which the domain-theoretic semantics is already known to be computationally adequate.

(It should be noted that the term NBE is also sometimes used for a related concept, based on reducing – usually in a compositional way – the *normalization problem*, which may in general involve open terms of higher type, to an *evaluation problem*, which involves normalization of only closed terms of base type. The required transformation is often syntactically related to the model-based construction above, but the model itself is not made explicit; and in fact, the subsequent evaluation process may still be specified entirely in terms of reductions.)

1.3. The Berger-Schwichtenberg normalization algorithm

Perhaps the best-known NBE algorithm is due to Berger and Schwichtenberg [3]. It finds $\beta\eta$-long normal forms of simply-typed λ-terms. We present here its outline, glossing over inessential details.

Types are of the form $\tau ::= b \mid \tau_1 \to \tau_2$. A natural set-theoretic model interprets each base type b as some set, and the function type as the set of all functions between the interpretations of the types, *i.e.*, $[\![\tau_1 \to \tau_2]\!] = [\![\tau_1]\!] \to [\![\tau_2]\!]$. For a type assignment Γ, we also take $[\![\Gamma]\!] = \prod_{x \in \mathrm{dom}\,\Gamma} [\![\Gamma(x)]\!]$.

Let Λ be the set of syntactic λ-terms (written with explicit constructors for emphasis) over a set of variables V. For a well-typed term $\Gamma \vdash m : \tau$, we can then express its semantics $[\![m]\!] \in [\![\Gamma]\!] \to [\![\tau]\!]$ as follows:

$$\begin{aligned} [\![\mathrm{VAR}(x)]\!]\,\rho &= \rho(x) \\ [\![\mathrm{LAM}(x^\tau, m_0)]\!]\,\rho &= \lambda a^{[\![\tau]\!]}.\, [\![m_0]\!]\,\rho[x \mapsto a] \\ [\![\mathrm{APP}(m_1, m_2)]\!]\,\rho &= [\![m_1]\!]\,\rho\,([\![m_2]\!]\,\rho). \end{aligned}$$

It is easy to check that such a model is sound for conversion, *i.e.*, that when $m \leftrightarrow_{\beta\eta} m'$, then $[\![m]\!] = [\![m']\!]$.

Consider now a model where all base types are interpreted as the set of (open) syntactic λ-terms, *i.e.*, $[\![b]\!] = \Lambda$ for all b. In this model, we can define a pair of type-indexed function families – *reification*, $\downarrow^\tau : [\![\tau]\!] \to \Lambda$, and *reflection*, $\uparrow^\tau : \Lambda \to [\![\tau]\!]$ – by mutual induction on the type index τ:

$$\begin{aligned} \downarrow^b l &= l & \qquad \downarrow^{\tau_1 \to \tau_2} f &= \mathrm{LAM}(x^{\tau_1}, \downarrow^{\tau_2} (f(\uparrow^{\tau_1} \mathrm{VAR}(x)))) \quad (x \text{ “fresh”}) \\ \uparrow^b l &= l & \qquad \uparrow^{\tau_1 \to \tau_2} l &= \lambda a^{[\![\tau_1]\!]}.\, \uparrow^{\tau_2} (\mathrm{APP}(l, \downarrow^{\tau_1} a)). \end{aligned}$$

For simplicity, let us only consider normal forms of closed terms. Then reification can serve directly as the extraction function: one can check that, for a term $\vdash m : \tau$ in $\beta\eta$-long normal form, $\downarrow^\tau ([\![m]\!]\,\emptyset) \leftrightarrow_\alpha m$. Hence, by soundness of the model, for any term m' with $m' \leftrightarrow_{\beta\eta} m$, $\downarrow^\tau ([\![m']\!]\,\emptyset) = \downarrow^\tau ([\![m]\!]\,\emptyset) \leftrightarrow_\alpha m \leftrightarrow_{\beta\eta} m'$. Alternatively, one can show the latter property directly, for an arbitrary m'. Either way, the typical proof ultimately involves a logical-relations argument, even if this argument is pushed entirely into a standard result about the syntax (namely, that every well-typed term has a $\beta\eta$-long normal form). The latter approach, however, generalizes better, especially to systems where not all terms have normal forms.

1.4. A tentative algorithm for untyped terms

In an untyped (or, more accurately, unityped) setting, we may hope to get a residualizing model by interpreting the single type of terms as a domain $D = \Lambda + (D \to D)$. (Again, we gloss over domain-theoretic subtleties for expository purposes.) We can then define variants of reification, $\downarrow : D \to \Lambda$, and reflection, $\uparrow : \Lambda \to D$, roughly analogous to the simply-typed case:

$$\begin{aligned} \downarrow d &= \text{case } d \text{ of } \begin{cases} in_1(l) \to l \\ in_2(f) \to \text{LAM}(x, \downarrow (f(\uparrow \text{VAR}(x)))) \quad (x \text{ “fresh”}) \end{cases} \\ \uparrow l &= in_1(l). \end{aligned}$$

Note that reification is now defined by general recursion, rather than induction. We can also construct an interpretation, $[\![m]\!] \in (\text{V} \to D) \to D$, by

$$\begin{aligned} [\![\text{VAR}(x)]\!]\,\rho &= \rho(x) \\ [\![\text{LAM}(x, m_0)]\!]\,\rho &= in_2(\lambda d.\, [\![m_0]\!]\,\rho[x \mapsto d]) \\ [\![\text{APP}(m_1, m_2)]\!]\,\rho &= \text{case } [\![m_1]\!]\,\rho \text{ of } \begin{cases} in_1(l) \to \uparrow (\text{APP}(l, \downarrow ([\![m_2]\!]\,\rho))) \\ in_2(f) \to f\,([\![m_2]\!]\,\rho). \end{cases} \end{aligned}$$

Here, reflection is performed “on demand”: when application needs a semantic function, but $[\![m_1]\!]\rho$ is a piece of syntax, it is reflected just enough to allow the application to be performed.

Again, it can be checked that β-convertible terms have the same denotation. It is also fairly easy to verify that, for a closed m in β-normal form, $\downarrow ([\![m]\!]\,\emptyset) \leftrightarrow_\alpha m$. What is not obvious at all, however, is that when $\downarrow ([\![m']\!]\,\emptyset) = m$ for a general m', then m' must be syntactically β-convertible to a normal form. Indeed, the problem is a generalization of the usual computational-adequacy problem for a denotational semantics of a functional language: if the denotation of a closed term is not $\perp$ (undefined), must the term then evaluate to a value?

For a simply typed language, PCF, adequacy of the natural domain-theoretic semantics was shown by Plotkin, using a logical-relations argument [13]. Pitts showed that essentially the same argument applies to an untyped language, except that the central relation is no longer constructed by induction on types, but as a solution of a more general “relation equation”; he also showed a general method for solving such equations, yielding *invariant relations* [11].

In this paper, we first formalize the construction of the normalization function from above, addressing especially the issues of potential divergence and generation of fresh variable names (Sect. 2). We then show correctness of this function by a generalized computational-adequacy construction (Sect. 3), and how the domain-theoretic analysis directly validates a functional program implementing the construction (Sect. 4). Finally, we show how the construction can be generalized naturally to Böhm trees (Sect. 5).

1.5. Related work

The closest related work to ours is probably the NBE-based (in the alternate, reduction-oriented sense) algorithm for untyped β-normalization proposed by Aehlig and Joachimski [1]. However, while the functional programs ultimately derived from the analyses are quite similar, the correctness arguments are completely different: theirs are based on syntactic concepts and results from higher-order rewriting theory, rather than on the domain-theoretic constructions underlying ours.

We believe that the domain-theoretic approach enables a more direct and precise correctness proof for the normalizer, as actually implemented. In Aehlig and Joachimski's work, the abstract algorithm is expressed as a small-step operational semantics for a specialized, two-level λ-calculus with named bound variables; whereas the actual normalization program is formulated as a compositional interpreter in Haskell, using de Bruijn indices for bound variables, and a reflexive type for the meanings of higher-typed terms. It thus remains a potentially significant task to verify that the concrete Haskell program correctly implements the abstract algorithm. On the other hand, formally relating the domain-theoretic constructions in the model-based normalizer to the functional terms implementing them, is completely straightforward.

An untyped, reduction-based NBE-like algorithm can also be found in disguise in Grégoire and Leroy's work [8], whose focus is on compilation. Their concrete algorithm of strong reduction (*i.e.*, β-normalization) by iterated symbolic weak reduction (akin to $\uparrow$ and $[\![\cdot]\!]$) and readback (akin to $\downarrow$) is ultimately quite similar to ours. Their algorithm also handles several language extensions, such as inductive datatypes and *guarded* fixpoints. However, as they consider only a strongly-normalizing fragment of the λ-calculus, establishing correctness becomes significantly simpler. Their implementation takes the form of an abstract machine, whose (5000-line) correctness proof is mechanically checked using a proof assistant. They do not mention how the abstract machine is actually implemented.

Many of the constructions in the present paper are inspired by the first author's work on type-directed partial evaluation [5]. Apart from the obvious differences arising from typed vs. untyped languages, a significant change is also that the TDPE work considered equivalence defined semantically (equality of denotations, under all interpretations of "dynamic" constants), while here we consider syntactic β-convertibility. Accordingly, the central invariant relation ties denotations to syntactic terms, rather than to denotations in another semantics.

Essentially the same program as in Section 4, but expressed in FreshML, appears in a recent paper by Shinwell *et al.* [14], Figure 7. However, the focus there is on a practical application of fresh-name generation, rather than on normalization as such. Indeed, the underlying algorithm (informally attributed to Coquand) is not supported by a formal correctness argument. In our variant, generation of fresh names is handled explicitly: since constructed output terms are never subsequently analyzed by pattern-matching, using a general framework such as FreshML, or

higher-order abstract syntax, is probably overkill. However, we anticipate that a different "back end" for output generation could be used, and have deliberately tried to keep the constructions and proofs modular with respect to the term-generation operations. We thus expect that essentially the same arguments – perhaps even a little simplified – could be used to verify correctness of the FreshML variant of the normalizer as well.

2. A semantic normalization construction

2.1. Syntax and semantics of the untyped λ-calculus

2.1.1. *Syntax*

Let $\mathrm{V} = \{\mathrm{x}, \mathrm{y}, \ldots\}$ be a countably infinite set of (object) variables, with x and v ranging over V. The set of λ-terms m is then the least set Λ such that

$$\begin{aligned}\Lambda \;=\; & \{\mathrm{VAR}(x) \mid x \in \mathrm{V}\} \cup \{\mathrm{LAM}(x, m_0) \mid x \in \mathrm{V}, m_0 \in \Lambda\} \cup \\ & \{\mathrm{APP}(m_1, m_2) \mid m_1 \in \Lambda, m_2 \in \Lambda\}\,.\end{aligned}$$

Note that we do *not* identify α-equivalent terms at the level of syntax. The set of free variables of a term, $FV(m)$, is defined in the usual way. For any finite set of variables Δ, we write Λ^{Δ} for the set of λ-terms over Δ, *i.e.*,

$$\Lambda^{\Delta} = \{m \in \Lambda \mid FV(m) \subseteq \Delta\}\,.$$

2.1.2. *Substitutions*

For technical reasons, we take simultaneous (as opposed to single-variable), capture-avoiding substitution as the basic concept. Accordingly, we say that a substitution θ is a finite partial function from variables to terms. We take $FV(\theta) = \bigcup_{x \in \mathrm{dom}\,\theta} FV(\theta(x))$, and define the action of θ on a term m in the usual way, by structural induction on m:

$$\begin{aligned}\mathrm{VAR}(x)[\theta] &= \begin{cases}\theta(x) & \text{if } x \in \mathrm{dom}\,\theta \\ \mathrm{VAR}(x) & \text{otherwise}\end{cases} \\ \mathrm{LAM}(x, m_0)[\theta] &= \mathrm{LAM}(x', m_0[\theta[x \mapsto \mathrm{VAR}(x')]]) \\ &\qquad \text{where } x' \notin FV(\theta) \cup (FV(m_0) \setminus \{x\}) \\ \mathrm{APP}(m_1, m_2)[\theta] &= \mathrm{APP}(m_1[\theta], m_2[\theta])\end{aligned}$$

where $f[a \mapsto b]$ is function extension: $f[a \mapsto b](a') = b$ if $a' = a$, and $f(a')$ otherwise. To keep the substitution operation deterministic, we assume that the x' in the LAM-clause is picked as some arbitrary but fixed function of the (finite) set of variables it needs to avoid. As a special case, we use the standard notation $m[m'/x]$ to mean $m[\,\emptyset[x \mapsto m']\,]$.

2.1.3. *Convertibility and normalization*

We define convertibility between λ-terms, written $m \leftrightarrow m'$, by the axiom schemas for α- and β-conversion,

$$\begin{aligned} \mathrm{LAM}(x, m) &\leftrightarrow \mathrm{LAM}(x', m[x'/x]) \quad (x' \notin FV(m) \setminus \{x\}) \\ \mathrm{APP}(\mathrm{LAM}(x, m), m') &\leftrightarrow m[m'/x], \end{aligned}$$

together with the standard equivalence and compatibility rules, making $\leftrightarrow$ into a congruence relation on terms.

We further define *atomic* (also known as *neutral*) and *normal* forms, as follows:

$$\frac{}{\vdash_{\mathrm{at}} \mathrm{VAR}(x)} \qquad \frac{\vdash_{\mathrm{at}} m_1 \quad \vdash_{\mathrm{nf}} m_2}{\vdash_{\mathrm{at}} \mathrm{APP}(m_1, m_2)} \qquad \frac{\vdash_{\mathrm{at}} m}{\vdash_{\mathrm{nf}} m} \qquad \frac{\vdash_{\mathrm{nf}} m_0}{\vdash_{\mathrm{nf}} \mathrm{LAM}(x, m_0)}.$$

For $s \in \{\mathrm{nf}, \mathrm{at}\}$, we take $\mathcal{N}_s = \{m \mid \vdash_s m\}$, *i.e.*, the set of terms that can be shown to be of syntactic form s by a finite number of rule applications.

We then expect a normalization function on terms to satisfy that the output, if any, is in normal form and convertible to the input (soundness); convertible terms either give the same output, or neither one does (identification); and if a term has a normal form at all, the normalization function will return one (completeness).

2.1.4. *Semantics*

A natural way of defining a denotational model of convertibility is in terms of a reflexive pointed cpo D. Reflexivity means that the continuous-function space $[D \to D]$ is a retract of D, *i.e.*, that there exist continuous functions

$$\phi : [D \to D] \to D \quad \text{and} \quad \psi : D \to [D \to D]\,,$$

with $\psi \circ \phi = id_{[D \to D]}$. The induced interpretation, $[\![m]\!] \in [[\mathrm{V} \to D] \to D]$, is then:

$$\begin{aligned} [\![\mathrm{VAR}(x)]\!]\,\rho &= \rho(x) \\ [\![\mathrm{LAM}(x, m_0)]\!]\,\rho &= \phi(\lambda d^D.\, [\![m_0]\!]\,\rho[x \mapsto d]) \\ [\![\mathrm{APP}(m_1, m_2)]\!]\,\rho &= \psi([\![m_1]\!]\,\rho)\,([\![m_2]\!]\,\rho). \end{aligned}$$

Lemma 1. *The interpretation has two expectable properties:*

(a) *If* $\forall x \in FV(m).\, \rho(x) = \rho'(x)$, *then* $[\![m]\!]\,\rho = [\![m]\!]\,\rho'$.

(b) *Let* $\theta = [x_1 \mapsto m_1, \ldots, x_n \mapsto m_n]$ *be a substitution. Then* $[\![m[\theta]]\!]\,\rho = [\![m]\!]\,\rho[x_1 \mapsto [\![m_1]\!]\,\rho, \ldots, x_n \mapsto [\![m_n]\!]\,\rho]$.

Proof. Part (a) is a straightforward induction on the structure of m. Part (b) follows by induction on the structure of m, using part (a) in the LAM-case. □

Lemma 2 (model soundness)**.** *If* $m \leftrightarrow m'$ *then* $[\![m]\!] = [\![m']\!]$.

Proof. By induction on the derivation of $m \leftrightarrow m'$, using Lemma 1 for α- and β-conversion, and using that $\psi \circ \phi = id_{[D \to D]}$ for β-conversion. □

2.2. Output-term generation

We want to account rigorously for the generation of fresh names, and do so in a modular manner. We will therefore construct a pointed cpo $\widehat{\Lambda}$ (dependent on the name generation scheme) with elements denoted by l, together with continuous *wrapper* functions,

$$\widehat{\mathrm{VAR}} : \mathrm{V} \to \widehat{\Lambda} \qquad \widehat{\mathrm{LAM}} : [\mathrm{V} \to \widehat{\Lambda}] \to \widehat{\Lambda} \qquad \widehat{\mathrm{APP}} : \widehat{\Lambda} \times \widehat{\Lambda} \to \widehat{\Lambda}\,,$$

where, in particular, $\widehat{\mathrm{LAM}}$ provides a fresh name to be used in constructing the body of the λ-abstraction.

Let N be a set (discrete cpo) containing at least the natural numbers, with an operation $\cdot + 1 : \mathrm{N} \to \mathrm{N}$, agreeing with the successor operation on naturals. Let $\mathrm{G} = \{\mathrm{g}_0, \mathrm{g}_1, ...\}$ be a countably infinite subset of V, such that $\mathrm{g}_i = \mathrm{g}_j$ implies $i = j$, and let $\mathrm{gen} : \mathrm{N} \to \mathrm{V}$ be such that $\mathrm{gen}(n) = \mathrm{g}_n$ when $n \in \omega$.

We write $\lfloor\cdot\rfloor$ for the inclusion from A to $A_\perp$; and for $f : A \to B$ with B pointed, we write $\cdot \star f$ for f's strict extension to $A_\perp$, *i.e.*, $\perp \star f = \perp_B$ and $\lfloor a \rfloor \star f = f(a)$. (As is conventional in functional-programming syntax, function application by juxtaposition binds tighter than all explicit infix operators, including $\star$.)

Definition 1. We take $\widehat{\Lambda} = [\mathrm{N} \to \Lambda_\perp]$ and define wrapper functions for constructing λ-terms using de Bruijn-level (not -index!) naming as follows:

$$\begin{aligned}
\widehat{\mathrm{VAR}}(v) &= \lambda n^{\mathrm{N}}.\, \lfloor \mathrm{VAR}(v) \rfloor \\
\widehat{\mathrm{LAM}}(f) &= \lambda n^{\mathrm{N}}.\, f\,(\mathrm{gen}(n))\,(n+1) \star \lambda m_0^{\Lambda}.\, \lfloor \mathrm{LAM}(\mathrm{gen}(n), m_0) \rfloor \\
\widehat{\mathrm{APP}}(l_1, l_2) &= \lambda n^{\mathrm{N}}.\, l_1\, n \star \lambda m_1^{\Lambda}.\, l_2\, n \star \lambda m_2^{\Lambda}.\, \lfloor \mathrm{APP}(m_1, m_2) \rfloor .
\end{aligned}$$

Note 1. *If we took freshness as a primitive concept, like in FreshML, we could simply use* $\widehat{\Lambda} = \Lambda_\perp$*;* $\widehat{\mathrm{VAR}}(v) = \lfloor \mathrm{VAR}(v) \rfloor$*;* $\widehat{\mathrm{LAM}}(f) = f\, x \star \lambda m_0.\, \lfloor \mathrm{LAM}(x, m_0) \rfloor$*, with* x fresh *for* f*; and* $\widehat{\mathrm{APP}}(l_1, l_2) = l_1 \star \lambda m_1.\, l_2 \star \lambda m_2.\, \lfloor \mathrm{APP}(m_1, m_2) \rfloor$*.*

2.3. A residualizing model

From standard domain-theoretic results (*e.g.*, Pitts [11]), we know that there exists a pointed cpo D_{r}, together with an isomorphism

$$i_D : D_{\mathrm{r}} \xrightarrow{\cong} (\widehat{\Lambda} + [D_{\mathrm{r}} \to D_{\mathrm{r}}])_\perp .$$

We write

$$\mathrm{tm}(l) = i_D^{-1}(\lfloor in_1(l) \rfloor) \qquad \mathrm{fun}(f) = i_D^{-1}(\lfloor in_2(f) \rfloor) \qquad \perp_{D_{\mathrm{r}}} = i_D^{-1}(\perp)\,.$$

Then any element of D_{r} can be uniquely written as one of $\mathrm{tm}(l)$, $\mathrm{fun}(f)$, or $\perp_{D_{\mathrm{r}}}$.

Moreover, the standard domain-theoretic solution is in fact a so-called *minimal invariant* [11], which we will exploit in the correctness proof. (In the specific case of D_{r}, the minimal-invariant condition says that the following "copy function" $e : D_{\mathrm{r}} \to D_{\mathrm{r}}$, recursively defined in the least-fixed-point sense,

$$e(d) = \text{case } d \text{ of } \begin{cases} \mathrm{tm}(l) & \to \mathrm{tm}(l) \\ \mathrm{fun}(f) & \to \mathrm{fun}(e \circ f \circ e) \\ \bot_{D_{\mathrm{r}}} & \to \bot_{D_{\mathrm{r}}} \end{cases}$$

is in fact the identity function on D_{r}.)

We can now define the reification function, $\downarrow : D_{\mathrm{r}} \to \widehat{\Lambda}$, and the reflection function, $\uparrow : \widehat{\Lambda} \to D_{\mathrm{r}}$, as follows:

$$\begin{aligned} \downarrow d &= \text{case } d \text{ of } \begin{cases} \mathrm{tm}(l) & \to l \\ \mathrm{fun}(f) & \to \widehat{\mathrm{LAM}}(\lambda x^{\mathrm{V}}.\downarrow(f(\uparrow \widehat{\mathrm{VAR}}(x)))) \\ \bot_{D_{\mathrm{r}}} & \to \bot_{\widehat{\Lambda}} \end{cases} \\ \uparrow l &= \mathrm{tm}(l), \end{aligned}$$

where the recursive definition of $\downarrow$ is again interpreted as the least fixed point. Using these, we construct appropriate functions $\phi_{\mathrm{r}} : [D_{\mathrm{r}} \to D_{\mathrm{r}}] \to D_{\mathrm{r}}$ and $\psi_{\mathrm{r}} : D_{\mathrm{r}} \to [D_{\mathrm{r}} \to D_{\mathrm{r}}]$:

$$\begin{aligned} \phi_{\mathrm{r}}(f) &= \mathrm{fun}(f) \\ \psi_{\mathrm{r}}(d) &= \text{case } d \text{ of } \begin{cases} \mathrm{tm}(l) & \to \lambda d'^{D_{\mathrm{r}}}.\uparrow \widehat{\mathrm{APP}}(l, \downarrow d') \\ \mathrm{fun}(f) & \to f \\ \bot_{D_{\mathrm{r}}} & \to \bot_{[D_{\mathrm{r}} \to D_{\mathrm{r}}]}. \end{cases} \end{aligned}$$

Clearly, we have that $\psi_{\mathrm{r}} \circ \phi_{\mathrm{r}} = id_{[D_{\mathrm{r}} \to D_{\mathrm{r}}]}$, since i_D was an isomorphism. The induced interpretation is denoted by $[\![\cdot]\!]_{\mathrm{r}}$.

We can now define a putative normalization function:

Definition 2. For any Δ, let $\sharp\Delta = \max(\{n+1 \mid \mathrm{g}_n \in \Delta\} \cup \{0\})$ (*i.e.*, the least n such that $\forall n' \geq n.\, \mathrm{g}_{n'} \notin \Delta$). We then define the function $\mathrm{norm}_\Delta : \Lambda^\Delta \to \Lambda_\bot$ by

$$\mathrm{norm}_\Delta(m) \;=\; \downarrow([\![m]\!]_{\mathrm{r}}\,(\lambda x^{\mathrm{V}}.\uparrow \widehat{\mathrm{VAR}}(x)))\;\sharp\Delta\,.$$

We also define the general function $\mathrm{norm} : \Lambda \to \Lambda_\bot$ like above, but with $\sharp\Delta$ replaced by 0. Then for any Δ such that $\Delta \cap \mathrm{G} = \emptyset$, and $m \in \Lambda^\Delta$, $\mathrm{norm}(m) = \mathrm{norm}_\Delta(m)$.

3. Correctness of the construction

3.1. Correctness of the wrappers

We first define what it means for an element of $\widehat{\Lambda}$ to represent a λ-term with some additional properties:

Definition 3. For $l \in \widehat{\Lambda}$, $\Delta \subseteq_{\text{fin}} \mathrm{V}$, $s \in \{\text{at}, \text{nf}\}$, and $m \in \Lambda^{\Delta}$, we define the *representation* relation $\lessapprox$ by

$$l \lessapprox_s^{\Delta} m \quad \text{iff} \quad \forall n \geq \sharp\Delta.\, l\, n = \bot \vee \exists m' \in \Lambda^{\Delta}.\, l\, n = \lfloor m' \rfloor \wedge m' \leftrightarrow m \wedge m' \in \mathcal{N}_s\,.$$

That is, as long as we avoid clashes with generated bound-variable names, any concrete term generated from l has only free variables in Δ, is convertible to m, and is of syntactic form s. Note, however, that we only capture a notion of partial correctness here: if l does not generate a term at all, the conditions are vacuously satisfied.

Lemma 3. *For fixed Δ, s, and m, the predicate $P = \{l \mid l \lessapprox_s^{\Delta} m\}$ is pointed (i.e., $\bot_{\widehat{\Lambda}} \in P$) and inclusive (i.e., closed under limits of ω-chains).*

Proof. Straightforward, noting that $\lessapprox$ is expressed as an intersection of inverse images by a continuous function (application to n) of a (necessarily inclusive) predicate on the flat domain $\Lambda_{\bot}$. □

Lemma 4. *The representation relation is closed under weakening and conversion:*

(a) *If $l \lessapprox_s^{\Delta} m$ and $\Delta \subseteq \Delta'$, then also $l \lessapprox_s^{\Delta'} m$.*

(b) *If $l \lessapprox_s^{\Delta} m$ and $m' \in \Lambda^{\Delta}$ with $m \leftrightarrow m'$, then also $l \lessapprox_s^{\Delta} m'$.*

Proof. Both parts are immediate from the definition. □

Lemma 5. *Representations of terms behave much like the terms themselves:*

(a) *If $v \in \Delta$, then $\widehat{\mathrm{VAR}}(v) \lessapprox_{\text{at}}^{\Delta} \mathrm{VAR}(v)$.*

(b) *If $l_1 \lessapprox_{\text{at}}^{\Delta} m_1$ and $l_2 \lessapprox_{\text{nf}}^{\Delta} m_2$, then $\widehat{\mathrm{APP}}(l_1, l_2) \lessapprox_{\text{at}}^{\Delta} \mathrm{APP}(m_1, m_2)$.*

(c) *If $l \lessapprox_{\text{at}}^{\Delta} m$, then also $l \lessapprox_{\text{nf}}^{\Delta} m$.*

(d) *Let $f \in [\mathrm{V} \to \widehat{\Lambda}]$ and $m \in \Lambda^{\Delta \cup \{x\}}$. If $\forall v \notin \Delta. f v \lessapprox_{\text{nf}}^{\Delta \cup \{v\}} m[\mathrm{VAR}(v)/x]$, then $\widehat{\mathrm{LAM}}(f) \lessapprox_{\text{nf}}^{\Delta} \mathrm{LAM}(x, m)$.*

Proof. All parts are relatively straightforward. (b) and (d) exploit that $\leftrightarrow$ is a congruence relation. For (d), the assumption about m's free variables is also essential. □

The constructions and results in the next section rely only on those properties of the wrappers from Definition 1 and the relation $\lessapprox$ from Definition 3 that were established in Lemmas 3–5, not on the definitions themselves.

3.2. Adequacy of the residualizing model

To construct the central relation between terms and their residualizing denotations, we first state an abstract version of a result due to Pitts [11]:

Theorem 1 (existence of invariant relations)**.** *Let A be a cpo, and let $i : D \xrightarrow{\cong} (A + [D \to D])_\perp$ be a minimal invariant. Let T be a set, and let predicates $P_1 \subseteq A \times T$, $P_2 \subseteq T$, and $P_3 \subseteq T \times T \times T$ be given, such that $\{a \mid P_1(a,t)\}$ is inclusive for every $t \in T$. Then there exists a relation $\lhd \subseteq D \times T$, with $\{d \mid d \lhd t\}$ inclusive for every $t \in T$, such that, for all $d \in D$ and $t \in T$,*

$$d \lhd t \text{ iff } \left(\begin{array}{l} d = \perp_D \vee \\ \exists a.\, d = i^{-1}(\lfloor in_1(a) \rfloor) \wedge P_1(a,t) \vee \\ \exists f.\, d = i^{-1}(\lfloor in_2(f) \rfloor) \wedge P_2(t) \wedge \\ \quad \forall d' \in D; t', t'' \in T.\ P_3(t,t',t'') \wedge d' \lhd t' \Rightarrow f(d') \lhd t'' \end{array} \right).$$

Proof. The proof proceeds almost exactly as in Pitts's paper: we separate positive and negative occurrences of $\lhd$ in its defining equation, then use the Knaster-Tarski fixed-point theorem to construct a pair of relations $\lhd^+$ and $\lhd^-$, bounding the desired one. Finally, using the minimal-invariant property of D, we show that $\lhd^+ = \lhd^-$. □

We can recover Pitts's original result as follows. Let $\Lambda_\mathbb{Z}$ be an extension of Λ with PCF-style integer arithmetic, and let $\Downarrow$ be the usual big-step, call-by-name evaluation relation on $\Lambda^\emptyset_\mathbb{Z}$. We then take A as $\mathbb{Z}$, T as $\Lambda^\emptyset_\mathbb{Z}$, $P_1(n,m)$ as $m \Downarrow \underline{n}$, $P_2(m)$ as $\exists x, m_0.\, m \Downarrow \mathrm{LAM}(x, m_0)$, and $P_3(m,m',m'')$ as $\forall x, m_0.\, m \Downarrow \mathrm{LAM}(x,m_0) \Rightarrow m'' = m_0[m'/x]$. Note that, by determinacy of $\Downarrow$, the x and m_0 are uniquely determined when they exist, so P_2 and P_3 would naturally be joined into a single condition.

The computational-adequacy proof for evaluation then shows, by a straightforward structural induction on m, that if $\rho : \mathrm{V} \to D$ and $\theta : \mathrm{V} \rightharpoonup \Lambda^\emptyset_\mathbb{Z}$ are such that $\forall x \in FV(m).\, \rho(x) \lhd \theta(x)$, then $[\![m]\!]\rho \lhd m[\theta]$. For the special case when m is itself a closed term, we then immediately read off that if $[\![m]\!]\emptyset \neq \perp_D$ then m must evaluate to a value.

When generalizing to normalization, there are two complications. First, we consider symmetric equivalence, not directed evaluation, so the relation $d \lhd -$ must be closed under arbitrary β-conversions, not just head-β-expansions as before. Second, since we also normalize under lambdas, we must in general consider substitutions that replace variables with open terms. Accordingly, we replace the fixed set of closed terms, $\Lambda^\emptyset$, with a Kripke-style family of term sets, indexed by their allowed free variables, Λ^Δ. Somewhat surprisingly, Pitts's result – although in the generalized formulation – accounts directly for these adaptations:

Lemma 6. *There exists a relation $\lesssim$ such that for all Δ, $d \in D_\mathrm{r}$ and $m \in \Lambda^\Delta$,*

$$d \lesssim^\Delta m \text{ iff } \left(\begin{array}{l} d = \bot_{D_\mathrm{r}} \vee \\ \exists l.\, d = \mathrm{tm}(l) \wedge l \lessapprox_{\mathrm{at}}^\Delta m \vee \\ \exists f.\, d = \mathrm{fun}(f) \wedge (\exists x \in \mathrm{V}, m_0 \in \Lambda^{\Delta \cup \{x\}}.\mathrm{LAM}(x, m_0) \leftrightarrow m) \\ \quad \wedge \forall \Delta' \supseteq \Delta, d' \in D_\mathrm{r}, m' \in \Lambda^{\Delta'}, m_1 \in \Lambda^{\Delta'}. \\ \qquad m \leftrightarrow m_1 \wedge d' \lesssim^{\Delta'} m' \Rightarrow f(d') \lesssim^{\Delta'} \mathrm{APP}(m_1, m') \end{array} \right).$$

Proof. By Theorem 1, taking $A = \widehat{\Lambda}$ and $T = \{(\Delta, m) \mid \Delta \subseteq_{\mathrm{fin}} \mathrm{V} \wedge m \in \Lambda^\Delta\}$, with the predicates chosen as

$$\begin{aligned} P_1 &= \{(l, (\Delta, m)) \mid l \lessapprox_{\mathrm{at}}^\Delta m\} \\ P_2 &= \{(\Delta, m) \mid \exists x \in \mathrm{V}, m_0 \in \Lambda^{\Delta \cup \{x\}}.\mathrm{LAM}(x, m_0) \leftrightarrow m\} \\ P_3 &= \{((\Delta, m), (\Delta', m'), (\Delta'', m'')) \mid \\ &\qquad \Delta \subseteq \Delta' = \Delta'' \wedge \exists m_1 \in \Lambda^{\Delta'}.m \leftrightarrow m_1 \wedge m'' = \mathrm{APP}(m_1, m')\} \end{aligned}$$

using the equivalence $[\forall x.(\exists y.P(x,y)) \Rightarrow Q(x)] \Leftrightarrow [\forall x.\forall y.P(x,y) \Rightarrow Q(x)]$. P_1 is inclusive in its first argument by Lemma 3. We write $d \lesssim^\Delta m$ instead of $d \lhd (\Delta, m)$. □

Note how, in the function case, we require that f and m can also be meaningfully applied to arguments from later worlds Δ', much like in a conventional, type-indexed Kripke logical relation [10], p. 590.

Lemma 7. *The relation $\lesssim$ shares two key properties with $\lessapprox$:*

(a) *If $d \lesssim^\Delta m$ and $\Delta \subseteq \Delta'$, then also $d \lesssim^{\Delta'} m$.*

(b) *If $d \lesssim^\Delta m$ and $m' \in \Lambda^\Delta$ with $m \leftrightarrow m'$, then also $d \lesssim^\Delta m'$.*

Proof. We proceed according to the cases for $d \lesssim^\Delta m$ in Lemma 6. Both parts are straightforward, given Lemma 4, and noting the transitivity of $\subseteq$ and $\leftrightarrow$. □

The following two lemmas will combine to establish adequacy of our semantics:

Lemma 8. *For all $l \in \widehat{\Lambda}$, $d \in D_\mathrm{r}$, and $m \in \Lambda^\Delta$,*

(a) *If $l \lessapprox_{\mathrm{at}}^\Delta m$, then $\uparrow l \lesssim^\Delta m$.*

(b) *If $d \lesssim^\Delta m$, then $\downarrow d \lessapprox_{\mathrm{nf}}^\Delta m$.*

Proof. Part (a) follows immediately from the definition of $\uparrow$. Part (b) exploits $\downarrow$'s definition as a least fixed point, and proceeds by fixed-point induction on the pointed and inclusive (by Lem. 3) predicate,

$$R = \{\varphi \in [D_\mathrm{r} \to \widehat{\Lambda}] \mid \forall d, \Delta, m \in \Lambda^\Delta.\, d \lesssim^\Delta m \Rightarrow \varphi(d) \lessapprox_{\mathrm{nf}}^\Delta m\}.$$

The verification is straightforward, given the previous lemmas: the case $d = \mathrm{tm}(l)$ is immediate by Lemma 5c, while $d = \mathrm{fun}(f)$ uses Lemma 5(a,d), and that both $\lessapprox$ and $\lesssim$ are closed under conversion (Lems. 4b and 7b). □

Lemma 9. *Let $m \in \Lambda^\Gamma$, and for all $x \in \Gamma$, let $\theta(x) \in \Lambda^\Delta$ (in particular, $\Gamma \subseteq \operatorname{dom}\theta$). If $\forall x \in \Gamma.\, \rho(x) \lesssim^\Delta \theta(x)$ then $[\![m]\!]_\mathrm{r}\,\rho \lesssim^\Delta m[\theta]$.*

Proof. By structural induction on m. The case for variables is immediate. For abstractions, like in a standard Kripke-logical-relations proof, monotonicity of $\lesssim$ (Lem. 7a) ensures that the environment and substitution remain related in the later world Δ'; also, closure under conversion (Lem. 7b) in particular implies closure under β-expansion. Both parts of Lemma 8, as well as Lemma 5b, are used in the non-standard subcase for applications. □

3.3. Correctness of the normalization function

For showing completeness of the normalizer, we first establish that, for a term already in normal form, reifying its residualizing denotation gives an always-defined term generator.

Definition 4. For any $l \in \widehat{\Lambda}$, we define the *uniform definedness* predicate $\mathrm{def}(l)$ by $\mathrm{def}(l) \Leftrightarrow \forall n \in \omega.\, l\,n \neq \bot$.

Lemma 10. *The wrapper functions preserve definedness:*

(a) *For all $v \in \mathrm{V}$, $\mathrm{def}(\widehat{\mathrm{VAR}}(v))$.*

(b) *If for all $v \in \mathrm{V}$, $\mathrm{def}(f\,v)$, then $\mathrm{def}(\widehat{\mathrm{LAM}}(f))$.*

(c) *If $\mathrm{def}(l_1)$ and $\mathrm{def}(l_2)$, then $\mathrm{def}(\widehat{\mathrm{APP}}(l_1, l_2))$.*

Proof. Straightforward verification in each case. □

Lemma 11. *Let $m \in \Lambda$ and $\rho \in [\mathrm{V} \to D_\mathrm{r}]$ be such that for all $x \in FV(m)$, there exists an l with $\rho(x) = \uparrow l$ and $\mathrm{def}(l)$. Then,*

(a) *If $m \in \mathcal{N}_\mathrm{at}$, then $\exists l \in \widehat{\Lambda}.\, [\![m]\!]_\mathrm{r}\,\rho = \uparrow l \wedge \mathrm{def}(l)$.*

(b) *If $m \in \mathcal{N}_\mathrm{nf}$, then $\mathrm{def}(\downarrow([\![m]\!]_\mathrm{r}\,\rho))$.*

Proof. By simultaneous rule induction on $\vdash_\mathrm{at} \cdot$ and $\vdash_\mathrm{nf} \cdot$, relying on Lemma 10. □

Theorem 2 (semantic correctness)**.** norm_Δ *from Definition 2 is a normalization function on Λ^Δ, i.e.,*

(a) *(soundness) If $\mathrm{norm}_\Delta(m) = \lfloor m' \rfloor$, then $m' \in \Lambda^\Delta$, $m' \leftrightarrow m$, and $m' \in \mathcal{N}_\mathrm{nf}$.*

(b) *(identification) If $m \leftrightarrow m'$, then $\mathrm{norm}_\Delta(m) = \mathrm{norm}_\Delta(m')$.*

(c) *(completeness) If for some $m' \in \mathcal{N}_\mathrm{nf}$, $m' \leftrightarrow m$, then $\mathrm{norm}_\Delta(m) \neq \bot$.*

Proof. (Soundness) Let $\rho_0 = \lambda x^\mathrm{V}.\uparrow \widehat{\mathrm{VAR}}(x)$, and let θ_0 be the substitution mapping every x in Δ to $\mathrm{VAR}(x)$. By Lemma 5a, for every $x \in \Delta$, $\widehat{\mathrm{VAR}}(x) \lesssim^\Delta_\mathrm{at} \mathrm{VAR}(x) = \theta_0(x)$, and hence by Lemma 8a, $\rho_0(x) \lesssim^\Delta \theta_0(x)$. By Lemma 9, we then get that $[\![m]\!]_\mathrm{r}\,\rho_0 \lesssim^\Delta m[\theta_0] \leftrightarrow m$, and thus, by Lemma 8b, $\downarrow([\![m]\!]_\mathrm{r}\,\rho_0) \lesssim^\Delta_\mathrm{nf} m$. Assume now that $\mathrm{norm}_\Delta(m) = \downarrow([\![m]\!]_\mathrm{r}\,\rho_0)\,\sharp\Delta = \lfloor m' \rfloor$. Taking $n = \sharp\Delta$ in Definition 3, we can then immediately read off that m' has the required properties.

(Identification) This follows directly from model soundness (Lem. 2), since the residualizing model is indeed a model.

(Completeness) Using Lemma 10a, we see that ρ_0 satisfies the condition on ρ in Lemma 11. Hence, by part (b) of the latter lemma and Definition 4, $\downarrow([\![m']\!]_{\mathrm{r}}\,\rho_0)\,\sharp\Delta \neq \bot$. And thus, again by model soundness, also $\mathrm{norm}_\Delta(m) = \downarrow([\![m]\!]_{\mathrm{r}}\,\rho_0)\,\sharp\Delta \neq \bot$. □

Note that the correctness theorem does not completely pin down the behavior of the normalizer: the soundness specification allows it to return *any* valid α-variant of the normal form, including normalizing $\mathrm{LAM}(\mathrm{x}, \mathrm{LAM}(\mathrm{y}, \mathrm{VAR}(\mathrm{y})))$ to $\mathrm{LAM}(\mathrm{g}_0, \mathrm{LAM}(\mathrm{g}_0, \mathrm{VAR}(\mathrm{g}_0)))$. Conversely, completeness says only that if a term has a β-normal form, the normalizer will also find one, though not necessarily the same one.

It would also be possible to adopt a "tight" specification of normal forms, requiring them to also be α-normal, such as the current de Bruijn-level naming. Then, a term can have at most one normal form, and the normalizer will in fact find exactly that one when it exists – which would allow us to combine soundness and completeness into a single statement.

4. An implementation of the construction

4.1. Syntax and semantics of an ML-like call-by-value language

As our implementation language, we take a small fragment of Standard ML [9]. We deliberately choose an eager language, whose finer control over lifting allows us to mirror all the semantic constructions almost exactly (*i.e.*, up to isomorphism). Any necessary laziness can be easily added by explicit thunking. On the other hand, working in an inherently lazy language, such as Haskell, would make it harder to work with, *e.g.*, the set of λ-terms as a datatype, without also including spurious infinite and partially-defined elements.

4.1.1. *Syntax*

The fragment is parameterized by a sequence of recursive datatype declarations, each of the form

$$\texttt{datatype}\ \mathit{dt}^i\ \texttt{=}\ \mathit{In}^i_1\ \texttt{of}\ \tau^i_{11}\ \texttt{*}\ \cdots\ \texttt{*}\ \tau^i_{1n_1}\ \texttt{|}\ \cdots\ \texttt{|}\ \mathit{In}^i_k\ \texttt{of}\ \tau^i_{k1}\ \texttt{*}\ \cdots\ \texttt{*}\ \tau^i_{kn_k}\,,$$

where ML types τ are given by the grammar,

$$\tau ::= \texttt{unit} \mid \texttt{int} \mid \texttt{bool} \mid \texttt{string} \mid \tau_1\ \texttt{->}\ \tau_2 \mid \mathit{dt}^i\,.$$

The datatypes cannot be mutually recursive, but may be cumulative, *i.e.*, later declarations may refer to earlier ones. We say that a type is *ground* if it does not contain – directly, or indirectly (through a datatype declaration) – any function spaces. For notational simplicity, we assume that the set of λ-term variable names, V, is identified with the set of ML character strings.

The syntax of ML expressions is then

$$\begin{aligned} e ::= {} & x \mid \underline{n} \mid \texttt{"}v\texttt{"} \mid \texttt{()} \mid e_1 \texttt{ + } e_2 \mid e_1 \texttt{ = } e_2 \mid \texttt{"g"\^{}Int.toString } e \mid \\ & \texttt{fn () => } e \mid \texttt{fn } x \texttt{ => } e \mid e_1\ e_2 \mid \mathit{In}^i_j\texttt{(}e_1\texttt{,...,}e_n\texttt{)} \mid \\ & \texttt{case } e \texttt{ of } \mathit{In}^i_1\texttt{(}x_{11}\texttt{,...,}x_{1n_1}\texttt{) => } e_1 \texttt{ | } \cdots \texttt{ | } \mathit{In}^i_k\texttt{(}x_{k1}\texttt{,...,}x_{kn_k}\texttt{) => } e_k \\ & \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \mid \texttt{let fun } f\ \texttt{(}x\texttt{:}\tau_1\texttt{):}\tau_2 \texttt{ = } e_1 \texttt{ in } e_2 \texttt{ end} \end{aligned}$$

where x and f range over ML variable names.

4.1.2. *Typing*

We only consider well-typed ML expressions, as captured by the judgement $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \tau$, asserting that e is of type τ, with free variables $x_1, \ldots, x_n$ of types $\tau_1, \ldots, \tau_n$. It is defined in the usual way by inference rules. The only rule worth remarking on, is that `=` is restricted to comparing values of ground types.

4.1.3. *Denotational semantics*

For the meaning of ML types, we take

$$[\![\texttt{unit}]\!]^{\mathrm{ml}} = \mathbf{1} = \{*\} \qquad [\![\texttt{int}]\!]^{\mathrm{ml}} = \mathbb{Z} \qquad [\![\texttt{bool}]\!]^{\mathrm{ml}} = \mathbb{B} \qquad [\![\texttt{string}]\!]^{\mathrm{ml}} = \mathrm{V}$$
$$[\![\tau_1 \texttt{ -> } \tau_2]\!]^{\mathrm{ml}} = [[\![\tau_1]\!]^{\mathrm{ml}} \to [\![\tau_2]\!]^{\mathrm{ml}}_\perp] \qquad [\![\mathit{dt}^i]\!]^{\mathrm{ml}} = S^i,$$

where, for each dt^i,

$$i_{\mathit{dt}^i} : S^i \xrightarrow{\cong} ([\![\tau^i_{11}]\!]^{\mathrm{ml}} \times \cdots \times [\![\tau^i_{1n_1}]\!]^{\mathrm{ml}}) + \cdots + ([\![\tau^i_{k1}]\!]^{\mathrm{ml}} \times \cdots \times [\![\tau^i_{kn_k}]\!]^{\mathrm{ml}})$$

is a minimal-invariant solution to the evident predomain equation. We write

$$\iota_{\mathit{In}^i_j}(a_1, \ldots, a_n) = i^{-1}_{\mathit{dt}^i}(in_j(a_1, \ldots, a_n))$$

for the constructor functions. Any element of S^i can thus be uniquely written as the image of a constructor function.

When all the $\tau^i_{\cdot\cdot}$ (and hence also dt^i) are ground, the least solution S^i will again be a set (discrete cpo), and could be constructed using standard set-theoretic methods. In the general case, one must use, *e.g.*, the domain-theoretic inverse-limit construction, straightforwardly adapted to predomains.

The meaning of ML expressions is defined by induction on the typing derivation; for conciseness we write only the expressions themselves. The semantics is structured such that if $\Gamma \vdash e : \tau$ and for all $(x_i : \tau_i) \in \Gamma$, $\xi(x_i) \in [\![\tau_i]\!]^{\mathrm{ml}}$, then $[\![e]\!]^{\mathrm{ml}}\,\xi \in [\![\tau]\!]^{\mathrm{ml}}_\perp$. The full semantics is shown in Figure 1.

For notational convenience in the following, we will assume that all function names f in a program are distinct. For a fixed program, we can then unambiguously use $\theta_f = \mathrm{fix}(\Theta_f)$ to refer to the denotation of f in the `let fun`-construct.

$$[\![x]\!]^{\mathrm{ml}}\,\xi = \lfloor \xi(x) \rfloor \qquad [\![\underline{n}]\!]^{\mathrm{ml}}\,\xi = \lfloor n \rfloor \qquad [\![\texttt{"}v\texttt{"}]\!]^{\mathrm{ml}}\,\xi = \lfloor v \rfloor \qquad [\![\texttt{()}]\!]^{\mathrm{ml}}\,\xi = \lfloor * \rfloor$$

$$[\![e_1 \texttt{ + } e_2]\!]^{\mathrm{ml}}\,\xi = [\![e_1]\!]^{\mathrm{ml}}\,\xi \star \lambda n_1^{\mathbb{Z}}.\, [\![e_2]\!]^{\mathrm{ml}}\,\xi \star \lambda n_2^{\mathbb{Z}}.\, \lfloor n_1 + n_2 \rfloor$$

$$[\![e_1 \texttt{ = } e_2]\!]^{\mathrm{ml}}\,\xi = [\![e_1]\!]^{\mathrm{ml}}\,\xi \star \lambda a_1^{[\![\tau]\!]^{\mathrm{ml}}}.\, [\![e_2]\!]^{\mathrm{ml}}\,\xi \star \lambda a_2^{[\![\tau]\!]^{\mathrm{ml}}}.\, \lfloor a_1 = a_2 \rfloor$$

$$[\![\texttt{"g"\^{}Int.toString } e]\!]^{\mathrm{ml}}\,\xi = [\![e]\!]^{\mathrm{ml}}\,\xi \star \lambda n^{\mathbb{Z}}.\, \lfloor \underline{\mathrm{g}n} \rfloor$$

$$[\![\texttt{fn () => } e]\!]^{\mathrm{ml}}\,\xi = \lfloor \lambda u^{\mathbf{1}}.[\![e]\!]^{\mathrm{ml}}\,\xi \rfloor \qquad [\![\texttt{fn } x \texttt{ => } e]\!]^{\mathrm{ml}}\,\xi = \lfloor \lambda a^{[\![\tau_1]\!]^{\mathrm{ml}}}.[\![e]\!]^{\mathrm{ml}}\,\xi[x \mapsto a] \rfloor$$

$$[\![e_1\ e_2]\!]^{\mathrm{ml}}\,\xi = [\![e_1]\!]^{\mathrm{ml}}\,\xi \star \lambda f^{[[\![\tau_1]\!]^{\mathrm{ml}} \to [\![\tau_2]\!]^{\mathrm{ml}}_\perp]}.\, [\![e_2]\!]^{\mathrm{ml}}\,\xi \star \lambda a^{[\![\tau_1]\!]^{\mathrm{ml}}}.\, f\,a$$

$$[\![\mathit{In}_j^i(e_1, \ldots, e_n)]\!]^{\mathrm{ml}}\,\xi = [\![e_1]\!]^{\mathrm{ml}}\,\xi \star \lambda a_1^{[\![\tau^i_{j1}]\!]^{\mathrm{ml}}}.\, \cdots .[\![e_n]\!]^{\mathrm{ml}}\,\xi \star \lambda a_n^{[\![\tau^i_{jn}]\!]^{\mathrm{ml}}}.\, \lfloor \iota_{\mathit{In}_j^i}(a_1, \ldots, a_n) \rfloor$$

$$[\![\texttt{case } e \texttt{ of } \mathit{In}_1^i(x_{11}, \ldots, x_{1n_1}) \texttt{ => } e_1 \texttt{ | } \cdots \texttt{ | } \mathit{In}_k^i(x_{k1}, \ldots, x_{kn_k}) \texttt{ => } e_k]\!]^{\mathrm{ml}}\,\xi =$$

$$[\![e]\!]^{\mathrm{ml}}\,\xi \star \lambda s^{S^i}.\, \mathrm{case}\ s\ \mathrm{of} \begin{cases} \iota_{\mathit{In}_1^i}(a_1, \ldots, a_{n_1}) \to [\![e_1]\!]^{\mathrm{ml}}\,\xi[x_{11} \mapsto a_1, \ldots, x_{1n_1} \mapsto a_{n_1}] \\ \qquad\qquad \vdots \\ \iota_{\mathit{In}_k^i}(a_1, \ldots, a_{n_k}) \to [\![e_k]\!]^{\mathrm{ml}}\,\xi[x_{k1} \mapsto a_1, \ldots, x_{kn_k} \mapsto a_{n_k}] \end{cases}$$

$$[\![\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3]\!]^{\mathrm{ml}}\,\xi = [\![e_1]\!]^{\mathrm{ml}}\,\xi \star \lambda b^{\mathbb{B}}.\, \mathrm{case}\ b\ \mathrm{of} \begin{cases} \mathrm{tt} \to [\![e_2]\!]^{\mathrm{ml}}\,\xi \\ \mathrm{ff} \to [\![e_3]\!]^{\mathrm{ml}}\,\xi \end{cases}$$

$$[\![\texttt{let fun } f\ \texttt{(}x\texttt{:}\tau_1\texttt{):}\tau_2 \texttt{ = } e_1 \texttt{ in } e_2 \texttt{ end}]\!]^{\mathrm{ml}}\,\xi = [\![e_2]\!]^{\mathrm{ml}}\,\xi[f \mapsto \mathrm{fix}(\underbrace{\lambda\theta^{[[\![\tau_1]\!]^{\mathrm{ml}} \to [\![\tau_2]\!]^{\mathrm{ml}}_\perp]}.\lambda a^{[\![\tau_1]\!]^{\mathrm{ml}}}.[\![e_1]\!]^{\mathrm{ml}}\,\xi[f \mapsto \theta, x \mapsto a]}_{\Theta_f})]$$

FIGURE 1. Denotational semantics of the ML fragment.

4.1.4. *Evaluation semantics*

We say that a *complete program* is a closed expression of type τ_1 `->` τ_2 `->` $\cdots$ `->` τ_n `->` τ_0 ($n \geq 0$), where each τ_i is a ground type. For such types, let $C_\tau = [\![\tau]\!]^{\mathrm{ml}}$ denote the set of values underlying τ, *e.g.*, $C_{\texttt{int}} = \mathbb{Z}$. A complete program $e : \tau_1$ `->` τ_2 `->` $\cdots$ `->` τ_n `->` τ_0 then determines a computable partial function $e \bullet (\cdot) : C_{\tau_1} \times \cdots \times C_{\tau_n} \rightharpoonup C_{\tau_0}$, given, *e.g.*, by

$$e \bullet (c_1, \ldots, c_n) = c_0 \ \text{ iff } \ (e\ \underline{c_1}\ \cdots\ \underline{c_n}) \Downarrow^{\mathrm{ml}} \underline{c_0}\,,$$

where $\Downarrow^{\mathrm{ml}}$ is the usual big-step operational semantics of expressions, and $\underline{c}$ denotes the syntactic representation of the value c.

Theorem 3 (computational adequacy of denotational semantics)**.** *For a complete ML program e, $e \bullet (c_1, \ldots, c_n) = c_0$ iff $[\![e]\!]^{\mathrm{ml}}\emptyset \star \lambda f_1.\, f_1\, c_1 \star \cdots \star \lambda f_n.\, f_n\, c_n = \lfloor c_0 \rfloor$.*

Proof. Modulo trivial syntactic differences, an equivalent formulation of the semantics in terms of strict functions between pointed cpos, rather than general

```
datatype term = VAR of string | LAM of string*term | APP of term*term
datatype sem = TM of int -> term | FUN of (unit -> sem) -> sem;

let fun down (s:sem):int->term = fn n =>
     (case s of
        TM l => l n
      | FUN f => LAM("g"^Int.toString n,
          down (f (fn () => TM(fn n' => VAR("g"^Int.toString n)))) (n+1)))
in let fun eval (m:term):(string->sem)->sem = fn p =>
        (case m of
           VAR x => p x
         | LAM(x,m0) => FUN(fn d => eval m0
                                (fn x' => if x = x' then d () else p x'))
         | APP(m1,m2) => (case (eval m1 p) of
                            TM l => TM(fn n => APP(l n,down (eval m2 p) n))
                          | FUN f => f (fn () => eval m2 p)))
   in let fun norm (m:term):term =
            down (eval m (fn x => TM(fn n => VAR(x)))) 0
      in norm end end end
```

FIGURE 2. The normalization algorithm, *NORM*.

ones between cpos, and the obvious generalization to multiple (non-mutually recursive) datatypes, this is shown in, *e.g.*, [12], Section 5. The primary difficulty is, of course, the definition of the logical relation at types dt^i, which is again achieved by exploiting the minimal-invariant properties of the (S^i, i_{dt^i}). □

4.2. The normalization algorithm

The concrete representation of the normalization algorithm, with many of the auxiliary definitions inlined, is shown in Figure 2. We have taken $\mathit{dt}^1 = \mathtt{term}$ with three constructors $\mathit{In}_1^1 = \mathtt{VAR}$, $\mathit{In}_2^1 = \mathtt{LAM}$, and $\mathit{In}_3^1 = \mathtt{APP}$. Note that `term` is a ground type. To keep the notation concise, we assume that the chosen predomain-equation solution coincides exactly with our representation of terms, *i.e.*, $[\![\mathtt{term}]\!]^{\mathrm{ml}} = \Lambda$, $\iota_{\mathtt{LAM}}(v, m) = \mathrm{LAM}(v, m)$, etc. Similarly, we have instantiated dt^2 as the type `sem`, with constructors `TM` and `FUN`. We write $S = [\![\mathtt{sem}]\!]^{\mathrm{ml}}$, but here we do not *a priori* require that $S_\perp = D_{\mathrm{r}}$.

Since ML is a call-by-value language, we must simulate the implicit call-by-name nature of the residualizing semantics using thunking. We have defined `sem` so that $[\![\mathtt{sem}]\!]^{\mathrm{ml}}_\perp \cong D_{\mathrm{r}}$; then semantic functions with codomain D_{r} can be represented directly as ML functions into `sem`, while functions with domain D_{r} are represented as ML functions with source type `unit -> sem`. As an optimization, however, the *strict* function $\downarrow : D_{\mathrm{r}} \to \widehat{\Lambda}$ is represented as simply an ML function on `sem`.

It is easy to check that the top-level expression, *NORM* : `term -> term`, is a well-typed complete program in our sense.

Examples. The following examples illustrate how the program behaves. Let $SA \equiv \mathrm{LAM}(\mathrm{x}, \mathrm{APP}(\mathrm{VAR}(\mathrm{x}), \mathrm{VAR}(\mathrm{x})))$ and $Omega \equiv \mathrm{APP}(SA, SA)$.

(a) $\mathtt{NORM} \bullet (Omega)$ diverges.

(b) $\mathtt{NORM} \bullet (\mathrm{APP}(\mathrm{LAM}(\mathrm{x}, \mathrm{LAM}(\mathrm{x}, \mathrm{VAR}(\mathrm{x}))), Omega)) = \mathrm{LAM}(\mathrm{g}_0, \mathrm{VAR}(\mathrm{g}_0))$.

(c) $\mathtt{NORM} \bullet (\mathrm{LAM}(\mathrm{y}, \mathrm{LAM}(\mathrm{g}_4, \mathrm{VAR}(\mathrm{z})))) = \mathrm{LAM}(\mathrm{g}_0, \mathrm{LAM}(\mathrm{g}_1, \mathrm{VAR}(\mathrm{z})))$.

Let us now formally relate the abstract and concrete constructions. To obtain a perfect correspondence between semantic term generators and their implementation, we choose $\mathrm{N} = \mathbb{Z}$, with $\mathrm{gen}(n) = \underline{\mathtt{g}n} = \mathrm{g}_n$ when $n \geq 0$, *e.g.*, $\mathrm{gen}(13) = \mathtt{g13}$; then $[\![\mathtt{int\ ->\ term}]\!]^{\mathrm{ml}} = \widehat{\Lambda}$. Recall that we had $D_\mathrm{r} \cong (\widehat{\Lambda} + [D_\mathrm{r} \to D_\mathrm{r}])_\perp$, while $[\![\mathtt{sem}]\!]^{\mathrm{ml}} = S \cong \widehat{\Lambda} + [[\mathbf{1} \to S_\perp] \to S_\perp]$. We can then show:

Lemma 12. *There exists an isomorphism $i_{DS} : D_\mathrm{r} \xrightarrow{\cong} S_\perp$, satisfying*

(a) *For all $l \in \widehat{\Lambda}$, $i_{DS}(\mathrm{tm}(l)) = \lfloor \iota_{\mathtt{TM}}(l) \rfloor$.*

(b) *For all $f \in [D_\mathrm{r} \to D_\mathrm{r}]$, $i_{DS}(\mathrm{fun}(f)) = \lfloor \iota_{\mathtt{FUN}}(\lambda t^{[\mathbf{1} \to S_\perp]}.\, i_{DS}(f(i_{DS}^{-1}(t\ *)))) \rfloor$.*

(c) *$i_{DS}(\perp_{D_\mathrm{r}}) = \perp_{S_\perp}$.*

Proof. The function pair (i_{DS}, i_{DS}^{-1}) is constructed by mutual recursion (as usual, interpreted as a least-fixed-point construction):

$$i_{DS}(d) = \mathrm{case}\ d\ \mathrm{of} \begin{cases} \mathrm{tm}(l) & \to \lfloor \iota_{\mathtt{TM}}(l) \rfloor \\ \mathrm{fun}(f) & \to \lfloor \iota_{\mathtt{FUN}}(\lambda t^{[1 \to S_\perp]}.\, i_{DS}(f(i_{DS}^{-1}(t\,*)))) \rfloor \\ \perp_{D_\mathrm{r}} & \to \perp_{S_\perp} \end{cases}$$

$$i_{DS}^{-1}(s') = s' \star \lambda s.\, \mathrm{case}\ s\ \mathrm{of} \begin{cases} \iota_{\mathtt{TM}}(l) & \to \mathrm{tm}(l) \\ \iota_{\mathtt{FUN}}(g) & \to \mathrm{fun}(\lambda d. i_{DS}^{-1}(g(\lambda u^{\mathbf{1}}. i_{DS}(d)))). \end{cases}$$

That i_{DS} and i_{DS}^{-1} are actually two-sided inverses, follows from the minimal-invariant characterizations of D_r and S. Properties (a–c) can then be read off directly from the defining equation for i_{DS}. □

We can also state three lemmas, relating the central domain-theoretic functions to the denotations of their syntactic counterparts:

Lemma 13. *For all $d \in D_\mathrm{r}$ and $n \in \mathbb{Z}$, $\downarrow d\, n = i_{DS}(d) \star \lambda s^S.\, \theta_{\mathtt{down}}\ s \star \lambda l^{\widehat{\Lambda}}.\, l\, n$.*

Proof. By simultaneous fixed-point induction wrt. the relation $R = \{(\varphi, \theta) \in [D_\mathrm{r} \to \widehat{\Lambda}] \times [S \to \widehat{\Lambda}_\perp] \mid \forall d \in D_\mathrm{r}, n \in \mathbb{Z}.\, \varphi\ d\ n = i_{DS}(d) \star \lambda s^S.\theta\ s \star \lambda l^{\widehat{\Lambda}}.l\ n\}$. The inductive step proceeds by analysis of the three cases for d; they all follow straightforwardly, using Lemma 12. □

Lemma 14. *For all $m \in \Lambda$, $\rho \in [\mathrm{V} \to D_\mathrm{r}]$, and $\varrho \in [\mathrm{V} \to S_\perp]$, satisfying that $\forall x \in FV(m).\, i_{DS}(\rho(x)) = \varrho(x)$, $i_{DS}([\![m]\!]_\mathrm{r}\, \rho) = \theta_{\mathtt{eval}}\, m \star \lambda g.\, g\, \varrho$.*

Proof. By structural induction on m, using the fixed-point equation for $\mathrm{fix}(\Theta_{\mathtt{eval}})$ in the inductive steps, Lemma 12 throughout, and Lemma 13 in the non-standard subcase for application. □

Lemma 15. *For all* $m \in \Lambda$, $\mathrm{norm}(m) = \theta_{\mathtt{norm}}\ m$.

Proof. Follows easily from the definition of $\theta_{\mathtt{norm}}$, using Lemmas 12–14. □

Theorem 4 (implementation correctness)**.** *The program* ***NORM*** *satisfies that for all* $m, m' \in \Lambda$, ***NORM*** $\bullet\ (m) = m' \Leftrightarrow \mathrm{norm}(m) = \lfloor m' \rfloor$.

Proof. A direct consequence of Lemma 15 and Theorem 3, since $[\![\textbf{\textit{NORM}}]\!]^{\mathrm{ml}}\,\emptyset = \lfloor \theta_{\mathtt{norm}} \rfloor$. □

Together with semantic correctness (Th. 2) and Definition 2 of norm, this tells us that ***NORM*** correctly computes the normal form of all λ-terms without free occurrences of $\mathtt{g}\underline{i}$-variables (including, in particular, all closed terms).

5. A generalization to Böhm trees

Recall that the correctness of our normalization algorithm was expressed in terms of simple conditionals. Soundness was, essentially, "if the algorithm returns a result, that result is correct"; and completeness, "if a correct result exists, the algorithm will find one". We now set out to extend these statements to a more general notion of normal forms, effectively replacing "if" with "to the extent that".

5.1. From λ-terms to λ-trees

We adopt a new view of normalization results, generalizing the flat domain $\Lambda_\perp$ of lifted λ-terms to a more elaborate domain of lazy λ-trees, which will allow us to talk formally about partial and infinite terms. The intended reading of a λ-tree result is that the finitely reachable, defined parts of the tree represent committed output from the normalizer.

5.1.1. *Syntax*

Let $\underline{\Lambda}$ be the cpo of *λ-trees* M, defined as a minimal-invariant solution to the recursive domain equation

$$i_{\underline{\Lambda}} : \underline{\Lambda} \xrightarrow{\cong} (\mathrm{V} + \mathrm{V} \times \underline{\Lambda} + \underline{\Lambda} \times \underline{\Lambda})_\perp\,,$$

with the constructors for $\underline{\Lambda}$-elements given by:

$$\begin{array}{rlrl} \Box &= i_{\underline{\Lambda}}^{-1}(\perp) & \underline{\mathrm{LAM}}(x, M_0) &= i_{\underline{\Lambda}}^{-1}(\lfloor in_2(x, M_0) \rfloor) \\ \underline{\mathrm{VAR}}(x) &= i_{\underline{\Lambda}}^{-1}(\lfloor in_1(x) \rfloor) & \underline{\mathrm{APP}}(M_1, M_2) &= i_{\underline{\Lambda}}^{-1}(\lfloor in_3(M_1, M_2) \rfloor). \end{array}$$

Again, any element of $\underline{\Lambda}$ can be uniquely written as one of these four forms.

We also have a natural interpretation of the domain-theoretic ordering on $\underline{\Lambda}$: $M \sqsubseteq M'$ precisely when M' can be obtained by replacing some occurrences of $\Box$ in M with suitable subtrees. Note that, since $\underline{\Lambda}$ is a cpo, it necessarily also contains infinite trees, such as $\bigsqcup_{n\in\omega} \underline{\mathrm{LAM}}(\mathrm{x}_1, \cdots \underline{\mathrm{LAM}}(\mathrm{x}_n, \Box)) = \underline{\mathrm{LAM}}(\mathrm{x}_1, \underline{\mathrm{LAM}}(\mathrm{x}_2, \cdots))$.

We define the *cut* function $|\cdot|^{\cdot} : \underline{\Lambda} \times \omega \to \underline{\Lambda}$ by induction on k:

$$|M|^0 = \Box \qquad |\Box|^{k+1} = \Box \qquad |\underline{\mathrm{VAR}}(x)|^{k+1} = \underline{\mathrm{VAR}}(x)$$
$$|\underline{\mathrm{LAM}}(x, M_0)|^{k+1} = \underline{\mathrm{LAM}}(x, |M_0|^k) \qquad |\underline{\mathrm{APP}}(M_1, M_2)|^{k+1} = \underline{\mathrm{APP}}(|M_1|^k, |M_2|^k).$$

That is, $|M|^k$ replaces all parts of the tree M above height k with $\Box$. As we would expect, every tree is the limit of its finite cuts:

Lemma 16. *For any $M \in \underline{\Lambda}$, $M = \bigsqcup_{k\in\omega} |M|^k$.*

Proof. Follows directly from the observation that $|M|^k$ is precisely the k'th approximant of the recursively defined "copy function" appearing in the minimal-invariant characterization of $\underline{\Lambda}$. □

A tree is called *finite* if it is equal to one of its cuts, and *total* if it contains no $\Box$. Thus, finite, total λ-trees are in one-to-one correspondence with ordinary λ-terms, as previously defined. We also have a natural inclusion of ordinary λ-terms into λ-trees, $\langle\cdot\rangle : \Lambda \to \underline{\Lambda}$, defined inductively in the obvious way.

5.1.2. *Compatibility*

We can extend any predicate on λ-terms, $P \subseteq \Lambda$, to a corresponding predicate on λ-trees, $P^\dagger \subseteq \underline{\Lambda}$, by

$$P^\dagger = \{M \in \underline{\Lambda} \mid \forall k \in \omega.\, \exists m \in P.\, |M|^k \sqsubseteq \langle m\rangle\}\,.$$

That is, $M \in P^\dagger$ if every finite cut of M can be increased to a total tree, satisfying the original predicate. When a tree already represents a proper term, the extension has no effect: $\langle m\rangle \in P^\dagger$ iff $m \in P$. We also note that $P^\dagger$ is downward closed: if $M \in P^\dagger$ and $M' \sqsubseteq M$, then also $M' \in P^\dagger$; and that extension is monotone: if $P \subseteq P'$, then $P^\dagger \subseteq P'^\dagger$.

Definition 5. For $M \in \underline{\Lambda}$ and $m \in \Lambda^\Delta$, we define the *compatibility* relations $\leftrightarrow^\dagger$ and $\leftrightarrow^\dagger_\Delta$ by

$$M \leftrightarrow^\dagger m \Leftrightarrow M \in \{m' \in \Lambda \mid m' \leftrightarrow m\}^\dagger$$
$$M \leftrightarrow^\dagger_\Delta m \Leftrightarrow M \in \{m' \in \Lambda^\Delta \mid m' \leftrightarrow m\}^\dagger.$$

Note that, like convertibility, compatibility is defined with respect to concrete terms, not α-equivalence classes. Thus,

$$\underline{\mathrm{LAM}}(\mathrm{g}_0, \underline{\mathrm{LAM}}(\mathrm{g}_0, \Box)) \leftrightarrow^\dagger \mathrm{LAM}(\mathrm{x}, \mathrm{LAM}(\mathrm{y}, \mathrm{VAR}(\mathrm{y})))$$

(because the $\Box$ can still be increased to $\underline{\mathrm{VAR}}(\mathrm{g}_0)$, making the two sides convertible), but we do *not* have

$$\underline{\mathrm{LAM}}(\mathrm{g}_0, \underline{\mathrm{LAM}}(\mathrm{g}_0, \Box)) \leftrightarrow^\dagger \mathrm{LAM}(\mathrm{x}, \mathrm{LAM}(\mathrm{y}, \mathrm{VAR}(\mathrm{x})))\,.$$

5.1.3. *Böhm trees*

We can view Böhm trees [2], Chapter 10, as a particular kind of λ-trees. Informally, a Böhm tree is either $\Box$, or a generalized head normal form,

$$\underline{\mathrm{LAM}}(x_1, \cdots \underline{\mathrm{LAM}}(x_n, \underline{\mathrm{APP}}(\underline{\mathrm{APP}}(\underline{\mathrm{VAR}}(x), M_1), \cdots M_m)))\,,$$

where $n, m \geq 0$, and each M_i is itself a Böhm tree. However, we need to make precise how this evidently circular definition is to be interpreted.

Formally, we define Böhm trees in terms of the following rules:

$$\frac{}{\vdash_{\mathrm{bt}} \Box}\ (\text{BT-}\Box) \qquad \frac{\vdash_{\mathrm{at}} M}{\vdash_{\mathrm{nf}} M}\ (\text{NF-AT}) \qquad \frac{}{\vdash_{\mathrm{at}} \underline{\mathrm{VAR}}(x)}\ (\text{AT-VAR})$$

$$\frac{\vdash_{\mathrm{nf}} M}{\vdash_{\mathrm{bt}} M}\ (\text{BT-NF}) \qquad \frac{\vdash_{\mathrm{nf}} M_0}{\vdash_{\mathrm{nf}} \underline{\mathrm{LAM}}(x, M_0)}\ (\text{NF-LAM}) \qquad \frac{\vdash_{\mathrm{at}} M_1 \quad \vdash_{\mathrm{bt}} M_2}{\vdash_{\mathrm{at}} \underline{\mathrm{APP}}(M_1, M_2)}\ (\text{AT-APP}).$$

These rules determine an operator F on subsets of $B = \{\mathrm{bt}, \mathrm{nf}, \mathrm{at}\} \times \underline{\Lambda}$, where $F(X)$ is the set of conclusions occurring in rule instances with premises from X:

$$F(X) = \{(\mathrm{bt}, \Box)\} \cup \cdots \cup \{(\mathrm{at}, \underline{\mathrm{APP}}(M_1, M_2)) \mid (\mathrm{at}, M_1) \in X \wedge (\mathrm{bt}, M_2) \in X\}\,.$$

F is clearly monotone, *i.e.*, $X \subseteq X' \Rightarrow F(X) \subseteq F(X')$. We say that a set $X \subseteq B$ is *F-closed* if $F(X) \subseteq X$, *i.e.*, if everything derivable by the rule instances with premises from X, is already in X.

When X is inclusive, so is $F(X)$, because inclusiveness is preserved by finite unions, and the constructor functions (as well as pairing with constants) are order-monics, *i.e.*, also *reflect* $\sqsubseteq$, so their *direct* images preserve inclusiveness.

Since both subsets of B and inclusive subsets of B are closed under arbitrary intersections, they each form complete lattices. Thus, by the Knaster-Tarski fixed-point theorem, we get the least F-closed set $\mathcal{B}^{\mathrm{fin}} \subseteq B$ by taking the intersection over all F-closed subsets of B, and the least F-closed inclusive set $\mathcal{B}^{\mathrm{inf}} \subseteq B$ as the intersection of all F-closed inclusive subsets of B.

The associated rule-induction principles are: if a predicate P on B is F-closed, then $\mathcal{B}^{\mathrm{fin}} \subseteq P$ (since $\mathcal{B}^{\mathrm{fin}}$ was the least F-closed set). Analogously, if P is both F-closed and inclusive, then $\mathcal{B}^{\mathrm{inf}} \subseteq P$. As special cases we get *inversion principles*: $\mathcal{B}^{\mathrm{fin}} = F(\mathcal{B}^{\mathrm{fin}})$ and $\mathcal{B}^{\mathrm{inf}} = F(\mathcal{B}^{\mathrm{inf}})$, *i.e.*, every element of either set can be written as the conclusion of a rule with premises in the corresponding set. Naturally, $\mathcal{B}^{\mathrm{fin}} \subseteq \mathcal{B}^{\mathrm{inf}}$, because $\mathcal{B}^{\mathrm{fin}}$ is the least of *all* fixed points of F.

We write $\mathcal{B}^{\mathrm{fin}}_s$ for $\{M \mid (s, M) \in \mathcal{B}^{\mathrm{fin}}\}$, and analogously for $\mathcal{B}^{\mathrm{inf}}_s$. The set of Böhm trees is then defined to be $\mathcal{B}^{\mathrm{inf}}_{\mathrm{bt}}$; the *finite* Böhm trees are $\mathcal{B}^{\mathrm{fin}}_{\mathrm{bt}}$.

(Note, incidentally, that Böhm trees are not simply the uniform extension of finite normal forms to λ-trees, $\mathcal{N}^{\dagger}_{\mathrm{nf}}$. The latter (which could be called infinitary normal forms) are merely the λ-trees that do not contain any evident β-redexes. We thus have $\mathcal{B}^{\mathrm{inf}}_{\mathrm{bt}} \subseteq \mathcal{N}^{\dagger}_{\mathrm{nf}}$, but the opposite inclusion does not hold: infinitary normal forms include non-Böhm trees, such as $\underline{\mathrm{LAM}}(x, \Box)$ or $\underline{\mathrm{APP}}(\underline{\mathrm{APP}}(\ldots, \mathrm{x}), \mathrm{x})$.

Nor should $\mathcal{B}_{\mathrm{bt}}^{\mathrm{inf}}$ be confused with the set of trees determined by a coinductive reading of the above rules, *i.e.*, the *greatest* fixed point of F. That set still contains, *e.g.*, the tree $\underline{\mathrm{LAM}}(x_0, \underline{\mathrm{LAM}}(x_1, \cdots))$. Even though we allow Böhm trees to be infinite, each run of curried abstractions or applications must be finite, like in the inductive reading of the rules.)

Again, we expect that a reduction-free Böhm normalizer will output Böhm trees that are compatible with the input term (soundness); that convertible inputs are mapped to the same Böhm tree (identification); and that the output tree is as large as possible (completeness).

5.2. A semantic Böhm-tree construction

The modularity of output-term generation, originally motivated by flexible generation of fresh names, also allows us to "locally" re-target the existing normalization construction to Böhm trees.

5.2.1. *Output-tree generation*

For any $f : A \to B$, where A and B are pointed cpos, we define its *smashed* strict extension, $\cdot \circledast f : A \to B$, by $\bot_A \circledast f = \bot_B$, and $a \circledast f = f(a)$ otherwise.

We first define tree-based analogs of the wrapper functions from Section 2.2, again using de Bruijn-level naming:

Definition 6 (*cf.* Def. 1)**.** Let $\widehat{\Lambda} = [\mathrm{N} \to \underline{\Lambda}]$, and define

$$\begin{aligned}
\widehat{\mathrm{VAR}}(v) &= \lambda n^{\mathrm{N}}.\, \underline{\mathrm{VAR}}(v) \\
\widehat{\mathrm{LAM}}(f) &= \lambda n^{\mathrm{N}}.\, f\,(\mathrm{gen}(n))\,(n+1) \circledast \lambda M_0^{\Lambda}.\, \underline{\mathrm{LAM}}(\mathrm{gen}(n), M_0) \\
\widehat{\mathrm{APP}}(l_1, l_2) &= \lambda n^{\mathrm{N}}.\, l_1\, n \circledast \lambda M_1^{\Lambda}.\, \underline{\mathrm{APP}}(M_1, l_2\, n).
\end{aligned}$$

Note that $\widehat{\mathrm{APP}}$ is strict in its first argument only. Making it strict in both, would revert the normalizer to always produce either $\Box$ or a finite, total λ-tree as the result, just like the original version. Strictness in the first argument does not actually matter, since the function will never be applied to a $\Box$-representative; however, from an operational viewpoint, it is convenient to know that it is safe to evaluate the argument eagerly. Making $\widehat{\mathrm{LAM}}$ non-strict would not affect correctness with respect to compatibility, but the output would no longer necessarily be a Böhm tree.

5.2.2. *The residualizing model*

The construction of the residualizing model from Section 2.3 can be reused verbatim, since it only relies on $\widehat{\Lambda}$ being a pointed cpo with continuous wrapper functions. Only the codomain of the putative Böhm normalizer changes:

Definition 7 (*cf.* Def. 2)**.** For any Δ, we define the function $\mathrm{bt}_\Delta : \Lambda^\Delta \to \underline{\Lambda}$ by

$$\mathrm{bt}_\Delta(m) \;=\; \downarrow\left(\llbracket m \rrbracket_{\mathrm{r}}\,(\lambda x^{\mathrm{V}}.\, \uparrow \widehat{\mathrm{VAR}}(x))\right) \sharp\Delta.$$

Again, we write just $\mathrm{bt} : \Lambda \to \underline{\Lambda}$ for the variant where $\sharp\Delta$ is replaced with 0, noting that it agrees with bt_Δ whenever $\Delta \cap \mathrm{G} = \emptyset$.

5.3. Correctness of the construction

The proof proceeds very much like in the original, finitary case.

5.3.1. *Basic results about compatibility*

Lemma 17. *$P^\dagger$ is inclusive for any P.*

Proof. We need to show that $\forall k \in \omega.\, \exists m \in P.\, |M|^k \sqsubseteq \langle m \rangle$ is an inclusive predicate in M. By closure under intersections, it is enough to consider a fixed k. But, for any chain $(M_i)_{i\in\omega}$, there must be an i_0 such that $\forall i \geq i_0.|M_i|^k = |M_{i_0}|^k$. Hence, if $M_{i_0} \in P^\dagger$, the $m \in P$ such that $|M_{i_0}|^k \sqsubseteq \langle m \rangle$ will also work for all subsequent i, and thus also for $\bigsqcup_{i\in\omega} M_i$. □

Lemma 18. *The constructor functions preserve compatibility:*

(a) *For all m and Δ, $\Box \leftrightarrow^\dagger_\Delta m$.*

(b) *If $v \in \Delta$, then $\underline{\mathrm{VAR}}(v) \leftrightarrow^\dagger_\Delta \mathrm{VAR}(v)$.*

(c) *If $M \leftrightarrow^\dagger_{\Delta\cup\{v\}} m$, then $\underline{\mathrm{LAM}}(v, M) \leftrightarrow^\dagger_\Delta \mathrm{LAM}(v, m)$.*

(d) *If $M_1 \leftrightarrow^\dagger_\Delta m_1$ and $M_2 \leftrightarrow^\dagger_\Delta m_2$, then $\underline{\mathrm{APP}}(M_1, M_2) \leftrightarrow^\dagger_\Delta \mathrm{APP}(m_1, m_2)$.*

Proof. Straightforward. Part (a) uses that $\Box$ is the least element in $\underline{\Lambda}$ and that $\leftrightarrow$ is reflexive; part (b) uses that both $\sqsubseteq$ and $\leftrightarrow$ are reflexive; parts (c) and (d) use that $\underline{\mathrm{LAM}}(v, \cdot)$ and $\underline{\mathrm{APP}}(\cdot, \cdot)$ are monotonic and that $\leftrightarrow$ is a congruence wrt. LAM and APP, respectively. □

5.3.2. *Correctness of the wrappers*

Definition 8 (*cf.* Def. 3)**.** For $l \in \widehat{\Lambda}$, $m \in \Lambda^\Delta$, and $s \in \{\mathrm{nf}, \mathrm{at}\}$, we define the representation relation $\lessapprox$ by

$$l \lessapprox^\Delta_s m \text{ iff } \ \forall n \geq \sharp\Delta.\, l\, n \leftrightarrow^\dagger_\Delta m \wedge (l\, n = \Box \vee l\, n \in \mathcal{B}^{\mathrm{inf}}_s)\,.$$

The correctness of the wrappers will need to be established with respect to the new definition of $\lessapprox$. The original "interface" lemmas of $\lessapprox$ (Lems. 3–5) can actually be restated verbatim – this considerably simplifies establishing soundness. Of course, the underlying meanings, and hence the proofs, of the Lemmas do change, according to the new definitions for the Böhm-tree construction.

Lemma 19 (*cf.* Lem. 3)**.** *For fixed Δ, s, and m, the predicate $P = \{l \mid l \lessapprox^\Delta_s m\}$ is pointed and inclusive.*

Proof. Pointedness is immediate. Inclusiveness also follows directly, since the relation is defined in terms of universal quantification, conjunction, finite disjunction, and inverse image by the (continuous) application function from the inclusive predicates $\leftrightarrow^\dagger_\Delta$ (by Def. 5 and Lem. 17) and $\mathcal{B}^{\mathrm{inf}}_s$ (by construction). □

Lemma 20 (*cf.* Lem. 4)**.** *The representation relation is closed under weakening and conversion:*

(a) *If* $l \lesssim_s^\Delta m$ *and* $\Delta \subseteq \Delta'$*, then also* $l \lesssim_s^{\Delta'} m$*.*

(b) *If* $l \lesssim_s^\Delta m$ *and* $m' \in \Lambda^\Delta$ *with* $m \leftrightarrow m'$*, then also* $l \lesssim_s^\Delta m'$*.*

Proof. Both parts are immediate from the definition, with (a) using monotonicity of predicate extension, and (b) using transitivity of $\leftrightarrow$. □

Lemma 21 (*cf.* Lem. 5)**.** *Representations of terms behave much like the terms themselves:*

(a) *If* $v \in \Delta$*, then* $\widehat{\mathrm{VAR}}(v) \lesssim_{\mathrm{at}}^\Delta \mathrm{VAR}(v)$*.*

(b) *If* $l_1 \lesssim_{\mathrm{at}}^\Delta m_1$ *and* $l_2 \lesssim_{\mathrm{nf}}^\Delta m_2$*, then* $\widehat{\mathrm{APP}}(l_1, l_2) \lesssim_{\mathrm{at}}^\Delta \mathrm{APP}(m_1, m_2)$*.*

(c) *If* $l \lesssim_{\mathrm{at}}^\Delta m$*, then also* $l \lesssim_{\mathrm{nf}}^\Delta m$*.*

(d) *Let* $f \in [\mathrm{V} \to \widehat{\Lambda}]$ *and* $m \in \Lambda^{\Delta \cup \{x\}}$*. If* $\forall v \notin \Delta. fv \lesssim_{\mathrm{nf}}^{\Delta \cup \{v\}} m[\mathrm{VAR}(v)/x]$*, then* $\widehat{\mathrm{LAM}}(f) \lesssim_{\mathrm{nf}}^\Delta \mathrm{LAM}(x, m)$*.*

Proof. All parts are relatively straightforward, where Lemma 18 is used throughout. Part (d) also uses transitivity of $\leftrightarrow$ and the assumption on m's free variables. □

5.3.3. *Adequacy of the residualizing model*

By virtue of the above "interface" lemmas, the verbatim insertion of Lemmas 6–9 and their proofs remain correct with the Böhm tree definitions, modulo a simple substitution of references to Lemmas 3–5 with references to Lemmas 19–21, respectively.

5.3.4. *Correctness of the Böhm-tree normalization function*

The key technical property of Böhm trees we will need for the completeness result, is that any finite cut of a Böhm tree can be extended to a finite Böhm tree, still approximating the original one. Thus, it will suffice to consider only approximants of the output term that are themselves Böhm trees.

Definition 9. For any Böhm tree $M \in \mathcal{B}_{\mathrm{bt}}^{\mathrm{inf}}$, and $k \in \omega$, we define the *Böhm cut* $\|M\|^k$ by induction on k, as follows:

$$\|M\|^0 = \Box \qquad \|\Box\|^{k+1} = \Box \qquad \|\underline{\mathrm{VAR}}(x)\|^{k+1} = \underline{\mathrm{VAR}}(x)$$
$$\|\underline{\mathrm{LAM}}(x, M_0)\|^{k+1} = \|M_0\|^k \circledast \lambda M_0'. \underline{\mathrm{LAM}}(x, M_0')$$
$$\|\underline{\mathrm{APP}}(M_1, M_2)\|^{k+1} = \|M_1\|^k \circledast \lambda M_1'. \underline{\mathrm{APP}}(M_1', \|M_2\|^k).$$

Intuitively, $\|M\|^k$ is the largest (necessarily finite) Böhm tree such that $\|M\|^k \sqsubseteq |M|^k$. It is constructed by cutting off branches early, if they do not reach a complete Böhm-subtree within the remaining height limit.

Lemma 22. *Böhm cuts satisfy:*

(a) $\forall k. \|M\|^k \in \mathcal{B}_{\mathrm{bt}}^{\mathrm{fin}}$.

(b) $\forall k.\, \|M\|^k \sqsubseteq |M|^k$.

(c) $\forall k.\, \exists k'.\, |M|^k \sqsubseteq \|M\|^{k'}$.

(These are the only *properties of $\|M\|^k$ that we will subsequently use.)*

Proof. Part (a) is shown by induction on k, using the strengthened induction hypothesis $P(k) \Leftrightarrow \forall s \in \{\mathrm{bt}, \mathrm{at}, \mathrm{nf}\}, M \in \mathcal{B}_s^{\inf}.\|M\|^k = \Box \vee \|M\|^k \in \mathcal{B}_s^{\mathrm{fin}}$. In the inductive step, we use the inversion principle for $\mathcal{B}^{\inf}$ to derive from $M \in \mathcal{B}_s^{\inf}$, the relevant properties of its immediate subtrees.

Part (b) is a straightforward induction on k.

Part (c) is proved by the inclusive variant of rule induction for $\mathcal{B}^{\inf}$, with the strengthened predicate $P(s, M) \Leftrightarrow \forall k.\exists k'.|M|^k \sqsubseteq \|M\|^{k'} \wedge (s \neq \mathrm{bt} \Rightarrow \|M\|^{k'} \neq \Box)$. To verify that this property is indeed inclusive, we first note that, by closure under intersections, it suffices to consider a fixed k. Then, for any chain $(M_i)_{i\in\omega}$, the chain $(|M_i|^k)_{i\in\omega}$ will eventually be stationary, so the k' existing at this point in the chain will also work for $\bigsqcup_{i\in\omega} M_i$, by monotonicity of $\| - \|^{k'}$.

The induction proper then considers each rule in turn; in each case, it follows easily that P holds of the rule conclusion if it holds of the premise(s). (In the case of Rule AT-APP, we also exploit that the $(\|M\|^i)_{i\in\omega}$ form a chain, so that the maximum of the two k' from the premises will work as a k' for the whole application.) □

For showing completeness, it will be convenient to disregard the exact variable names occurring in the output tree. Accordingly, we define the (evidently continuous) *shape* function $\$: \underline{\Lambda} \to \underline{\Lambda}$, by

$$\$\,\Box = \Box \qquad \$\,\underline{\mathrm{VAR}}(x) = \underline{\mathrm{VAR}}(x_\$)$$
$$\$\,\underline{\mathrm{LAM}}(x, M_0) = \underline{\mathrm{LAM}}(x_\$, \$\,M_0) \qquad \$\,\underline{\mathrm{APP}}(M_1, M_2) = \underline{\mathrm{APP}}(\$\,M_1, \$\,M_2)$$

where $x_\$$ is some arbitrary but fixed variable. From the definition of the ordering relation on $\underline{\Lambda}$, it is easy to see that if $M \sqsubseteq M'$ but $\$\,M' \sqsubseteq \$\,M$, then $M = M'$.

We can now refine the previous characterization of the wrapper functions, to state that they produce representatives that are at least as defined as some given finite tree:

Definition 10 (*cf.* Def. 4)**.** For any finite M, and $l \in \widehat{\Lambda}$, we define the *bounded* uniform definedness predicate $\mathrm{def}_M(l)$ by $\mathrm{def}_M(l) \Leftrightarrow \forall n \in \omega.\, \$\,M \sqsubseteq \$\,(l\,n)$. We also write $\mathrm{def}_M^+(l)$ for $M \neq \Box \wedge \mathrm{def}_M(l)$.

Lemma 23 (*cf.* Lem. 10)**.** *The wrapper functions preserve bounded definedness:*

(a) *For all $v \in \mathrm{V}$, $\mathrm{def}^+_{\underline{\mathrm{VAR}}(x)}(\widehat{\mathrm{VAR}}(v))$.*

(b) *If for all $v \in \mathrm{V}$, $\mathrm{def}^+_{M_0}(f\,v)$, then $\mathrm{def}^+_{\underline{\mathrm{LAM}}(x,M_0)}(\widehat{\mathrm{LAM}}(f))$.*

(c) *If $\mathrm{def}^+_{M_1}(l_1)$ and $\mathrm{def}_{M_2}(l_2)$, then $\mathrm{def}^+_{\underline{\mathrm{APP}}(M_1,M_2)}(\widehat{\mathrm{APP}}(l_1, l_2))$.*

Proof. All three parts are straightforward, by the construction of $\$$. □

Lemma 24 (*cf.* Lem. 11)**.** *Let $m \in \Lambda$ and $\rho \in [\mathrm{V} \to D_\mathrm{r}]$ be such that $\forall x \in FV(m).\, \exists l \in \widehat{\Lambda}.\, \rho(x) = \uparrow l \wedge \mathrm{def}^+_{\underline{\mathrm{VAR}}(x)}(l)$. Then, for any $M \in \underline{\Lambda}$ with $M \sqsubseteq \langle m \rangle$,*

(a) *If* $M \in \mathcal{B}^{\mathrm{fin}}_{\mathrm{at}}$, *then* $\exists l.[\![m]\!]_{\mathrm{r}}\,\rho = \uparrow l \wedge \mathrm{def}^{+}_{M}(l)$.
(b) *If* $M \in \mathcal{B}^{\mathrm{fin}}_{\mathrm{nf}}$, *then* $\mathrm{def}^{+}_{M}(\downarrow [\![m]\!]_{\mathrm{r}}\,\rho)$.
(c) *If* $M \in \mathcal{B}^{\mathrm{fin}}_{\mathrm{bt}}$, *then* $\mathrm{def}_{M}(\downarrow [\![m]\!]_{\mathrm{r}}\,\rho)$.

Proof. By rule induction for $\mathcal{B}^{\mathrm{fin}}$, with respect to the evident combined predicate. All cases are straightforward, by appeal to Lemma 23. □

We can now show the main completeness lemma:

Lemma 25. *Let* $m \in \Lambda^{\Delta}$. *If* $M \leftrightarrow^{\dagger} m$ *and* $M \in \mathcal{B}^{\mathrm{inf}}_{\mathrm{bt}}$ *then* $\$\,M \sqsubseteq \$\,\mathrm{bt}_{\Delta}(m)$.

Proof. Since for any λ-tree M, $M = \bigsqcup_k |M|^k$ (Lem. 16), by continuity of $\$$, we get the desired result from $\bigsqcup_k \$\,|M|^k \sqsubseteq \$\,\mathrm{bt}_{\Delta}(m)$. By the definition of $\bigsqcup$, it thus suffices to show, for all k, $\$\,|M|^k \sqsubseteq \$\,\mathrm{bt}_{\Delta}(m)$.

Let k be given. By Lemma 22c, there exists a k' such that $|M|^k \sqsubseteq \|M\|^{k'}$. From the definition of $M \leftrightarrow^{\dagger} m$, we get that, for this k', there exists an $m' \in \Lambda$, such that $|M|^{k'} \sqsubseteq \langle m' \rangle$ and $m \leftrightarrow m'$. Since $\|M\|^{k'} \sqsubseteq |M|^{k'}$ (Lem. 22b), we must also have $\|M\|^{k'} \sqsubseteq \langle m' \rangle$.

Let $\rho_0 = \lambda x.\uparrow \widehat{\mathrm{VAR}}(x)$; by Lemma 23a, this clearly satisfies the condition on ρ in Lemma 24. Since $\|M\|^{k'} \in \mathcal{B}^{\mathrm{fin}}_{\mathrm{bt}}$ (Lem. 22a), Lemma 24c gives us that $\mathrm{def}_{\|M\|^{k'}}(\downarrow([\![m']\!]_{\mathrm{r}}\,\rho_0))$, so in particular $\$\,\|M\|^{k'} \sqsubseteq \$\,(\downarrow([\![m']\!]_{\mathrm{r}}\,\rho_0)\,\sharp\Delta)$. Thus, using model soundness, $\$\,\|M\|^{k'} \sqsubseteq \$\,(\downarrow([\![m]\!]_{\mathrm{r}}\,\rho_0)\,\sharp\Delta) = \$\,\mathrm{bt}_{\Delta}(m)$. Finally, from $|M|^k \sqsubseteq \|M\|^{k'}$, we get $\$\,|M|^k \sqsubseteq \$\,\|M\|^{k'}$, and thus $\$\,|M|^k \sqsubseteq \$\,\mathrm{bt}_{\Delta}(m)$, as required. □

Theorem 5 (*cf.* Th. 2)**.** bt_{Δ} *from Definition 7 is a Böhm-tree normalization function on* Λ^{Δ}, i.e., *for all* $m, m' \in \Lambda^{\Delta}$,

(a) *(soundness)* $\mathrm{bt}_{\Delta}(m) \leftrightarrow^{\dagger}_{\Delta} m$ *and* $\mathrm{bt}_{\Delta}(m) \in \mathcal{B}^{\mathrm{inf}}_{\mathrm{bt}}$.
(b) *(identification) If* $m \leftrightarrow m'$, *then* $\mathrm{bt}_{\Delta}(m) = \mathrm{bt}_{\Delta}(m')$.
(c) *(completeness)* $\mathrm{bt}_{\Delta}(m)$ *is maximal among* $M \in \mathcal{B}^{\mathrm{inf}}_{\mathrm{bt}}$ *such that* $M \leftrightarrow^{\dagger} m$.

Proof. (Soundness) and *(identification)* are shown verbatim as in Theorem 2 (using Lem. 21a instead of Lem. 5a), with the unsurprising exception that unfolding the new definition for $\lessapprox$ (using Def. 8 instead of Def. 3), again taking $n = \sharp\Delta$, yields $\mathrm{bt}_{\Delta}(m) = \Box \vee \mathrm{bt}_{\Delta}(m) \in \mathcal{B}^{\mathrm{inf}}_{\mathrm{nf}}$, from which we get the desired $\mathrm{bt}_{\Delta}(m) \in \mathcal{B}^{\mathrm{inf}}_{\mathrm{bt}}$ by Rule BT-□ or Rule BT-NF, respectively.

(Completeness) Let $M \in \mathcal{B}^{\mathrm{inf}}_{\mathrm{bt}}$ with $M \leftrightarrow^{\dagger} m$ be given; we must show that M cannot be strictly greater than $\mathrm{bt}_{\Delta}(m)$. So assume that $\mathrm{bt}_{\Delta}(m) \sqsubseteq M$. By Lemma 25, $\$\,M \sqsubseteq \$\,\mathrm{bt}_{\Delta}(m)$, so we must actually have $\mathrm{bt}_{\Delta}(m) = M$. □

From downwards closure of $\leftrightarrow^{\dagger}$, we get a simple, intuitive characterization of soundness: in any finite approximation (not necessarily a level-uniform cut) of $\mathrm{bt}_{\Delta}(m)$, we can replace all holes □ with proper terms, to obtain a term convertible to the original m. (In particular, if $\mathrm{bt}_{\Delta}(m) \neq \Box$, by the inversion principle for $\mathcal{B}^{\mathrm{inf}}_{\mathrm{bt}}$, we see that the original term must have at least a head normal form.) On the other hand, completeness says that no such replacement for a hole already present

```
datatype term = VAR of string | LAM of string*term | APP of term*term
datatype tree = LVAR of string | LLAM of string*(unit->tree)
              | LAPP of (unit->tree)*(unit->tree)
datatype sem = TM of int -> tree | FUN of (unit -> sem) -> sem;

let fun down (s:sem):int->tree = fn n =>
     (case s of
        TM l => l n
      | FUN f => LLAM("g"^Int.toString n, (fn v => fn () => v)
          (down (f (fn () => TM(fn n' => LVAR("g"^Int.toString n))))
                (n+1))))
in let fun eval (m:term):(string->sem)->sem = fn p =>
        (case m of
           VAR x => p x
         | LAM(x,m0) => FUN(fn d => eval m0
                                 (fn x' => if x = x' then d () else p x'))
         | APP(m1,m2) => (case (eval m1 p) of
                            TM l => TM(fn n =>
                              LAPP((fn v => fn () => v) (l n),
                                   fn () => down (eval m2 p) n))
                          | FUN f => f (fn () => eval m2 p)))
   in let fun bt (m:term):tree =
            down (eval m (fn x => TM(fn n => LVAR(x)))) 0
      in bt end end end
```

FIGURE 3. The Böhm normalization algorithm, *BT*.

in $\mathrm{bt}_\Delta(m)$ can have even a head normal form, since this would contradict that the result tree was maximal.

Like in the finitary case, the characterization of normal forms for soundness and completeness is based on β-normalization only. If we restricted our definition of Böhm trees to only α-normal ones (*i.e.*, using a fixed naming convention for bound variables), instead of saying that the output was a *maximal* Böhm tree compatible with the input, we would have that it was the *greatest*.

5.4. An implementation of the construction

As before, the development of an actual algorithm and its proof of correctness is straightforward, given the domain-theoretic construction. Unsurprisingly, we shall need to identify $\Box$ with divergence, to obtain a computable algorithm (shown in Fig. 3), returning so-called *effective* Böhm trees.

As could be expected, we need to extend our ML program with a (necessarily non-ground) datatype `tree`, with an isomorphism $i_{\underline{\Lambda}T} : \underline{\Lambda} \xrightarrow{\cong} [\![\texttt{tree}]\!]^{\mathrm{ml}}_{\perp}$ (whereas, in the finitary case, we could simply assume that $\Lambda = [\![\texttt{term}]\!]^{\mathrm{ml}}$). This isomorphism makes it possible to reuse Lemmas 12–14 and their proofs with only few, obvious modifications. Like $\underline{\Lambda}$, `tree` is overly lazy for representing Böhm trees, so we need

to strictify the representations of $\widehat{\mathrm{LAM}}$ and $\widehat{\mathrm{APP}}$ explicitly, using the idiom `(fn v => fn () => v)`. As remarked in Section 5.2.1, the latter strictification is in fact optional, but advantageous from an efficiency perspective.

Theorem 6 (*cf.* Lem. 15)**.** *For all* $m \in \Lambda$, $i_{\underline{\Lambda}T}(\mathrm{bt}(m)) = \theta_{\mathtt{bt}}\ m$.

Proof. Straightforward adaptation of Lemmas 12–15. Additionally, the existence and basic properties of $i_{\underline{\Lambda}T}$ must be established, similarly to Lemma 12. (Here, since there are no negative occurrences of X in the domain equations, the isomorphism candidates can even be defined without *mutual* recursion.) □

We thus have that the concrete Böhm-tree algorithm is denotationally correct (up to isomorphism). However, `BT`, although well-typed and closed, is not a complete program, since `tree` is not a ground type. Hence, unlike for `NORM`, we cannot readily *observe* the program output: we first need a formal model of observation of Böhm trees.

5.5. Observing Böhm trees

5.5.1. *Computations with infinite results*

When the output of the normalizer is a partial, infinitary data structure, it is far less clear what to consider a legitimate notion of observation of the output. In particular, unlike linear streams, which can be naturally produced and printed incrementally, general trees need either a concept of "fair" autonomous production (every non-□ node will eventually be printed), or a model based on interaction, where an independent *observer* explicitly asks for successive nodes of the tree, while avoiding branches that are (or might be) □. Properly formalizing each of these in the context of our simple functional language, would be far beyond the scope of the present paper, however.

Instead, we will consider a very simple model of observation, where the observer can only ask about one specific node in the tree, for each run; the notion of interaction is thus lifted out as an extralinguistic concept into multiple (possibly even concurrent) top-level evaluations. (Of course, since the language fragment we consider is pure, many subcomputations can be shared across such evaluations; but our denotational model deliberately does not account for such quantitative aspects.) This approach will still allow us to state precisely that we can correctly inspect any reachable part of the output tree, and observationally distinguish any non-identical trees.

To keep the construction concise in our limited ML fragment, we use a uniform, numeric indexing scheme for nodes. In general, for any finitely branching (but potentially infinitely deep) rooted tree, we can associate a unique natural number to each node as follows: the root node has index 0, and for a node with index n in the i'th subtree of a k-ary root node, its index in the whole tree is $n \cdot k + i$. That is, if we consider a tree to be a node label a, and k (possibly 0) subtrees, we access the label of the n'th node of a tree t, $t \mathbin{@} n$, as follows:

$$a(t_1, \ldots, t_k) \mathbin{@} 0 = a \ \ {\scriptstyle (0 \le k)} \qquad a(t_1, \ldots, t_k) \mathbin{@} n{\cdot}k{+}i = t_i \mathbin{@} n \ \ {\scriptstyle (1 \le i \le k).}$$

```
datatype res = VR of string | LM of string | AP of unit | ER of unit;

let fun obs (t:tree):int->res = fn n =>
     (case t of
        LVAR x => if n = 0 then VR x else ER ()
      | LLAM(x,t0) => if n = 0 then LM x else obs (t0 ()) (n-1)
      | LAPP(t1,t2) => if n = 0 then AP ()
                       else if n mod 2 = 1 then obs (t1 ()) ((n-1) div 2)
                                           else obs (t2 ()) ((n-2) div 2))
in obs end
```

FIGURE 4. A simple observation function, *OBS*.

Note that the only invalid indices are those that would correspond to subtrees of a zero-ary (*i.e.*, leaf) node.

5.5.2. *Observing λ-trees*

For the specific case of λ-trees, we must also take into account partiality, and the fact that various nodes have different information as the label. Accordingly, we define the set of *observation results* by

$$\mathrm{O} = \{\mathrm{VR}(v) \mid v \in \mathrm{V}\} \cup \{\mathrm{LM}(v) \mid v \in \mathrm{V}\} \cup \{\mathrm{AP}\} \cup \{\mathrm{ER}\}$$

and define the operation $\cdot \mathbin{@} \cdot : \underline{\Lambda} \times \omega \to \mathrm{O}_\perp$ by course-of-values induction on the second argument:

$$\Box \mathbin{@} n = \perp \qquad \underline{\mathrm{VAR}}(v) \mathbin{@} 0 = \lfloor \mathrm{VR}(v) \rfloor \qquad \underline{\mathrm{VAR}}(v) \mathbin{@} n{+}1 = \lfloor \mathrm{ER} \rfloor$$
$$\underline{\mathrm{LAM}}(v, M_0) \mathbin{@} 0 = \lfloor \mathrm{LM}(v) \rfloor \qquad \underline{\mathrm{LAM}}(v, M_0) \mathbin{@} n{+}1 = M_0 \mathbin{@} n$$
$$\underline{\mathrm{APP}}(M_1, M_2) \mathbin{@} 0 = \lfloor \mathrm{AP} \rfloor$$
$$\underline{\mathrm{APP}}(M_1, M_2) \mathbin{@} 2{\cdot}n{+}1 = M_1 \mathbin{@} n \qquad \underline{\mathrm{APP}}(M_1, M_2) \mathbin{@} 2{\cdot}n{+}2 = M_2 \mathbin{@} n.$$

We note that node-observations completely characterize a λ-tree:

Lemma 26. *If for all $n \in \omega$, $M \mathbin{@} n = M' \mathbin{@} n$, then $M = M'$.*

Proof. By Lemma 16, it suffices to show that $\forall k.|M|^k = |M'|^k$, which follows by a straightforward induction on k. □

5.5.3. *Implementing tree-observations in ML*

The ML implementation of the observation function is shown in Figure 4. To represent observation results, we introduce another ground datatype `res`; like for `term`, we assume that $[\![\texttt{res}]\!]^{\mathrm{ml}} = \mathrm{O}$, and that the meanings of the constructors agree. We also assume that the ML fragment has been extended with aritmetic operators `-`, `div`, and `mod`, completely analogous to the existing `+`.

Lemma 27. *For all $M \in \underline{\Lambda}$ and $n \in \omega$, $M \mathbin{@} n = i_{\underline{\Lambda}T}(M) \star \lambda t.\, \theta_{\mathtt{obs}}\, t \star \lambda f.\, f\, n$.*

Proof. The proof is by a straightforward course-of-values induction on n, using the fixed-point equation for $\mathrm{fix}(\Theta_{\mathtt{obs}})$ and the "translating" properties of $i_{\underline{\Lambda}T}$ (analogous to those of i_{DS} in Lem. 12) throughout. □

Consider now an ML program whose datatype declarations are a union of those in Figures 3 and 4 (in any order), and take

```
OBSBT = fn m => OBS (BT m).
```

This is an ML expression of type `term -> int -> res`, *i.e.*, a complete program.

Theorem 7 (*cf.* Th. 4)**.** *For all $m \in \Lambda$, $n \in \omega$, and $o \in \mathrm{O}$, `OBSBT` $\bullet\, (m, n) = o$ iff* $\mathrm{bt}(m) \mathbin{@} n = \lfloor o \rfloor$.

Proof. We first note that, since `BT` and `OBS` are closed, for any ξ, $[\![\mathtt{BT}]\!]^{\mathrm{ml}}\,\xi = \lfloor \theta_{\mathtt{bt}} \rfloor$ and $[\![\mathtt{OBS}]\!]^{\mathrm{ml}}\,\xi = \lfloor \theta_{\mathtt{obs}} \rfloor$. The result then follows directly from Theorem 6, Lemma 27, and Theorem 3. □

Moreover, Lemma 26 tells us that `BT` is also operationally correct with respect to any other observation function (including ones using more user-friendly access paths), because `OBS`-observations can already distinguish all elements of type `tree`, even those that do not represent proper Böhm trees.

6. Conclusions and perspectives

We have presented a domain-theoretic analysis of a normalization-by-evaluation construction for the untyped λ-calculus. Compared to the typed case, the main difference is a change from induction on types to general recursion, both for function definitions and for the domains and relations on them. That the correctness proof has a generalized computational-adequacy result at its core, further strengthens the connection between normalization and evaluation. Moreover, the algorithmic content of the construction corresponds very directly to a simple functional program, enabling a precise verification of the normalizer as actually implemented.

There are several possible directions in which to extend or modify the present work. Especially in the infinitary variant of the algorithm, there is some leeway in exactly what kind of λ-trees we wish to consider as output, and our observation model for them. It should also be possible to extend the language and notion of normalization with interpreted constants in a suitable sense. But already the current results indicate that the fundamental ideas of NBE are not incompatible with general recursive types. Thus, reduction-free normalization may provide a complementary view of other equational systems that are currently analyzed using exclusively reduction-based methods. It might even be possible to find unified formulations of rewriting-theoretic and model-theoretic normalization results about particular such systems.

Acknowledgements. The authors wish to thank Olivier Danvy, as well as the FOSSACS'04 and RAIRO-ITA reviewers, for their insightful comments.

References

[1] K. Aehlig and F. Joachimski, Operational aspects of untyped normalization by evaluation. *Math. Structures Comput. Sci.* **14** (2004) 587–611.

[2] H.P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, revised edition (1984).

[3] U. Berger and H. Schwichtenberg, An inverse of the evaluation functional for typed λ-calculus, in *Proc. of the Sixth Annual IEEE Symposium on Logic in Computer Science*, Amsterdam, The Netherlands (July 1991) 203–211.

[4] T. Coquand and P. Dybjer, Intuitionistic model constructions and normalization proofs. *Math. Structures Comput. Sci.* **7** (1997) 75–94.

[5] A. Filinski, A semantic account of type-directed partial evaluation, in *International Conference on Principles and Practice of Declarative Programming*, edited by G. Nadathur, Springer-Verlag, Paris, France. *Lect. Notes Comput. Sci.* **1702** (1999) 378–395

[6] A. Filinski and H.K. Rohde, A denotational account of untyped normalization by evaluation, in *7th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2004)*, edited by I. Walukiewicz, Springer-Verlag, Barcelona, Spain *Lect. Notes Comput. Sci.* **2987** (2004) 167–181.

[7] A. Filinski and H.K. Rohde, *Denotational aspects of untyped normalization by evaluation* (extended version, with detailed proofs). BRICS Report RS-05-4, University of Aarhus, Denmark (February 2005). Available from `http://www.brics.dk/RS/05/4/`.

[8] B. Grégoire and X. Leroy, A compiled implementation of strong reduction, in *Proc. of the Seventh ACM SIGPLAN International Conference on Functional Programming*, edited by S. Peyton Jones, ACM Press, Pittsburgh, Pennsylvania, SIGPLAN Notices **37** (2002) 235–246.

[9] R. Milner, M. Tofte, R. Harper and D. MacQueen, *The Definition of Standard ML*. The MIT Press, revised edition (1997).

[10] J.C. Mitchell, *Foundations for Programming Languages*. The MIT Press (1996).

[11] A.M. Pitts, Computational adequacy via "mixed" inductive definitions, in *Mathematical Foundations of Programming Semantics*. Springer-Verlag. *Lect. Notes Comput. Sci.* **802** (1993) 72–82.

[12] A.M. Pitts, Relational properties of domains. *Inform. Comput.* **127** (1996) 66–90.

[13] G.D. Plotkin, LCF considered as a programming language. *Theor. Comput. Sci.* **5** (1977) 223–255.

[14] M.R. Shinwell, A.M. Pitts and M.J. Gabbay, FreshML: Programming with binders made simple, in *Eighth ACM SIGPLAN International Conference on Functional Programming*, ACM Press, Uppsala, Sweden (2003) 263–274.