CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

On Definability and Normalization by Evaluation in the Logical Framework

Master's thesis in Computer science and engineering

Frederik Ramcke

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

On Definability and Normalization by Evaluation in the Logical Framework

Frederik Ramcke

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

On Definability and Normalization by Evaluation in the Logical Framework

Frederik Ramcke

Supervisor: Andreas Abel, Department of Computer Science and Engineering
Examiner: Nils Anders Danielsson, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

On Definability and Normalization by Evaluation in the Logical Framework

Frederik Ramcke
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The question of *definability* asks to characterise what objects of the meta-theory are definable by a given calculus; for example, the *untyped λ-calculus* characterises exactly the *Turing-computable* functions.
This thesis gives a characterisation of LF-definability—definability in a variant of the *Edinburgh Logical Framework*—in terms of a notion of *Kripke predicates*. The constructions in this thesis are heavily inspired by, and generalise, those by Jung and Tiuryn, who gave a definability result for the simply-typed λ-calculus in 'A New Characterization of Lambda Definability'.

Keywords: Definability, Logical Framework, Normalization by Evaluation, NbE.

Acknowledgements

Thank you to Andreas for introducing me to the topic and providing guidance and the occasional enlightening explanation. Thank you also to Nisse for the detailed and constructive feedback—you helped me improve the thesis significantly! Finally, thank you to Sandro and everybody at the Initial Types Club for interesting lectures and a welcome change of scenery.

Frederik Ramcke, Gothenburg, November 2019

Contents

Chapter 1

Introduction

The notion of λ-*definability* characterises those mathematical entities which can be expressed in Church's untyped λ-calculus, i.e. those x for which a λ-term t satisfying $\langle t \rangle = x$ exists (where $\langle t \rangle$ is the interpretation of t). As one of the earliest results in the field of theoretical computer science λ-definability was proven equivalent to *Turing-computability* [22], i.e. functions $f : \mathbb{N} \to \mathbb{N}$ are λ-definable if and only if they are computable (cf. the Church–Turing thesis). This early line of research leads us to the problem of finding a similarly *semantic* characterisation of definability for the various *typed* variants of the λ-calculus, in which the valid terms are restricted to those that are 'well-typed' according to a chosen set of typing rules.

A semantic formulation for the corresponding notion of definability for the *simply typed lambda calculus* (STLC) was given in 1993 by Jung and Tiuryn [17]. The STLC is arguably the most primitive of the *typed* λ-calculi, with only *function types* $A \to B$ *and type constants* ι. Not many definability results are known for more powerful systems.

We are particularly interested in type systems with *dependent types*, which are central to the use of type theory as a language for mathematics—the ultimate goal of this line of research is to find a semantic formulation of definability for a fully dependent lambda calculus (e.g. $\lambda\Pi\omega$, the Calculus of Constructions). Such a result can be expected to hold some philosophical value, in that it would give us a deeper intuitive understanding of this calculus; on the practical side of things it would likely involve novel technical constructions that could be independently useful as tools in reasoning about dependently typed lambda calculi.

This thesis makes a small step towards that overarching goal. We derive a semantic characterisation of definability in the *Edinburgh Logical Framework (LF):*

Theorem (Main theorem)**.** *An object is definable in LF iff it satisfies all Kripke predicates.*

The Edinburgh Logical Framework [15] features a dependent type system without universes, where quantification is limited to *small* types (and where types may only depend non-computationally on values), corresponding to $\lambda\Pi$ in Barendregt's λ-cube [6].

1.1 Inspiration and Prior Work

The main inspiration for this thesis stems from Jung and Tiuryn's work on STLC-definability [17], in which they proved it equivalent to satisfaction of all 'Kripke logical relations with varying arity'. The idea to apply Jung and Tiuryn's ideas to the LF calculus came from Andreas Abel; Andreas Abel had formalised the STLC-definability result completely in Agda [1], which made the structure of the constructions and proofs more obvious, and he proposed an extension to dependent types as an interesting problem for future work.

'Kripke logical relations with varying arity' are an extension of the earlier notion of *logical relations*, which were already employed by Plotkin in 1973 [21] in analysing the decision problem of STLC-definability (i.e. *deciding* for a given object x whether a defining STLC term t exists). Jung and Tiuryn's *Kripke relations* generalise plain logical relations to *Kripke models*, which are a well-known way of assigning a semantics to the simply-typed λ-calculus, introduced by Mitchell and Moggi [20].

The second key piece to the definability result is a formulation of *evaluation* as a Kripke predicate, which turns out to correspond to the algorithm now known as *Normalisation by Evaluation* (NbE) [8]. Coquand gave a version of the NbE algorithm using a Kripke model for the STLC with explicit substitutions [12] which we will extend to the case of LF (without explicit substitions).

1.2 Contribution

The main contribution of this thesis lies in the derivation of a novel characterisation of LF-definability. Central to this result is a novel 'semi-semantic' formulation of LF. As a by-product of our efforts we also obtain an elegant normalisation proof for LF.

It is important to point out that the proof ideas in this thesis are not fundamentally novel, but are adapted from Jung and Tiuryn [17].

1.3 The Structure of this Thesis

The first part of this thesis summarises Jung and Tiuryn's STLC-definability result. Its function is to convey an idea of the 'bigger picture', that is, to introduce the reader to proof ideas that will later reappear in the second part in a much more technically involved setting.

Part II comprises the heart of this thesis work—it contains the proof of the LF-definability theorem (in Section 7.3). The structure of this part mirrors Part I to a large extent. Note that Part II constitutes novel work while Part I consists entirely of a presentation of previously known results.

Chapter 2

Preliminaries

2.1 Expectations from the Reader

This thesis assumes from the reader familiarity with:

- Type theory as a foundation of mathematics; this thesis document is set in an informal, extensional type theory, while the accompanying code contains formal proofs in Agda, an intensional type theory.
- Agda; the reader is advised to read the technical developments in Part 2 in tandem with the accompanying Agda code. Furthermore, the presentation on paper makes use of several Agda-isms (in particular as far as notation is concerned).
- The simply-typed λ-calculus (STLC); the first part of the thesis briefly summarises a result on STLC-definability, but does not give much intuition to the definitions. The reader is expected to be familiar with the usual intrinsically-typed presentation of the STLC (cf. for example Section 5 in Altenkirch and Reus [4]).
- De Bruijn indices [11]; both the simply-typed calculus in Part 1 as well as the presentation of LF in Part 2 use de Bruijn indices to encode variables in their syntax.

2.2 Informal Type Theory

Throughout the thesis we will work in an informal, *extensional* type theory. If you like, you can think of this as Agda plus extensional equality (and some amount of handwaving); this allows us to skim over some of the complications plaguing the Agda code, which is based on an *intensional* type theory [18] where the wrangling of type equalities incurs a lot of syntactical and mental overhead.

The notational conventions we are going to follow are meant to mirror Agda's syntax. We are additionally going to take a few liberties in our notation:

- We will sometimes write arguments as subscripts. For example, we write $[\![A]\!]_\Gamma$ instead of $[\![A]\!]\,\Gamma$.

- We will use set-theoretic notation when working with subsets.
  - Given a predicate $P : S \to \mathrm{Set}$ we write $x \in P$ to mean $P\,x$.
  - We will also informally quantify over set-elements, i.e. we write $(x \in P) \to \ldots$ instead of $\{x : S\} \to P\,x \to \ldots$.
  - Finally, we will confuse *elements* and *witnesses of element-hood*, i.e. if $f : (x \in P) \to X$ (informal for $f : \{x : S\} \to P\,x \to X$), $x : S$ and $p : x \in P$, we will write $f\,x$ to mean $f\,\{x\}\,p$, leaving the witness implicit.

2.3 Agda

The code accompanying the thesis is written in Agda, an implementation of an intensional type theory [10, 23]. The properties of Agda's type theory are heavily customisable. Our code uses the following particular features of Agda:

Pattern matching Agda allows the user to write definitions by pattern matching. This allows us to translate the informal definitions we might write on paper into Agda code in a natural way.

Note that Agda's pattern matching allows us to prove *Axiom K* (equivalently: *uniqueness of identity proofs*). Our code makes use of this feature, that is, we are implicitly assuming Axiom K in our meta-theory.

Customisable syntax Agda allows us to define custom mixfix operators; this allows us for example to define and use semantic brackets the same way we do on paper. Furthermore, Agda's customisable syntax allows us to swap the order of function arguments even in the case of dependencies between them.

Part I

Review of Definability in the Simply-Typed Lambda Calculus

Chapter 3

STLC

The main result of this thesis, a characterisation of LF-definability in terms of Kripke predicates, is heavily inspired by Jung and Tiuryn's work on definability in the simply-typed λ-calculus (STLC) [17]. In order to provide context to the second part of the thesis, as well as to guide the reader, we will review (a variant of) the results of Jung and Tiuryn in this first part of the thesis. Note that we didn't formalise this part of the thesis in Agda.

This chapter comprises a definition of the simply-typed λ-calculus along with its semantics, as well as a notion of Kripke models.

3.1 Syntax

The *types* of the STLC consist of base types (or type constants) and function types. Note that we will not make the notion of a base type fully precise; in general one would parameterise the calculus over a set of base types and their semantics.

$$\mathrm{Ty} : \mathrm{Set} \qquad \frac{\iota \text{ base type}}{\iota : \mathrm{Ty}} \qquad \frac{A, B : \mathrm{Ty}}{A \Rightarrow B : \mathrm{Ty}}$$

A *context* consists of a list of types representing term variables.

$$\mathrm{Cxt} : \mathrm{Set} \qquad \frac{}{\emptyset : \mathrm{Cxt}} \qquad \frac{\Gamma : \mathrm{Cxt} \quad A : \mathrm{Ty}}{(\Gamma, A) : \mathrm{Cxt}}$$

Variables are simply de Bruijn indices into the context.

$$\mathrm{Var} : \mathrm{Cxt} \to \mathrm{Ty} \to \mathrm{Set} \qquad \frac{\Gamma : \mathrm{Cxt} \quad A : \mathrm{Ty}}{\mathrm{v}_0 : \mathrm{Var}\,(\Gamma, A)\,A} \qquad \frac{\mathrm{v}_n : \mathrm{Var}\,\Gamma\,A \quad B : \mathrm{Ty}}{\mathrm{v}_{n+1} : \mathrm{Var}\,(\Gamma, B)\,A}$$

The *terms* of the simply-typed λ-calculus are given by variables, abstraction (corresponding to the λ-binder) and application.

$$\mathrm{Tm} : \mathrm{Cxt} \to \mathrm{Ty} \to \mathrm{Set} \qquad \frac{v : \mathrm{Var}\,\Gamma\,A}{\mathrm{var}\,v : \mathrm{Tm}\,\Gamma\,A}$$

$$\frac{t : \mathrm{Tm}\,(\Gamma, A)\,B}{\mathrm{abs}\,t : \mathrm{Tm}\,\Gamma\,(A \Rightarrow B)} \qquad \frac{t : \mathrm{Tm}\,\Gamma\,(A \Rightarrow B) \quad u : \mathrm{Tm}\,\Gamma\,A}{\mathrm{app}\,t\,u : \mathrm{Tm}\,\Gamma\,B}$$

3.1.1 Context Inclusions and Weakening

The natural notion of *inclusion* of contexts is given by *order-preserving embeddings* $\Gamma \subseteq \Delta$.

$$\frac{\Gamma : \mathrm{Cxt}}{\mathrm{refl}^{\mathrm{weak}} : \Gamma \subseteq \Gamma} \qquad \frac{i : \Gamma \subseteq \Delta}{\downarrow^{\mathrm{weak}}\ i : \Gamma \subseteq (\Delta, A)} \qquad \frac{i : \Gamma \subseteq \Delta}{\uparrow^{\mathrm{weak}}\ i : (\Gamma, A) \subseteq (\Delta, A)}$$

Given a term $t : \mathrm{Tm}\,\Gamma\,A$ and an embedding $i : \Gamma \subseteq \Delta$ one can construct a correspondingly *weakened* term $t{\downarrow_i} : \mathrm{Tm}\,\Delta\,A$ by updating the de Bruijn indices accordingly. Furthermore, one can define a corresponding *semantic action* on contexts $\langle i \rangle^{\mathrm{weak}} : [\Delta]^* \to [\Gamma]^*$. (The detailed definitions are left for the second part of the thesis.)

3.2 Standard Semantics

The simply-typed λ-calculus can be seen as a 'sub-theory' of our meta-type-theory (i.e. Agda). We will refer to this natural embedding of STLC types and terms into the meta-theory as the *standard semantics*.

The standard semantics of function types is given by the full function space of the meta-theory, that is STLC functions will be denoted as proper functions.

$$\begin{aligned} [\_] &: \mathrm{Ty} \to \mathrm{Set} \\ [\iota] &= (\text{parameter to the theory}) \\ [A \Rightarrow B] &= [A] \to [B] \end{aligned}$$

The semantics of a context (a list of types) is given by tuples of values of the corresponding types. Note that $\top$ denotes the *unit* type with exactly one inhabitant.

$$\begin{aligned} [\_]^* &: \mathrm{Cxt} \to \mathrm{Set} \\ [\emptyset]^* &= \top \\ [\Gamma, A]^* &= [\Gamma]^* \times [A] \end{aligned}$$

Finally, the standard-semantic denotation of STLC terms is given by the following well-known evaluation function:

$$\begin{aligned} \langle \_ \rangle &: \forall\{\Gamma\ A\} \to \mathrm{Tm}\,\Gamma\,A \to [\Gamma]^* \to [A] \\ \langle \mathrm{var}\ \mathrm{v}_0 \rangle\,(\gamma, a) &= a \\ \langle \mathrm{var}\ \mathrm{v}_{n+1} \rangle\,(\gamma, a) &= \langle \mathrm{var}\ \mathrm{v}_n \rangle\,\gamma \\ \langle \mathrm{app}\,t\,u \rangle\,\gamma &= \langle t \rangle\,\gamma\,(\langle u \rangle\,\gamma) \\ \langle \mathrm{abs}\,t \rangle\,\gamma &= \lambda a \to \langle t \rangle\,(\gamma, a) \end{aligned}$$

Note that the question of *definability* ultimately concerns these standard semantics: we want to characterise those meta-theoretic objects $f : [\Gamma]^* \to [A]$ for which there exists a defining term $t : \mathrm{Tm}\,\Gamma\,A$ such that $\langle t \rangle = f$.

3.3 Kripke Models

Mitchell and Moggi [20] introduced *Kripke-style models* for the simply-typed λ-calculus in 1987 (the citation leads to a later version of the paper from 1991). A Kripke model consists of the following data:

Worlds A set of 'worlds' ordered by a reflexive and transitive relation.

$$W : \mathrm{Set} \qquad \_\prec\_ : W \to W \to \mathrm{Set}$$

Type semantics A semantics of types indexed by worlds.

$$[\![\_]\!]\_ : \mathrm{Ty} \to W \to \mathrm{Set}$$

Transport An operation that transports values to 'later' worlds.

$$\_\nearrow\_ : \forall\{A\, w\, w'\} \to [\![A]\!]_w \to w \prec w' \to [\![A]\!]_{w'}$$

Application An application function at each world.

$$\mathrm{apply} : \forall\{A\, B\, w\} \to [\![A \Rightarrow B]\!]_w \to [\![A]\!]_w \to [\![B]\!]_w$$

Term semantics A denotation function indexed by worlds,

$$(\!|\_|\!) : \forall\{\Gamma\, A\, w\} \to \mathrm{Tm}\, \Gamma\, A \to [\![\Gamma]\!]^*_w \to [\![A]\!]_w$$

where $[\![\_]\!]^*$ is the pointwise extension of world-indexed type semantics to contexts.

$$\begin{aligned} [\![\_]\!]^*\_ &: \mathrm{Cxt} \to \mathrm{Cxt} \to \mathrm{Set} \\ [\![\emptyset]\!]^*_w &= \top \\ [\![\Gamma, A]\!]^*_w &= [\![\Gamma]\!]^*_w \times [\![A]\!]_w \end{aligned}$$

The supplied data further need to satisfy a list of laws that ensure that the structure does indeed constitute a model of the simply-typed λ-calculus. Since we are not going to talk about properties of Kripke models in general we do not need to talk about these laws explicitely here.

3.3.1 Kripke Predicates

A *Kripke predicate* over a given Kripke model is an indexed family of predicates

$$\mathcal{P}[\![\_]\!]\_ : (A : \mathrm{Ty}) \to (w : W) \to [\![A]\!]_w \to \mathrm{Set}$$

that is

- monotone at base types,

$$a \in \mathcal{P}[\![\iota]\!]_w \to (p : w \prec w') \to (x \nearrow p) \in \mathcal{P}[\![\iota]\!]_{w'},$$

- and has the 'Kripke function space' property,

$$f \in \mathcal{P}[\![A \Rightarrow B]\!]_w \iff \forall(w' : W)(p : w \prec w')(a \in \mathcal{P}[\![A]\!]_{w'}) \to \mathrm{apply}\,(f \nearrow p)\, a \in \mathcal{P}[\![B]\!]_{w'},$$

where ' $\iff$ ' stands for *logical equivalence* (if and only if).

We extend such a $\mathcal{P}[\![\_]\!]\_$ to environments pointwise as follows:

$$\begin{aligned}\mathcal{P}[\![\_]\!]^*_\_ &: (\Gamma : \mathrm{Cxt}) \to (w : W) \to [\![\Gamma]\!]^*_w \to \mathrm{Set}\\ \eta \in \mathcal{P}[\![\varnothing]\!]^*_w &= \top\\ (\eta, \alpha) \in \mathcal{P}[\![\Gamma, A]\!]^*_w &= \eta \in \mathcal{P}[\![\Gamma]\!]^*_w \wedge \alpha \in \mathcal{P}[\![A]\!]_w\end{aligned}$$

Any Kripke predicate can be shown to satisfy the 'fundamental lemma' of Kripke predicates.

Lemma (Fundamental Lemma of Kripke Predicates)**.** *Given* $t : \mathrm{Tm}\,\Gamma\,A$ *and* $\eta \in \mathcal{P}[\![\Gamma]\!]^*_w$ *we have* $(\!|t|\!)\,\eta \in \mathcal{P}[\![A]\!]_w$*.*

Proof. For a proof see Mitchell and Moggi[20] (note that they use somewhat different notations and definitions). □

Chapter 4

Normalisation by Evaluation (NbE)

In this chapter we will briefly review the normalisation algorithm [8] known as *normalisation by evaluation* (NbE). This turns out to be a key part of Jung and Tiuryn's [17] STLC-definability result (which we will summarise in Chapter 5).

The key idea of NbE is to define a particular Kripke model that contains normal forms at base types, and that allows weakening at function types. One then defines *reflection* and *reification* functions mutually recursively, respectively 'reflecting' *neutral* terms *into* the model and 'reifying' *normal* terms *out of* the model.

4.1 Normal Forms

A term is in *(β-)normal form* if it contains no β-redexes, i.e. subterms of the form $\mathrm{app}\,(\mathrm{abs}\,t)\,u$. We define normal forms mutually with neutral terms as subsets of (predicates over) the type of all terms: $\mathrm{Ne}, \mathrm{Nf} : (\Gamma : \mathrm{Cxt}) \to (A : \mathrm{Ty}) \to \mathrm{Tm}\,\Gamma\,A \to \mathrm{Set}$.

Note that we use set-style notation informally here, leaving out the witnesses of element-hood. Note further that A and B are intended to be quantify over *all* types.

$$\frac{v : \mathrm{Var}\,\Gamma\,A}{\mathrm{var}\,v \in \mathrm{Ne}\,\Gamma\,A} \qquad \frac{t \in \mathrm{Ne}\,\Gamma\,(A \Rightarrow B) \quad u \in \mathrm{Nf}\,\Gamma\,A}{\mathrm{app}\,t\,u \in \mathrm{Ne}\,\Gamma\,B}$$

$$\frac{t \in \mathrm{Ne}\,\Gamma\,A}{t \in \mathrm{Nf}\,\Gamma\,A} \qquad \frac{t \in \mathrm{Nf}\,(\Gamma, A)\,B}{\mathrm{abs}\,t \in \mathrm{Nf}\,\Gamma\,(A \Rightarrow B)}$$

4.2 NbE Kripke Model

The Kripke model used by the NbE algorithm is defined as follows.

Worlds Worlds are given by the set of contexts, i.e. $W = \mathrm{Cxt}$, ordered by *order-preserving embeddings* $\Gamma \subseteq \Delta$. Transitivity of the $\_ \subseteq \_$ relation is witnessed by a *composition* operation $\_ \bullet \_ : \Gamma \subseteq \Delta \to \Delta \subseteq \Omega \to \Gamma \subseteq \Omega$.

Type semantics The key to constructing the NbE model is to define the indexed type semantics to allow for weakening at function types. We add a superscript $\mathcal{N}$ to denote that we are in the NbE model.

$$
\begin{aligned}
[\![\_]\!]^{\mathcal{N}}_{\_} &: \mathrm{Ty} \to \mathrm{Cxt} \to \mathrm{Set} \\
[\![\iota]\!]^{\mathcal{N}}_{\Gamma} &= \{t | t \in \mathrm{Ne}\,\Gamma\,\iota\} \\
[\![A \Rightarrow B]\!]^{\mathcal{N}}_{\Gamma} &= \{\Delta : \mathrm{Cxt}\} \to (i : \Gamma \subseteq \Delta) \to [\![A]\!]^{\mathcal{N}}_{\Delta} \to [\![B]\!]^{\mathcal{N}}_{\Delta}
\end{aligned}
$$

Transport The previous definition ensures that we can 'transport' values along context embeddings:

$$
\begin{aligned}
& & \_\nearrow\_ &: \forall\{\Gamma\,A\,\Delta\} \to [\![A]\!]^{\mathcal{N}}_{\Gamma} \to \Gamma \subseteq \Delta \to [\![A]\!]^{\mathcal{N}}_{\Delta} \\
\{A = \iota\} & & t \nearrow i &= t \downarrow i \\
\{A = (A \Rightarrow B)\} & & f \nearrow i &= \lambda i'\,a \to f\,(i \bullet i')\,a
\end{aligned}
$$

Application Application in the NbE model is simply meta-application after applying the identity weakening.

$$
\begin{aligned}
\mathrm{apply}^{\mathcal{N}} &: \forall\{A\,B\,\Gamma\} \to [\![A \Rightarrow B]\!]^{\mathcal{N}}_{\Gamma} \to [\![A]\!]^{\mathcal{N}}_{\Gamma} \to [\![B]\!]^{\mathcal{N}}_{\Gamma} \\
\mathrm{apply}^{\mathcal{N}}\,f\,x &= f\,\mathrm{refl}^{\mathrm{weak}}\,x
\end{aligned}
$$

Term semantics Evaluation into the NbE model is defined recursively:

$$
\begin{aligned}
(\!|\_|\!)^{\mathcal{N}} &: \forall\{\Gamma\,A\,\Delta\} \to \mathrm{Tm}\,\Gamma\,A \to [\![\Gamma]\!]^{*\mathcal{N}}_{\Delta} \to [\![A]\!]^{\mathcal{N}}_{\Delta} \\
(\!|\mathrm{var}\,v_0|\!)^{\mathcal{N}}\,(\eta, \alpha) &= \alpha \\
(\!|\mathrm{var}\,v_{n+1}|\!)^{\mathcal{N}}\,(\eta, \alpha) &= (\!|\mathrm{var}\,v_n|\!)^{\mathcal{N}}\,\eta \\
(\!|\mathrm{app}\,t\,u|\!)^{\mathcal{N}}\,\eta &= \mathrm{apply}^{\mathcal{N}}\,((\!|t|\!)^{\mathcal{N}}\,\eta)\,((\!|u|\!)^{\mathcal{N}}\,\eta) \\
(\!|\mathrm{abs}\,t|\!)^{\mathcal{N}}\,\eta &= \lambda i\,a \to (\!|t|\!)^{\mathcal{N}}(\eta \nearrow i, a)
\end{aligned}
$$

Here $\eta \nearrow i$ is the environment that we obtain by transporting the values in η along i pointwise. (We reuse the same notation as we used for transport of values, for typographical reasons.)

$$
\begin{aligned}
& & \_ \nearrow \_ &: \forall\{\Gamma\,A\,\Delta\} \to [\![\Gamma]\!]^{*\mathcal{N}}_{\Delta} \to \Delta \subseteq \Omega \to [\![\Gamma]\!]^{*\mathcal{N}}_{\Omega} \\
\{\Gamma = \varnothing\} & & \mathrm{tt} \nearrow i &= \mathrm{tt} \\
\{\Gamma = (\Gamma, A)\} & & (\eta, \alpha) \nearrow i &= (\eta \nearrow i, \alpha \nearrow i)
\end{aligned}
$$

4.3 Normalisation

The heart of the NbE algorithm consists of two mutually defined functions: 'reflect' *injects* neutral terms into the model while 'reify' *extracts* normal forms

out of the model.

$$\begin{aligned}
\mathrm{reflect} &: \{\Gamma : \mathrm{Cxt}\} \to (A : \mathrm{Ty}) \to \mathrm{Ne}\,\Gamma\,A \to [\![A]\!]^{\mathcal{N}}_{\Gamma} \\
\mathrm{reflect}_{\iota}\,t &= t \\
\mathrm{reflect}_{A \Rightarrow B}\,t &= \lambda i\,a \to \mathrm{reflect}_{B}\,(\mathrm{app}\,(t{\downarrow_i})\,(\mathrm{reify}_{A}\,a))
\end{aligned}$$

$$\begin{aligned}
\mathrm{reify} &: \{\Gamma : \mathrm{Cxt}\} \to (A : \mathrm{Ty}) \to [\![A]\!]^{\mathcal{N}}_{\Gamma} \to \mathrm{Nf}\,\Gamma\,A \\
\mathrm{reify}_{\iota}\,t &= t \\
\mathrm{reify}_{A \Rightarrow B}\,f &= \mathrm{abs}\,(\mathrm{reify}_{B}\,(f\,(\downarrow^{\mathrm{weak}}\ \mathrm{refl}^{\mathrm{weak}})\,(\mathrm{reflect}_{A}\,(\mathrm{var}\,v_0))))
\end{aligned}$$

To normalise a term $t : \mathrm{Tm}\,\Gamma\,A$ all we have to do is *evaluate* it in the identity environment $\mathrm{fresh}_{\Gamma} : [\![\Gamma]\!]^{*\mathcal{N}}_{\Gamma}$ and *reify* the result:

$$\begin{aligned}
\mathrm{fresh}_{_} &: (\Gamma : \mathrm{Cxt}) \to [\![\Gamma]\!]^{*\mathcal{N}}_{\Gamma} \\
\mathrm{fresh}_{\emptyset} &= \mathrm{tt} \\
\mathrm{fresh}_{(\Gamma,A)} &= (\mathrm{fresh}_{\Gamma} \nearrow (\downarrow^{\mathrm{weak}}\ \mathrm{refl}^{\mathrm{weak}}), \mathrm{reflect}_{A}\,(\mathrm{var}\,v_0))
\end{aligned}$$

$$\begin{aligned}
\mathrm{norm} &: \forall\{\Gamma\,A\} \to \mathrm{Tm}\,\Gamma\,A \to \mathrm{Nf}\,\Gamma\,A \\
\mathrm{norm}\,t &= \mathrm{reify}_{A}\,((\!|t|\!)^{\mathcal{N}}\,\mathrm{fresh}_{\Gamma})
\end{aligned}$$

This concludes the computational story behind the NbE algorithm; what remains is to prove the procedure sound, i.e. that normalisation preserves the computational behaviour of terms. This will follow as a by-product of the definability result in the next chapter.

Chapter 5

STLC-Definability

This chapter contains a proof of a slightly restricted version of Jung and Tiuryn's [17] STLC-definability result, characterising STLC-definability in terms of Kripke predicates. The original theorem uses the more general notion of 'Kripke logical relations with varying arity', which does not readily generalise to the LF calculus.

In this chapter we skip almost all of the technical details since we are going to revisit them in the next part in great detail; the results in this chapter can intuitively be seen as special cases of the results for LF in Part 2 (if one identifies the STLC with a subset of LF).

5.1 Kripkefied Standard Semantics

The first part of the definability theorem consists of a presentation of the standard semantics of the STLC as a Kripke model (marked by a superscript $^{\mathcal{S}}$).

Worlds Like in the NbE model the worlds are given by contexts, ordered by order-preserving embeddings.

Type semantics The type semantics are simply the functions $[\![A]\!]^{\mathcal{S}}_{\Gamma} = [\Gamma]^* \to [A]$, which means that for a given term $t : \mathrm{Tm}\,\Gamma\,A$ we have $\langle t \rangle : [\![A]\!]^{\mathcal{S}}_{\Gamma}$.

Transport Transport is implemented by composing with the semantic action of weakening (here we are leaving out the superscript for typographical reasons)

$$\begin{aligned} \_ \nearrow \_ &: [\![A]\!]^{\mathcal{S}}_{\Gamma} \to \Gamma \subseteq \Delta \to [\![A]\!]^{\mathcal{S}}_{\Delta} \\ x \nearrow i &= x \circ \langle i \rangle^{\mathrm{weak}}, \end{aligned}$$

where

$$\begin{aligned} \langle \_ \rangle^{\mathrm{weak}} &: \forall\{\Gamma\,\Delta\} \to \Gamma \subseteq \Delta \to [\Delta]^* \to [\Gamma]^* \\ \langle \mathrm{refl}_{\Gamma} \rangle^{\mathrm{weak}}\,\gamma &= \gamma \\ \langle \downarrow^{\mathrm{weak}}\ i \rangle^{\mathrm{weak}}\,(\delta, a) &= \langle i \rangle^{\mathrm{weak}}\,\delta \\ \langle \uparrow^{\mathrm{weak}}\ i \rangle^{\mathrm{weak}}\,(\delta, a) &= (\langle i \rangle^{\mathrm{weak}}\,\delta, a) \end{aligned}$$

Application Kripkefied application is given by $\mathrm{apply}^{\mathcal{S}}\,f\,x = \lambda\gamma \to f\,\gamma\,(x\,\gamma)$.

Term semantics The term semantics is defined as $(\!|t|\!)^{\mathcal{S}}\,\eta = \langle t\rangle \circ \langle \eta\rangle^*$, where $\langle\eta\rangle^*$ casts the environment η into a function on contexts:

$$\begin{aligned} &\langle \_\rangle^* : \forall\{\Gamma\,\Delta\} \to [\![\Gamma]\!]^{*\mathcal{S}}_{\Delta} \to [\Delta]^* \to [\Gamma]^* \\ \{\Gamma = \emptyset\} \quad &\langle f\rangle^* = \lambda\delta \to f \\ \{\Gamma = (\Gamma, A)\} \quad &\langle \eta, \alpha\rangle^* = \lambda\delta \to (\langle\eta\rangle^*\,\delta, \alpha\,\delta) \end{aligned}$$

5.2 Definability as a Kripke Predicate

The NbE Kripke model from Section 4.2 gives rise to the following Kripke predicate on the Kripkefied standard semantics:

$$\begin{aligned} \mathcal{N}[\![\_]\!]_{\_} &: (A : \mathrm{Ty}) \to (\Gamma : \mathrm{Cxt}) \to [\![A]\!]^{\mathcal{S}}_{\Gamma} \to \mathrm{Set} \\ x \in \mathcal{N}[\![\iota]\!]_{\Gamma} &= \Sigma(t \in \mathrm{Ne}\,\Gamma\,\iota)(x = \langle t\rangle) \\ f \in \mathcal{N}[\![A \Rightarrow B]\!]_{\Gamma} &= \forall\Delta\,(i : \Gamma \subseteq \Delta)(x \in \mathcal{N}[\![A]\!]_{\Delta}) \to \mathrm{apply}^{\mathcal{S}}\,(f \nearrow i)\,x \in \mathcal{N}[\![B]\!]_{\Delta} \end{aligned}$$

Into this 'sub-model' we can again 'reflect' neutral terms, as well as 'reify' normal forms:

$$\begin{aligned} \mathrm{reflect} &: \forall\{\Gamma\,A\} \to (t \in \mathrm{Ne}\,\Gamma\,A) \to \langle t\rangle \in \mathcal{N}[\![A]\!]_{\Gamma} \\ \mathrm{reify} &: \forall\{\Gamma\,A\} \to x \in \mathcal{N}[\![A]\!]_{\Gamma} \to \Sigma(t \in \mathrm{Nf}\,\Gamma\,A)(x = \langle t\rangle) \end{aligned}$$

Given a term $t : \mathrm{Tm}\,\Gamma\,A$ the fundamental lemma of Kripke predicates states that $(\!|t|\!)^{\mathcal{S}}\,\eta \in \mathcal{N}[\![A]\!]_{\Delta}$ whenever $\eta \in \mathcal{N}[\![\Gamma]\!]^*_{\Delta}$. Since $(\!|t|\!)^{\mathcal{S}}\,\eta = \langle t\rangle \circ \langle\eta\rangle^*$ by definition and since $\mathrm{fresh}_{\Gamma} \in \mathcal{N}[\![\Gamma]\!]^*_{\Gamma}$ with $\langle \mathrm{fresh}_{\Gamma}\rangle^* = \mathrm{id}$ we have that for any $t : \mathrm{Tm}\,\Gamma\,A$, $\langle t\rangle \in \mathcal{N}[\![A]\!]_{\Gamma}$ (the definition of fresh_{Γ} is left out). This means that we obtain a *correct-by-construction* normalisation function as follows:

$$\begin{aligned} &\mathrm{norm} : \forall\{\Gamma\,A\} \to (t : \mathrm{Tm}\,\Gamma\,A) \to \Sigma(t' \in \mathrm{Nf}\,\Gamma\,A)(\langle t\rangle = \langle t'\rangle) \\ &\mathrm{norm} = \mathrm{reify}_A\,((\!|t|\!)^{\mathcal{S}}\,\mathrm{fresh}_{\Gamma}) \end{aligned}$$

5.3 STLC-Definability Theorem

Using the NbE predicate we can prove that STLC-definability is in fact equivalent to 'satisfying all Kripke predicates'. Note again that this theorem is a slightly restricted version of the one given by Jung and Tiuryn [17]. The LF-definability theorem that we are going to arrive at in Chapter 7 will mirror this one.

Theorem. *An object $x : [A]$ at $A : Ty$ is definable in the simply-typed λ-calculus iff it satisfies all Kripke predicates. In other words, there exists a $t : \mathrm{Tm}\,\varnothing\,A$ such that $(\lambda\_ \to x) = \langle t\rangle$ iff for every Kripke predicate $\mathcal{P}$ (on the model defined in Section 5.1) we have $(\lambda\_ \to x) \in \mathcal{P}[\![A]\!]_{\varnothing}$.*

Proof. (Sketch) In one direction, if x is defined by some term t then $\langle t\rangle \in \mathcal{P}[\![A]\!]_{\varnothing}$ follows directly from the 'fundamental lemma' of Kripke predicates. In the other direction, if x satisfies all Kripke predicates it also satisfies the NbE predicate; reify then gives us a term defining x. □

Part II

LF Definability

Chapter 6

The Edinburgh Logical Framework (LF)

The LF system [15] consists of three levels: *kinds*, *type families* and *terms*. Objects at each of these levels are indexed by *contexts*, which are lists of term variables. This makes LF a *dependently typed* λ-calculus—LF's types *depend on* (are indexed by) *values* (terms). Note that as far as dependently typed λ-calculi are concerned, LF (corresponding to $\lambda\Pi$ in Barendregt's λ-cube [6]) is not very powerful: it does for example not allow terms to depend on types, or types to depend on other types.

This chapter is structured as follows: in the first section we summarise a somewhat standard formulation of LF, roughly following Harper, Honsell and Plotkin [15]. We will then use this more standard presentation as motiviation in defining another, non-standard variant of LF in Section 6.2. Throughout the rest of the thesis we will be working with this second, non-standard presentation of LF.

6.1 An Intrinsically Typed Sketch of LF

In this section we have a brief look at the Edinburgh Logical Framework following its original presentation by Harper, Honsell and Plotkin. Amongst other differences, we are going to ignore any technical issues concerning type equality (and the corresponding conversion rules). We are also going to *pretend* we can make the syntax intrinsically typed as this makes the definitions much easier to follow—we are not claiming this can be made formal![1]

Recall that the Edinburgh Logical Framework consists of objects at three levels: *kinds*, *type families* and *terms*. We can think of these as inductive data types

$$\mathrm{Kind} : \mathrm{Cxt} \to \mathrm{Set}, \qquad \mathrm{Fam} : (\Gamma : \mathrm{Cxt}) \to \mathrm{Kind}\,\Gamma \to \mathrm{Set}$$

$$\text{and} \quad \mathrm{Tm} : (\Gamma : \mathrm{Cxt}) \to \mathrm{Fam}\,\Gamma\,\mathrm{Type} \to \mathrm{Set},$$

[1] Author's note: I was struggling to get an intuition for the LF calculus as presented in Harper et al. [15], and found that adding types in an informal sketch helped a lot since it made the structure of the calculus more obvious.

respectively, where Cxt : Set is the set of *contexts* (or telescopes) of term variables.

Kinds The kinds of LF are rather restricted; in particular, we do not have abstraction or application at the level of kinds.

$$\frac{\Gamma : \mathrm{Cxt}}{\mathrm{Type} : \mathrm{Kind}\,\Gamma} \qquad \frac{A : \mathrm{Fam}\,\Gamma\,\mathrm{Type} \quad K : \mathrm{Kind}\,(\Gamma, A)}{\mathrm{Pi}\,A\,K : \mathrm{Kind}\,\Gamma}$$

Note that all kinds are of the form $\mathrm{Pi}\,A_1\,(\ldots(\mathrm{Pi}\,A_n\,\mathrm{Type})\ldots)$, where the A_i are type families.

Type families At type level we do have abstraction and application. Let $K/_x\,t$ denote the substitution (where the subscript x is part of the operator name and *not* a variable) of the term t for the first free variable (i.e. de Bruijn index 0) in the kind K.

$$\frac{K : \mathrm{Kind}\,(\Gamma, B) \quad A : \mathrm{Fam}\,(\Gamma, B)\,K}{\mathrm{Abs}\,A : \mathrm{Fam}\,\Gamma\,(\mathrm{Pi}\,B\,K)} \qquad \frac{A : \mathrm{Fam}\,\Gamma\,(\mathrm{Pi}\,B\,K) \quad t : \mathrm{Tm}\,\Gamma\,B}{\mathrm{App}\,A\,t : \mathrm{Fam}\,\Gamma\,(K/_x\,t)}$$

Note that we don't bother to define the substitution operator in this sketch – our intuition shall suffice.

The constructor for function types in LF gives rise to *dependent* functions since the return type B is allowed to depend (in its context) on the value of the argument.

$$\frac{A : \mathrm{Fam}\,\Gamma\,\mathrm{Type} \quad B : \mathrm{Fam}\,(\Gamma, A)\,\mathrm{Type}}{\mathrm{pi}\,A\,B : \mathrm{Fam}\,\Gamma\,\mathrm{Type}}$$

Terms The terms of LF are given by the usual triad of abstraction, application and variables. Let $B/_x\,u$ denote substituting the term u for the first free variable (which has type A) in the type B.

$$\frac{t : \mathrm{Tm}\,(\Gamma, A)\,B}{\mathrm{abs}\,t : \mathrm{Tm}\,\Gamma\,(\mathrm{pi}\,A\,B)} \qquad \frac{t : \mathrm{Tm}\,\Gamma\,(\mathrm{pi}\,A\,B) \quad u : \mathrm{Tm}\,\Gamma\,A}{\mathrm{app}\,t\,u : \mathrm{Tm}\,\Gamma\,(B/_x\,u)}$$

Term variables are given by indices into the context,

$$\frac{v : \mathrm{Var}\,\Gamma\,A}{\mathrm{var}\,v : \mathrm{Tm}\,\Gamma\,A}$$

defined using the auxiliary notion $\mathrm{Var} : (\Gamma : \mathrm{Cxt}) \to \mathrm{Fam}\,\Gamma\,\mathrm{Type} \to \mathrm{Set}$.

$$\frac{A : \mathrm{Fam}\,\Gamma\,\mathrm{Type}}{\mathrm{vzero} : \mathrm{Var}\,(\Gamma, A)\,A{\downarrow}} \qquad \frac{v : \mathrm{Var}\,\Gamma\,A \quad B : \mathrm{Fam}\,\Gamma\,\mathrm{Type}}{\mathrm{vsuc}\,v : \mathrm{Var}\,(\Gamma, B)\,A{\downarrow}}$$

Here we let $A{\downarrow} : \mathrm{Fam}\,(\Gamma, A)\,\mathrm{Type}$ denote the appropriately *weakened* $A : \mathrm{Fam}\,\Gamma$, where all references to variables in Γ are replaced by the corresponding variables in (Γ, A), shifting the de Bruijn indices by one. The definition is again left out in this sketch.

Constants The calculus as presented above is not particularly useful—so far there aren't even any types! In order to be able to remedy this situation we would want to be able to inject constants into our syntax, both at type as well as at term level.

We parameterise our calculus over a set of *type constants* Const : Set together with their kinds $\kappa : \text{Const} \to \text{Kind}\,\emptyset$. These we embed into our calculus with a corresponding type constructor. ($\kappa(C)\downarrow$ denotes the kind of the constant, appropriately weakened to fit the context.)

$$\frac{C : \text{Const} \quad \Gamma : \text{Cxt}}{\text{Con}\,C : \text{Fam}\,\Gamma\,\kappa(C)\downarrow}$$

In order to have any types at all one would probably want a 'unit' type, corresponding to Agda's $\top$: Set. We can add such a type by defining a name Unit : Const and choosing $\kappa(\text{Unit}) = \text{Type}$; this allows us to form types like pi (Con Unit) (Con Unit) corresponding to functions $\top \to \top$.

To encode *term constants* we similarly parameterise over a set const : Set with typing information $\tau : \text{const} \to \text{Fam}\,\emptyset\,\text{Type}$ and add a corresponding term constructor.

$$\frac{c : \text{const} \quad \Gamma : \text{Cxt}}{\text{con}\,c : \text{Tm}\,\Gamma\,\tau(c)\downarrow}$$

To inhabit our unit type we can now add a term constant unit : const with $\tau(\text{unit}) = \text{Con Unit}$.

Example. To motivate how we could make use of the *dependent* types in this calculus, let us try to express natural numbers and vectors (i.e. lists indexed by their lengths) of natural numbers through appropriate constants.

To embed natural numbers into the calculus we need a type constant Nat : Const and two term constants zero and suc : const, along with the following typing information: $\kappa(\text{Nat}) = \text{Type}$, $\tau(\text{zero}) = \text{Con Nat}$ and $\tau(\text{suc}) = \text{Pi (Con Nat) (Con Nat)}$.

To encode vectors we need a type family constant Vec : Const and term constants nil and cons : const. We set $\kappa(\text{Vec}) = \text{Pi (Con Nat) Type}$, corresponding to $\text{Vec} : \mathbb{N} \to \text{Set}$ in our meta-theory. Similarly, $\tau(\text{nil}) = \text{App (Con Vec) (con zero)}$ corresponds to $\text{nil} : \text{Vec}\,0$. Finally, the type of cons, corresponding to $\text{cons} : (n : \mathbb{N}) \to \mathbb{N} \to \text{Vec}\,n \to \text{Vec}\,(n+1)$, is:

$$\begin{aligned}\tau(\text{cons}) = \text{pi (Con Nat) (pi (Con Nat) (pi (App (Con Vec) (var (vsuc vzero})))}\\ \text{(App (Con Vec) (app (con suc) (var (vsuc (vsuc vzero)))))))}\end{aligned}$$

After adding these constants we could now write down LF types corresponding to the Agda types $\mathbb{N}$ and $\text{Vec}\,42$, as well as terms inhabiting those types.

6.2 A Semi-Semantic Variant of LF

In this section we will define the 'semi-semantic' variant of LF that we are going to base our LF-definability theorem in Chapter 7 on. The reason for us to work with this non-standard variant is that it will allow us to translate the constructions of Jung and Tiuryn, giving rise to an LF-definability theorem following the same recipe as their STLC-definability theorem. The key property

of our LF variant that enables this correspondence is the fact that we define types such that *convertible* (i.e. computationally equivalent) types will already be *equal*, that is, the representation is sufficiently semantic that type equality coincides with the meta-equality of our type theory.[2]

The contents of this section (excluding the introductory exposition up to and incl. Section 6.2.1) are to be read as formal definitions, and are to be considered in parallel with the corresponding Agda code. Some of the more technical details are left entirely for the Agda formalisation, and are only referred to in this section. Note that in this written document we are going to work in an extensional meta-theory; as a result, the following definitions are somewhat more readable than the corresponding Agda code (for example, we get to ignore all the transports that type equalities introduce).

Getting rid of kinds Before we get into the technical definitions of our calculus, let us motivate the key change that we are going to make compared to a standard presentation of LF: getting rid of the book-keeping level of *kinds* entirely. Considering the standard presentation of LF in the previous section, what would the denotations of kinds, type families and terms look like? We would presumably end up with something like the following:

$$
\begin{aligned}
[\_]^{*} &: \mathrm{Cxt} \to \mathrm{Set} \\
[\_]^{\mathrm{Kind}} &: \{\Gamma : \mathrm{Cxt}\} \to \mathrm{Kind}\,\Gamma \to [\Gamma]^{*} \to \mathrm{Set}_1 \\
[\_] &: \{\Gamma : \mathrm{Cxt}\} \to \{K : \mathrm{Kind}\,\Gamma\} \to \mathrm{Fam}\,\Gamma\,K \to (\gamma : [\Gamma]^{*}) \to [K]^{\mathrm{Kind}}\,\gamma \\
\langle\_\rangle &: \{\Gamma : \mathrm{Cxt}\} \to \mathrm{Tm}\,\Gamma\,A \to (\gamma : [\Gamma]^{*}) \to [A]\,\gamma
\end{aligned}
$$

Remember now the remark we made earlier, that the LF kinds are all of the form $\mathrm{Pi}\,A_1\,(\dots(\mathrm{Pi}\,A_n\,\mathrm{Type})\dots)$, where the A_i are type families. Their denotation are therefore functions of the shape $\mathcal{T} \to \mathrm{Set}$, where $\mathcal{T}$ corresponds to the context given by the sequence of arguments $A_1, \dots, A_n$. In fact, one can see that the denotation of such a kind is going to be *isomorphic* (via currying) to a function $[\Delta]^{*} \to \mathrm{Set}$, where $\Delta = (\dots(\emptyset, A_1)\dots, A_n)$ *is* the context corresponding to the arguments.

Using the above observation we can get away with only dealing with type families of kind Type, that is, we don't need to define syntax for any of the $\mathrm{Fam}\,\Gamma\,K$ with $K \neq \mathrm{Type}$. We can encode fully applied type family constants by carrying around a (semantic) substitution into their index sets leading to the following type former:

$$\frac{C : \mathrm{Const} \quad \sigma : [\Gamma]^{*} \to \mathrm{Idx}\,C}{\mathrm{dat}\,C\,\sigma : \mathrm{Fam}\,\Gamma\,\mathrm{Type}}$$

To close this sketch, consider the example of natural numbers and vectors we looked at previously: in this new formulation, the constant type family $\mathbb{N} : \mathrm{Set}$ can be encoded by $\mathrm{Nat} : \mathrm{Const}$ by setting $\mathrm{Idx}\,\mathrm{Nat} = \top$. The equality of our meta-theory now ensures that any two occurrences of Nat in the same context are propositionally equal in the meta-theory, that is we have $\mathrm{dat}\,\mathrm{Nat}\,\sigma = \mathrm{dat}\,\mathrm{Nat}\,\sigma'$,

[2] Our approach shares some similarities with McBride's 'Outrageous but Meaningful Coincidences' [19], though this only became clear in hindsight.

since necessarily $\sigma = \sigma'$ (since $\top$ is the terminal type). Similarly, for Vec : Const we now set $\mathrm{Idx}\,\mathrm{Vec} = \mathbb{N}$, which is precisely the index set of the corresponding Agda type $\mathrm{Vec} : \mathbb{N} \to \mathrm{Set}$.

6.2.1 Overview

In order to not lose track of the bigger picture amongst all the technical details that are going to follow, let us start by enumerating the key notions that we are going to define.

Contexts We need to define contexts and their denotations. We will use the symbols Γ, Δ and Ω for context variables.

$$\mathrm{Cxt} : \mathrm{Set} \qquad [\_]^* : \mathrm{Cxt} \to \mathrm{Set}$$

Types We also need to define type families and their denotations. We will use A and B for types. (Note the lack of kinds.)

$$\mathrm{Fam} : \mathrm{Cxt} \to \mathrm{Set} \qquad [\_] : \mathrm{Fam}\,\Gamma \to [\Gamma]^* \to \mathrm{Set}$$

Terms We will define terms and their denotations. We will use t and u for terms.

$$\mathrm{Tm} : (\Gamma : \mathrm{Cxt}) \to \mathrm{Fam}\,\Gamma \to \mathrm{Set} \qquad \langle\_\rangle : \mathrm{Tm}\,\Gamma\,A \to (\gamma : [\Gamma]^*) \to [A]\,\gamma$$

Terms will be defined using the auxiliary notion of variables, with corresponding denotation.

$$\mathrm{Var} : (\Gamma : \mathrm{Cxt}) \to \mathrm{Fam}\,\Gamma \to \mathrm{Set} \qquad \langle\_\rangle^{\mathrm{var}} : \mathrm{Var}\,\Gamma\,A \to (\gamma : [\Gamma]^*) \to [A]\,\gamma$$

Type Constants As motivated earlier, type family constants are encoded using a set of names and semantic index sets.

$$\mathrm{Const} : \mathrm{Set} \qquad \mathrm{Idx} : \mathrm{Const} \to \mathrm{Set} \qquad [\_]^{\mathrm{Const}} : (C : \mathrm{Const}) \to \mathrm{Idx}\,C \to \mathrm{Set}$$

They are going to be embedded into the syntax of types with a corresponding type former:

$$\frac{C : \mathrm{Const} \quad \sigma : [\Gamma]^* \to \mathrm{Idx}\,C}{\mathrm{dat}\,C\,\sigma : \mathrm{Fam}\,\Gamma}$$

Term Constants Term level constants are given by names and types, together with their denotations,

$$\mathrm{const} : \mathrm{Set} \qquad \tau : \mathrm{const} \to \mathrm{Fam}\,\emptyset \qquad \langle\_\rangle^{\mathrm{const}} : (c : \mathrm{const}) \to [\tau(c)]$$

as well as a corresponding term former.

$$\frac{c : \mathrm{const} \quad \Gamma : \mathrm{Cxt}}{\mathrm{con}\,c : \mathrm{Tm}\,\Gamma\,\tau(c)\downarrow}$$

6.2.2 Contexts and Type Families

The Agda code corresponding to the definitions in this subsection can be found in Types.agda. Note that we believe the following inductive-recursive definitions to be well-defined (and Agda 2.6 thinks so as well), but we do not prove this formally.

We start by assuming a set of names for type-level constants together with their index sets and denotations.[3]

$$\mathrm{Const} : \mathrm{Set} \qquad \mathrm{Idx} : \mathrm{Const} \to \mathrm{Set} \qquad [\_]^{\mathrm{Const}} : (C : \mathrm{Const}) \to \mathrm{Idx}\, C \to \mathrm{Set}$$

We define contexts $\mathrm{Cxt} : \mathrm{Set}$ and type families $\mathrm{Fam} : \mathrm{Cxt} \to \mathrm{Set}$ mutually inductively along with their denotations in an inductive-recursive definition:

$$\mathrm{Cxt} : \mathrm{Set} \qquad \frac{}{\emptyset : \mathrm{Cxt}} \qquad \frac{\Gamma : \mathrm{Cxt} \quad A : \mathrm{Fam}\,\Gamma}{(\Gamma, A) : \mathrm{Cxt}}$$

$$\mathrm{Fam} : \mathrm{Cxt} \to \mathrm{Set} \qquad \frac{A : \mathrm{Fam}\,\Gamma \quad B : \mathrm{Fam}\,(\Gamma, A)}{\mathrm{pi}\,A\,B : \mathrm{Fam}\,\Gamma} \qquad \frac{C : \mathrm{Const} \quad \sigma : [\Gamma]^* \to \mathrm{Idx}\, C}{\mathrm{dat}\, C\, \sigma : \mathrm{Fam}\,\Gamma}$$

$$[\_]^* : \mathrm{Cxt} \to \mathrm{Set} \qquad\qquad [\_] : \forall\{\Gamma\} \to \mathrm{Fam}\,\Gamma \to [\Gamma]^* \to \mathrm{Set}$$
$$[\emptyset]^* = \top \qquad\qquad [\mathrm{pi}\,A\,B]\,\gamma = (a : [A]\,\gamma) \to [B]\,(\gamma, a)$$
$$[\Gamma, A]^* = \Sigma(\gamma : [\Gamma]^*)([A]\,\gamma) \qquad [\mathrm{dat}\, C\, \sigma]\,\gamma = [C]^{\mathrm{Const}}\,(\sigma\,\gamma)$$

6.2.3 Weakening and Subtitutions in Types

In order to define the terms of our calculus we will already need to use the notions of *weakening* and *substitution* (in types). To motivate this, consider a function $t : A \Rightarrow B$ and an argument $u : A$ in the simply-typed case; the type of the application of $t\,u$ is simply $t\,u : B$. In the case of LF we have $t : \mathrm{pi}\,A\,B$ and $u : A$; since the result type is allowed to depend on the value of the argument we need to propagate the applied argument into the type! More specifically, to obtain the resulting type of $t\,u$ we need to substitute any occurrence of the top-most variable in B by u.

Substitution The Agda code corresponding to the following definitions and lemmas can be found in Substitution.agda.

In our semi-semantic formulation of LF the appropriate notion of type substitution is wholly semantic. That is, we express a substitution of the open variables in a type $A : \mathrm{Fam}\,\Gamma$ via semantic maps $\sigma : [\Delta]^* \to [\Gamma]^*$, yielding a resulting type $A/\sigma : \mathrm{Fam}\,\Delta$. In the special case where $\Delta = \emptyset$ we can think of $A/\sigma : \mathrm{Fam}\,\emptyset$ being an *instance* of the type family A.

We define the action of substituting in types recursively,

[3] Note that instead of parameterising the theory over the set of constants we chose to postulate them in the Agda development. The reason for this is that later, when we add term constants to the mix, we would have ended up with a mess of stacked modules had we gone down the route of module parameters: the types of those later parameters depend on previous constructions, that in turn depend on other parameters. (Author's opinion) Using postulates, while stylistically questionable, ended up being significantly easier to work with.

$$
\begin{aligned}
\_/\_ &: \forall\{\Gamma\,\Delta\} \to \mathrm{Fam}\,\Gamma \to ([\Delta]^* \to [\Gamma]^*) \to \mathrm{Fam}\,\Delta \\
(\mathrm{pi}\,A\,B)/\sigma &= \mathrm{pi}\,(A/\sigma)\,(B/\mathrm{lift}\,\sigma) \\
(\mathrm{dat}\,C\,\sigma')/\sigma &= \mathrm{dat}\,C\,(\sigma' \circ \sigma)
\end{aligned}
$$

where $\mathrm{lift}\,\sigma = \lambda(\delta, a) \to (\sigma\,\delta, a) : [\Delta, A/\sigma]^* \to [\Gamma, A]^*$ *lifts* the substitution. Note that intensionally we only have $\lambda(\delta, a) \to (\sigma\,\delta, a) : \Sigma(\delta : [\Delta]^*)([A]\,(\sigma\,\delta)) \to [\Gamma, A]^*$; to witness the fact that these types are extensionally equivalent we need the following substitution lemma.

Lemma (sub-den)**.** *For every type family* $A : \mathrm{Fam}\,\Gamma$ *and substitution* $\sigma : [\Delta]^* \to [\Gamma]^*$ *we have* $[A/\sigma] = [A] \circ \sigma$*.*

Proof. In Agda. □

Two additional facts about substitutions that we will need later are the fact that they compose and that the identity substitution behaves like one would expect.

Lemma (sub-comp)**.** *For* $A : \mathrm{Fam}\,\Gamma$*,* $\sigma : [\Delta]^* \to [\Gamma]^*$ *and* $\sigma' : [\Omega]^* \to [\Delta]^*$ *we have that* $(A/\sigma)/\sigma' = A/(\sigma \circ \sigma')$*.*

Proof. In Agda. □

Lemma (sub-id)**.** *For* $A : \mathrm{Fam}\,\Gamma$ *we have that* $A/id = A$*.*

Proof. In Agda. □

Weakening The Agda code corresponding to the following definitions and lemmas can be found in Weakening.agda. This file contains a use of the `TERMINATING` pragma, which is justified by a termination proof in the appendix in Section 9.

The fact that we need to deal with weakening at all is an artefact of our types and terms being intrinsically *well-scoped.* In an unscoped representation, a term (or type) t can be considered in any environment that assigns a meaning to all of its free variables—in particular, we can interpret t in any 'larger' environment that mentions more variables than just those appearing free in t!

In our case, in order to make sense of a type $A : \mathrm{Fam}\,\Gamma$ in a 'bigger' context $\Delta \supseteq \Gamma$ (for some notion of 'bigger'), we need to explicitly transform (i.e. *weaken*) the type into a corresponding type $A' : \mathrm{Fam}\,\Delta$. As far as comparing contexts goes, it turns out that for our purposes it suffices to consider Δ as 'bigger than' Γ, iff there exists an *order-preserving embedding* of Γ into Δ, that is we define a binary predicate $\_ \subseteq \_ : \mathrm{Cxt} \to \mathrm{Cxt} \to \mathrm{Set}$ as follows. (We will later drop the superscript $^{\mathrm{weak}}$ when it is clear from the context what is meant.)

$$
\frac{}{\mathrm{refl}^{\mathrm{weak}} : \Gamma \subseteq \Gamma}
\qquad
\frac{i : \Gamma \subseteq \Delta \quad A : \mathrm{Fam}\,\Delta}{\downarrow^{\mathrm{weak}}\ i : \Gamma \subseteq (\Delta, A)}
\qquad
\frac{i : \Gamma \subseteq \Delta \quad A : \mathrm{Fam}\,\Gamma}{\uparrow^{\mathrm{weak}}\ i : (\Gamma, A) \subseteq (\Delta, A{\downarrow_i})}
$$

This definition depends on the mutually defined action of weakening along such an embedding

$$
\begin{aligned}
\_{\downarrow_\_} &: \forall\{\Gamma\,\Delta\} \to \mathrm{Fam}\,\Gamma \to \Gamma \subseteq \Delta \to \mathrm{Fam}\,\Delta \\
(\mathrm{pi}\,A\,B){\downarrow_i} &= \mathrm{pi}\,(A{\downarrow_i})\,(B{\downarrow_{(\uparrow^{\mathrm{weak}}\,i)}}) \\
(\mathrm{dat}\,C\,\sigma){\downarrow_i} &= \mathrm{dat}\,C\,(\sigma \circ \langle i \rangle^{\mathrm{weak}})
\end{aligned}
$$

which in turn depends on the semantic action of embeddings, which are substitutions.

$$\begin{aligned}
\langle _ \rangle^{\text{weak}} &: \forall\{\Gamma\,\Delta\} \to \Gamma \subseteq \Delta \to [\Delta]^* \to [\Gamma]^* \\
\langle \text{refl}^{\text{weak}} \rangle^{\text{weak}} &= \text{id} \\
\langle \downarrow^{\text{weak}}\ i \rangle^{\text{weak}} &= \langle i \rangle^{\text{weak}} \circ \pi_1 \\
\langle \uparrow^{\text{weak}}\ i \rangle^{\text{weak}} &= \text{lift}\ \langle i \rangle^{\text{weak}}
\end{aligned}$$

The following lemma proves that weakening is a special case of substitution.

Lemma (weak-sub)**.** *For $A : \text{Fam}\,\Gamma$ and $i : \Gamma \subseteq \Delta$ we have that $A{\downarrow_i} = A/\langle i \rangle^{weak}$.*

Proof. In Agda. □

We will later also use the following weakening lemma which follows directly from the corresponding substitution lemma.

Lemma (weak-den)**.** *For every type $A : \text{Fam}\,\Gamma$ and embedding $i : \Gamma \subseteq \Delta$ we have $[A{\downarrow_i}] = [A] \circ \langle i \rangle^{weak}$.*

Proof. In Agda. □

We can compose embeddings; this witnesses the transitivity of $\_ \subseteq \_$.

$$\begin{aligned}
_ \cdot _ &: \forall\{\Gamma\,\Delta\,\Omega\} \to \Gamma \subseteq \Delta \to \Delta \subseteq \Omega \to \Gamma \subseteq \Omega \\
i \cdot \text{refl}^{\text{weak}} &= i \\
i \cdot \downarrow^{\text{weak}}\ i' &= \downarrow^{\text{weak}}\ (i \cdot i') \\
\text{refl}^{\text{weak}} \cdot \uparrow^{\text{weak}}\ i &= \uparrow^{\text{weak}}\ i \\
\downarrow^{\text{weak}}\ i \cdot \uparrow^{\text{weak}}\ i' &= \downarrow^{\text{weak}}\ (i \cdot i') \\
\uparrow^{\text{weak}}\ i \cdot \uparrow^{\text{weak}}\ i' &= \uparrow^{\text{weak}}\ (i \cdot i')
\end{aligned}$$

Note that the very last clause is not intensionally well-typed. The left-hand side has type $(\Gamma, A) \subseteq (\Omega, (A{\downarrow_i}){\downarrow_{i'}})$ while the right-hand side is of type $(\Gamma, A) \subseteq (\Omega, A{\downarrow_{i \cdot i'}})$. The fact that these types are extensionally compatible is witnessed by the following lemmas.

Lemma (weak-comp-den)**.** *Given $i : \Gamma \subseteq \Delta$ and $i' : \Delta \subseteq \Omega$ we have $\langle i \cdot i' \rangle^{weak} = \langle i \rangle^{weak} \circ \langle i' \rangle^{weak}$*

Proof. In Agda. □

Lemma (weak-comp)**.** *Given $A : \text{Fam}\,\Gamma$, $i : \Gamma \subseteq \Delta$ and $i' : \Delta \subseteq \Omega$ we have $(A{\downarrow_i}){\downarrow_{i'}} = A{\downarrow_{i \cdot i'}}$.*

Proof. In Agda. □

Finally, we also observe that composition with the identity embedding on the left does nothing.

Lemma (weak-id-comp)**.** *For any $i : \Gamma \subseteq \Delta$, $\text{refl}^{\text{weak}} \cdot i = i$.*

Proof. In Agda. □

6.2.4 Variables and Terms

The Agda code corresponding to the definitions and lemmas in this section can be found in Terms.agda.

Variables are simply de Bruijn indices, and their denotations projections.

$$\mathrm{Var} : (\Gamma : \mathrm{Cxt}) \to \mathrm{Fam}\,\Gamma \to \mathrm{Set}$$

$$\frac{A : \mathrm{Fam}\,\Gamma}{\mathrm{vzero} : \mathrm{Var}\,(\Gamma, A)\,A{\downarrow_{\mathrm{refl}}}} \qquad \frac{v : \mathrm{Var}\,\Gamma\,A \quad B : \mathrm{Fam}\,\Gamma}{\mathrm{vsuc}\,v : \mathrm{Var}\,(\Gamma, B)\,A{\downarrow_{\mathrm{refl}}}}$$

$$\begin{aligned}\langle \_ \rangle^{\mathrm{var}} &: \forall\{\Gamma\,A\} \to \mathrm{Var}\,\Gamma\,A \to (\gamma : [\Gamma]^*) \to [A]\,\gamma\\ \langle \mathrm{vzero} \rangle^{\mathrm{var}}\,(\gamma, a) &= a\\ \langle \mathrm{vsuc}\,v \rangle^{\mathrm{var}}\,(\gamma, b) &= \langle v \rangle^{\mathrm{var}}\,\gamma\end{aligned}$$

We assume a set of *term constants* together with their types and denotations. Note that $\mathrm{tt} : \top$ is the empty environment ($\top = [\varnothing]^*$).

$$\mathrm{const} : \mathrm{Set} \qquad \tau : \mathrm{const} \to \mathrm{Fam}\,\varnothing \qquad \langle \_ \rangle^{\mathrm{con}} : (c : \mathrm{const}) \to [\tau(c)]\,\mathrm{tt}$$

Let $\epsilon_\Gamma = {\downarrow} \ldots {\downarrow}\,\mathrm{refl} : \varnothing \subseteq \Gamma$ be the embedding of the empty context in any other context. We define *terms* mutually with their denotation.

$$\mathrm{Tm} : (\Gamma : \mathrm{Cxt}) \to \mathrm{Fam}\,\Gamma \to \mathrm{Set}$$

$$\frac{c : \mathrm{const} \quad \Gamma : \mathrm{Cxt}}{\mathrm{con}\,c : \mathrm{Tm}\,\Gamma\,\tau(c){\downarrow_{\epsilon_\Gamma}}} \qquad \frac{v : \mathrm{Var}\,\Gamma\,A}{\mathrm{var}\,v : \mathrm{Tm}\,\Gamma\,A}$$

$$\frac{t : \mathrm{Tm}\,(\Gamma, A)\,B}{\mathrm{abs}\,t : \mathrm{Tm}\,\Gamma\,(\mathrm{pi}\,A\,B)} \qquad \frac{t : \mathrm{Tm}\,\Gamma\,(\mathrm{pi}\,A\,B) \quad u : \mathrm{Tm}\,\Gamma\,A}{\mathrm{app}\,t\,u : \mathrm{Tm}\,\Gamma\,(B/_x\langle u \rangle)}$$

Here $B/_x\langle u \rangle = B/(\lambda\gamma \to (\gamma, \langle u \rangle\,\gamma))$ is the result of substituting the top-most variable in B by the denotation of u.

$$\begin{aligned}\langle \_ \rangle &: \forall\{\Gamma\,A\} \to \mathrm{Tm}\,\Gamma\,A \to (\gamma : [\Gamma]^*) \to [A]\,\gamma\\ \langle \mathrm{con}\,c \rangle\,\gamma &= \langle c \rangle^{\mathrm{con}}\\ \langle \mathrm{var}\,v \rangle\,\gamma &= \langle v \rangle^{\mathrm{var}}\,\gamma\\ \langle \mathrm{abs}\,t \rangle\,\gamma &= \lambda a \to \langle t \rangle\,(\gamma, a)\\ \langle \mathrm{app}\,t\,u \rangle\,\gamma &= \langle t \rangle\,\gamma\,(\langle u \rangle\,\gamma)\end{aligned}$$

6.2.5 Weakening of Variables and Terms

The Agda code corresponding to the definitions and lemmas in this section can be found in TermWeakening.agda.

In addition to the weakening of types we will later also need to weaken terms, i.e. given an embedding $i : \Gamma \subseteq \Delta$ and $t : \mathrm{Tm}\,\Gamma\,A$ construct the corresponding $t{\downarrow_i} : \mathrm{Tm}\,\Delta\,A{\downarrow_i}$.

Since variables are represented syntactically in terms, we first need to define a corresponding reindexing of the de Bruijn indices.

$$
\begin{aligned}
\_{\downarrow}\_ &: \forall\{\Gamma\,\Delta\,A\} \to \mathrm{Var}\,\Gamma\,A \to (i : \Gamma \subseteq \Delta) \to \mathrm{Var}\,\Delta\,A{\downarrow_i} \\
v{\downarrow_{\mathrm{refl}}} &= v \\
v{\downarrow_{\downarrow i}} &= \mathrm{vsuc}\,(v{\downarrow_i}) \\
\mathrm{vzero}{\downarrow_{\uparrow i}} &= \mathrm{vzero} \\
(\mathrm{vsuc}\,v){\downarrow_{\uparrow i}} &= \mathrm{vsuc}\,(v{\downarrow_i})
\end{aligned}
$$

This satisfies the weakening lemma.

Lemma (weak-var-den)**.** *Given* $v : \mathrm{Var}\,\Gamma\,A$ *and* $i : \Gamma \subseteq \Delta$, $\langle v{\downarrow_i}\rangle^{var} = \langle v\rangle^{var} \circ \langle i\rangle^{weak}$.

Proof. In Agda (incomplete[4]). □

The weakening of terms is then defined as follows:

$$
\begin{aligned}
\_{\downarrow}\_ &: \forall\{\Gamma\,\Delta\,A\} \to \mathrm{Tm}\,\Gamma\,A \to (i : \Gamma \subseteq \Delta) \to \mathrm{Tm}\,\Delta\,A{\downarrow_i} \\
(\mathrm{con}\,c){\downarrow_i} &= \mathrm{con}\,c \\
(\mathrm{var}\,v){\downarrow_i} &= \mathrm{var}\,(v{\downarrow_i}) \\
(\mathrm{abs}\,t){\downarrow_i} &= \mathrm{abs}\,(t{\downarrow_{\uparrow i}}) \\
(\mathrm{app}\,t\,u){\downarrow_i} &= \mathrm{app}\,(t{\downarrow_i})\,(u{\downarrow_i})
\end{aligned}
$$

This also satisfies the corresponding weakening lemma.

Lemma (weak-term-den)**.** *Given* $t : \mathrm{Tm}\,\Gamma\,A$ *and* $i : \Gamma \subseteq \Delta$, $\langle t{\downarrow_i}\rangle = \langle t\rangle \circ \langle i\rangle^{weak}$.

Proof. (No Agda proof.)

1. Case $t = \mathrm{con}\,c$. Remains to show $\langle(\mathrm{con}\,c){\downarrow_i}\rangle = \langle \mathrm{con}\,c\rangle \circ \langle i\rangle^{\mathrm{weak}}$.
   (a) $\langle t\rangle = (\lambda\_ \to \langle c\rangle^{\mathrm{con}})$
   (b) $\langle t{\downarrow_i}\rangle = (\lambda\_ \to \langle c\rangle^{\mathrm{con}})$
   (c) Thus $\langle t{\downarrow_i}\rangle = \langle t\rangle \circ \langle i\rangle^{\mathrm{weak}}$.
   From (a), (b).
2. Case $t = \mathrm{var}\,v$. Remains to show $\langle v{\downarrow_i}\rangle^{\mathrm{var}} = \langle v\rangle^{\mathrm{var}} \circ \langle i\rangle^{\mathrm{weak}}$.
   (a) $\langle v{\downarrow_i}\rangle^{\mathrm{var}} = \langle v\rangle^{\mathrm{var}} \circ \langle i\rangle^{\mathrm{weak}}$
   From weak-var-den.
3. Case $t = \mathrm{abs}\,t'$. Remains to show $\langle(\mathrm{abs}\,t'){\downarrow_i}\rangle = \langle \mathrm{abs}\,t'\rangle \circ \langle i\rangle^{\mathrm{weak}}$.
   (a) $\langle t\rangle = (\lambda\gamma\,a \to \langle t'\rangle\,(\gamma, a))$
   (b) $\langle t{\downarrow_i}\rangle = (\lambda\delta\,a \to \langle t'{\downarrow_{\uparrow i}}\rangle\,(\delta, a))$
   (c) $\langle t'{\downarrow_{\uparrow i}}\rangle = \langle t'\rangle \circ \langle{\uparrow} i\rangle^{\mathrm{weak}}$
   By induction hypothesis.

[4] A prose proof of the remaining holes in the Agda proof can be found in Section 10 of the appendix.

(d) Thus $\langle t{\downarrow_i}\rangle = \langle t\rangle \circ \langle i\rangle^{\text{weak}}$.
From (a), (b), (c), unfolding $\langle{\uparrow i}\rangle^{\text{weak}} = \lambda(\delta, a) \to (\langle i\rangle^{\text{weak}}\, \delta, a)$.

4. Case $t = \text{app}\, t'\, u$. Remains to show $\langle(\text{app}\, t'\, u)\downarrow_i\rangle = \langle \text{app}\, t'\, u\rangle \circ \langle i\rangle^{\text{weak}}$.

(a) $\langle t\rangle = (\lambda\gamma \to \langle t'\rangle\, \gamma\, (\langle u\rangle\, \gamma))$

(b) $\langle t{\downarrow_i}\rangle = (\lambda\gamma \to \langle t'{\downarrow_i}\rangle\, \gamma\, (\langle u{\downarrow_i}\rangle\, \gamma))$

(c) $\langle t'{\downarrow_i}\rangle = \langle t'\rangle \circ \langle i\rangle^{\text{weak}}$
By induction hypothesis.

(d) $\langle u{\downarrow_i}\rangle = \langle u\rangle \circ \langle i\rangle^{\text{weak}}$
By induction hypothesis.

(e) Thus $\langle t{\downarrow_i}\rangle = \langle t\rangle \circ \langle i\rangle^{\text{weak}}$.
From (a), (b), (c), (d).

□

Chapter 7

LF-Definability

In this chapter we will finally state and prove our main result, the LF-definability theorem. All of the definitions and most of the proofs in this chapter are formalised in Agda.[1][2]

The structure of our LF-definability result will follow the ideas of Jung and Tiuryn's [17] work on STLC-definability; correspondingly, the structure of this chapter mirrors that of Chapter 5. Note that there is one key difference in our presentation of LF-definability compared to that of STLC-definability in Chapter 5: due to our particular formulation of LF we are not able to define a *general* notion of *Kripke semantics* of LF, and as a result we are not able to give the 'NbE' algorithm for LF in isolation. Instead we will work with *Kripke predicates* over a 'Kripke-style presentation' of the standard semantics of LF directly.

7.1 A Kripke-Style Presentation of LF Semantics

The Agda code corresponding to the definitions and lemmas in this section can be found in Kripke.agda.

We will now lay the groundwork for our LF-definability theorem by defining a *Kripkefied* version of the LF standard semantics. As mentioned previously, we are not able to define a general notion of *Kripke semantics* of LF, thus making our constructions seem a bit more ad-hoc than in the STLC case.

We consider each of the parts that make up our 'Kripkefied standard semantics of STLC' in turn and define a corresponding notion for LF. We will end up with constructions that have the same computational content, but with somewhat more complicated types.[3]

Worlds In the Kripkefied STLC semantics we chose the worlds to be contexts.

[1] A zip file containing the code is attached to this pdf *here*.

[2] Note that we mix proofs that are fully formalised in Agda's intensional theory with our extensional 'paper theory'. This is unproblematic if we consider our paper theory to be Agda + extensionality, which should be consistent (at least for the reasonably well-understood subset of Agda we actually use).

[3] This is not at all surprising, since in some sense LF is computationally equivalent to STLC (via the erasure of type dependencies [16]).

This carries over directly to LF, and we will think of contexts

$$\mathrm{Cxt} : \mathrm{Set}$$

as *worlds*, ordered by the order-preserving embeddings $\_ \subseteq \_ : \mathrm{Cxt} \to \mathrm{Cxt} \to \mathrm{Set}$.

Indexed Type semantics The type of the standard denotation function of LF types $[\_] : \{\Gamma : \mathrm{Cxt}\} \to \mathrm{Fam}\,\Gamma \to [\Gamma]^* \to \mathrm{Set}$ already has a certain 'Kripke flavour' to it, with the (implicit) index appearing before the type family. In order to make the correspondence with the simply-typed notion[4] more obvious, we flip the visual order of the arguments when defining the *indexed semantics* of LF types.

$$\begin{aligned}[\![\_2]\!]_{\_1} &: (\Gamma : \mathrm{Cxt}) \to \mathrm{Fam}\,\Gamma \to \mathrm{Set}\\ [\![A]\!]_\Gamma &= (\gamma : [\Gamma]^*) \to [A]\,\gamma\end{aligned}$$

Transport Just as in the simply-typed case we can *transport* along order-preserving embeddings.

$$\begin{aligned}\_ \nearrow \_ &: [\![A]\!]_\Gamma \to (i : \Gamma \subseteq \Delta) \to [\![A{\downarrow_i}]\!]_\Delta\\ a \nearrow i &= a \circ \langle i \rangle^{\mathrm{weak}}\end{aligned}$$

Compared to the simply-typed case[5] this bears the additional complication that we have to simultaneously 'transport' (i.e. weaken) the resulting type along the embedding.

Application Recall the type of the term constructor for *applications* in LF:

$$\frac{t : \mathrm{Tm}\,\Gamma\,(\mathrm{pi}\,A\,B) \quad u : \mathrm{Tm}\,\Gamma\,A}{\mathrm{app}\,t\,u : \mathrm{Tm}\,\Gamma\,(B/_x\langle u\rangle)}$$

Accordingly, *Kripke application* needs to substitute the function argument in the type of the result, with the difference that the argument is given semantically (rather than syntactically).[6]

$$\begin{aligned}\mathrm{apply} &: \forall\{\Gamma\,A\,B\} \to (f : [\![\mathrm{pi}\,A\,B]\!]_\Gamma) \to (a : [\![A]\!]_\Gamma) \to [\![B/_x a]\!]_\Gamma\\ \mathrm{apply}\,f\,a &= \lambda\gamma \to f\,\gamma\,(a\,\gamma)\end{aligned}$$

Environments The *Kripke environments* [7] $\eta : [\![\Gamma]\!]^*_\Delta$ and their denotations[8] $\langle\eta\rangle^*$ are defined mutually.

$$\begin{aligned}[\![\_]\!]^*_\_ &: \mathrm{Cxt} \to \mathrm{Cxt} \to \mathrm{Set}\\ [\![\varnothing]\!]^*_\Delta &= \top\\ [\![\Gamma, A]\!]^*_\Delta &= \sum(\eta : [\![\Gamma]\!]^*_\Delta)([\![A/\langle\eta\rangle^*]\!]_\Delta)\end{aligned}$$

[4] (STLC) $[\![\_]\!]^{\mathcal{S}} : \mathrm{Ty} \to \mathrm{Cxt} \to \mathrm{Set}$; $[\![A]\!]^{\mathcal{S}}_\Gamma = [\Gamma]^* \to [A]$.

[5] (STLC) $\_ \nearrow \_ : [\![A]\!]^{\mathcal{S}}_\Gamma \to \Gamma \subseteq \Delta \to [\![A]\!]^{\mathcal{S}}_\Delta$; $a \nearrow i = a \circ \langle i\rangle^{\mathrm{weak}}$.

[6] (STLC) $\mathrm{apply}^{\mathcal{S}} : [\![A \Rightarrow B]\!]^{\mathcal{S}}_\Gamma \to [\![A]\!]^{\mathcal{S}}_\Gamma \to [\![B]\!]^{\mathcal{S}}_\Gamma$; $\mathrm{apply}\,f\,x = \lambda\gamma \to f\,\gamma\,(x\,\gamma)$.

[7] (STLC) $[\![\_]\!]^* : \mathrm{Cxt} \to \mathrm{Cxt} \to \mathrm{Set}$; $[\![\varnothing]\!]^{*\mathcal{S}}_\Delta = \top$, $[\![\Gamma, A]\!]^{*\mathcal{S}}_\Delta = [\![\Gamma]\!]^{*\mathcal{S}}_\Delta \times [\![A]\!]^{\mathcal{S}}_\Delta$.

[8] (STLC) $\langle\_\rangle^{*\mathcal{S}} : \forall\{\Gamma\,\Delta\} \to [\![\Gamma]\!]^{*\mathcal{S}}_\Delta \to [\Delta]^* \to [\Gamma]^*$; $\langle f\rangle^{*\mathcal{S}}_{\{\Gamma=\emptyset\}} = f$, $\langle\eta, \alpha\rangle^{*\mathcal{S}}_{\{\Gamma=(\Gamma,A)\}} = \lambda\delta \to (\langle\eta\rangle^{*\mathcal{S}}\,\delta, \alpha\,\delta)$.

$$\begin{aligned} &\langle \_ \rangle^* : \forall\{\Gamma\,\Delta\} \to [\![\Gamma]\!]^*_\Delta \to [\Delta]^* \to [\Gamma]^* \\ \{\Gamma = \emptyset\} \quad &\langle \eta \rangle^*\,\delta = \mathrm{tt} \\ \{\Gamma = (\Gamma, A)\} \quad &\langle \eta, \alpha \rangle^*\,\delta = (\langle \eta \rangle^*\,\delta, \alpha\,\delta) \end{aligned}$$

In presenting STLC Kripke semantics we skipped over the technical detail of transporting environments. To transport a Kripke environment $\eta : [\![\Gamma]\!]^*_\Delta$ along an inclusion $i : \Delta \subseteq \Omega$ we transport its elements pointwise:

$$\begin{aligned} &\_ \nearrow \_ : \forall\{\Gamma\,\Delta\,\Omega\} \to [\![\Gamma]\!]^*_\Delta \to (i : \Delta \subseteq \Omega) \to [\![\Gamma]\!]^*_\Omega \\ \{\Gamma = \varnothing\} \quad &\eta \nearrow i = \mathrm{tt} \\ \{\Gamma = (\Gamma, A)\} \quad &(\eta, \alpha) \nearrow i = (\eta \nearrow i, \alpha \nearrow i) \end{aligned}$$

Term Semantics The *Kripkefied* semantics of LF terms[9] is given by:

$$\begin{aligned} &(\!|\_|\!) : \forall\{\Gamma\,A\,\Delta\} \to \mathrm{Tm}\,\Gamma\,A \to (\eta : [\![\Gamma]\!]^*_\Delta) \to [\![A/\langle \eta \rangle^*]\!]_\Delta \\ &(\!|t|\!)\,\eta = \langle t \rangle \circ \langle \eta \rangle^* \end{aligned}$$

7.1.1 Miscellaneous Facts

We will need to make use of the following fact, stating that the evaluation of Kripke environments is compatible with the action of transport.

Lemma (transp-env-den)**.** *Given $\eta : [\![\Gamma]\!]^*_\Delta$ and $i : \Delta \subseteq \Omega$ we have that $\langle \eta \nearrow i \rangle^* = \langle \eta \rangle^* \circ \langle i \rangle^{weak}$.*

Proof. In Agda. □

Furthermore, we are able to construct an *identity environment* $\mathrm{fresh}_\Gamma : [\![\Gamma]\!]^*_\Gamma$ whose denotation is the identity $\langle \mathrm{fresh}_\Gamma \rangle^* = \mathrm{id} : [\Gamma]^* \to [\Gamma]^*$. This connects our Kripkefied denotations with the standard semantics of LF, since $(\!|t|\!)\,\mathrm{fresh}_\Gamma = \langle t \rangle$.

$$\begin{aligned} \mathrm{fresh}_\_ &: (\Gamma : \mathrm{Cxt}) \to [\![\Gamma]\!]^*_\Gamma \\ \mathrm{fresh}_\varnothing &= \mathrm{tt} \\ \mathrm{fresh}_{(\Gamma,A)} &= (\mathrm{fresh}_\Gamma \nearrow (\downarrow \mathrm{refl}), \pi_2) \end{aligned}$$

Lemma (fresh-id)**.** *For every Γ, $\langle \mathrm{fresh}_\Gamma \rangle^* = id$.*

Proof. In Agda. □

7.1.2 Kripke Predicates

The Agda code corresponding to the definitions and lemmas in this section can be found in KripkePredicate.agda.

Like in the case of STLC-definability, our LF-definability theorem gives a characterisation of definability in terms of (LF) Kripke predicates.

A *Kripke predicate* of LF is an indexed predicate

$$\mathcal{P}[\![\_{}_2]\!]_{\_1} : (\Gamma : \mathrm{Cxt}) \to (A : \mathrm{Fam}\,\Gamma) \to [\![A]\!]_\Gamma \to \mathrm{Set}$$

that

[9] (STLC) $(\!|\_|\!)^{\mathcal{S}} : \forall\{\Gamma\,A\,\Delta\} \to \mathrm{Tm}\,\Gamma\,A \to (\eta : [\![\Gamma]\!]^*_\Delta) \to [\![A]\!]_\Delta$; $(\!|t|\!)^{\mathcal{S}}\,\eta = \langle t \rangle \circ \langle \eta \rangle^*$.

- is monotone at base types,

$$a \in \mathcal{P}[\![\mathrm{dat}\, C\, \sigma]\!]_\Gamma \to (\Delta : \mathrm{Cxt}) \to (i : \Gamma \subseteq \Delta) \to (a \nearrow i) \in \mathcal{P}[\![(\mathrm{dat}\, C\, \sigma)\downarrow_i]\!]_\Delta,$$

- contains all constants,

$$(c : \mathrm{const}) \to (\lambda\_ \to \langle c \rangle^{\mathrm{con}}) \in \mathcal{P}[\![\tau(c)]\!]_\varnothing,$$

- and has the 'Kripke function space' property[10]

$$f \in \mathcal{P}[\![\mathrm{pi}\, A\, B]\!]_\Gamma \iff \forall \Delta \to (i : \Gamma \subseteq \Delta) \to (a \in \mathcal{P}[\![A\downarrow_i]\!]_\Delta) \to \mathrm{apply}\,(f \nearrow i)\, a \in \mathcal{P}[\![(B\downarrow_{\uparrow i})\,/_x a]\!]_\Delta.$$

Such a Kripke predicate can be shown to be monotone at any type.

Lemma (mon)**.** *Given a Kripke predicate $\mathcal{P}[\![\_]\!]$, if $a \in \mathcal{P}[\![A]\!]_\Gamma$ and $i : \Gamma \subseteq \Delta$ then $a \nearrow i \in \mathcal{P}[\![A\downarrow_i]\!]_\Delta$.*

Proof. In Agda (incomplete[11]). □

We define the pointwise extension $\mathcal{P}[\![\Gamma]\!]^*_\Delta \subseteq [\![\Gamma]\!]_\Delta$ of a Kripke predicate to environments, which satisfies monotonicity pointwise.

$$\begin{aligned} \mathcal{P}[\![\Gamma]\!]^*_\Delta &: [\![\Gamma]\!]^*_\Delta \to \mathrm{Set} \\ \eta \in \mathcal{P}[\![\varnothing]\!]^*_\Delta &= \top \\ (\eta, \alpha) \in \mathcal{P}[\![\Gamma, A]\!]^*_\Delta &= \eta \in \mathcal{P}[\![\Gamma]\!]^*_\Delta \wedge \alpha \in \mathcal{P}[\![A/\langle \eta \rangle^*]\!]_\Delta \end{aligned}$$

Lemma (mon-env)**.** *Given a Kripke predicate $\mathcal{P}[\![\_]\!]$, if $\eta \in \mathcal{P}[\![\Gamma]\!]^*_\Delta$ and $i : \Delta \subseteq \Omega$ then $\eta \nearrow i \in \mathcal{P}[\![\Gamma]\!]^*_\Omega$.*

Proof. In Agda. □

We are now able to prove the 'fundamental lemma' of Kripke-logical predicates for LF.

Lemma (fund)**.** *Given a Kripke predicate $\mathcal{P}[\![\_]\!]$, $t : \mathrm{Tm}\,\Gamma\, A$ and $\eta \in \mathcal{P}[\![\Gamma]\!]^*_\Delta$ we have $(\!|t|\!)\,\eta \in \mathcal{P}[\![A/\langle \eta \rangle^*]\!]_\Delta$.*

Proof. We proceed by induction on t.

1. Case $t = \mathrm{var\ vzero}$, where $t : \mathrm{Tm}\,(\Gamma, A)\, A\downarrow_{\mathrm{refl}}$. Write $A' = A\downarrow_{\mathrm{refl}}/\langle \eta \rangle^* : \mathrm{Fam}\,\Delta$.
   (a) Have $\eta = (\eta', \alpha)$ for some $\eta' : [\![\Gamma]\!]^*_\Delta$ and $\alpha : [\![A/\langle \eta' \rangle^*]\!]_\Delta$.
   By definition of $[\![\_]\!]^*\_$, with $\eta : [\![\Gamma, A]\!]^*_\Delta$.
   (b) Have $(\!|t|\!)\,\eta = \alpha$
      i. $(\!|t|\!)\,\eta = \lambda\delta \to \langle \mathrm{var\ vzero} \rangle\,(\langle \eta' \rangle^*\,\delta, \alpha\,\delta)$
      By definition of $(\!|\_|\!)$ and unfolding $\langle \eta', \alpha \rangle^* = \lambda\delta \to (\langle \eta' \rangle^*\,\delta, \alpha\,\delta)$.

[10] The Agda code instead takes the semantics of the predicate at base types as a parameter, and *defines* the semantics at function types *to be* the 'Kripke function space'; the resulting notion is equivalent to the one presented here, while being slightly more convenient to work with in Agda. Note that in order for Agda to accept the code we had to turn of the termination checker. An informal termination argument can be found in the appendix.

[11] A prose proof of the remaining hole in the Agda proof can be found in Section 10 of the appendix.

ii. $(\lambda\delta \to \langle \mathrm{var\,vzero}\rangle\,(\langle\eta'\rangle^*\,\delta, \alpha\,\delta)) = (\lambda\delta \to \alpha\,\delta)$
By definition of $\langle\_\rangle$.

(c) Have $\alpha \in \mathcal{P}[\![A']\!]_\Delta$.
Since $\eta \in \mathcal{P}[\![\Gamma, A]\!]^*_\Delta$ by assumption and $A' = A/\langle\eta'\rangle^*$.

(d) Thus $(\!|t|\!)\,\eta \in \mathcal{P}[\![A']\!]_\Delta$
By (b), (c).

2. Case $t = \mathrm{var}\,(\mathrm{vsuc}\,v)$, where $t : \mathrm{Tm}\,(\Gamma, B)\,A{\downarrow_{\mathrm{refl}}}$ and $v : \mathrm{Var}\,\Gamma\,A$.

(a) Have $\eta = (\eta', \alpha)$ for some $\eta' : [\![\Gamma]\!]^*_\Delta$ and $\alpha : [\![B/\langle\eta'\rangle^*]\!]_\Delta$.
By definition of $[\![\_]\!]^*_\_$, with $\eta : [\![\Gamma, B]\!]^*_\Delta$.

(b) Have $\eta' \in \mathcal{P}[\![\Gamma]\!]^*_\Delta$
By assumption, since $\eta \in \mathcal{P}[\![\Gamma, B]\!]^*_\Delta$.

(c) $(\!|t|\!)\,\eta = (\!|\mathrm{var}\,v|\!)\,\eta'$

i. $(\!|t|\!)\,\eta = \lambda\delta \to \langle \mathrm{var}\,(\mathrm{vsuc}\,v)\rangle\,(\langle\eta'\rangle^*\,\delta, \alpha\,\delta)$
By definition of $(\!|\_|\!)$ and unfolding $\langle\eta', \alpha\rangle^* = \lambda\delta \to (\langle\eta'\rangle^*\,\delta, \alpha\,\delta)$.

ii. $\cdots = \lambda\delta \to \langle \mathrm{var}\,v\rangle\,(\langle\eta'\rangle^*\,\delta)$

iii. $\cdots = (\!|\mathrm{var}\,v|\!)\,\eta'$

(d) $(\!|\mathrm{var}\,v|\!)\,\eta' \in \mathcal{P}[\![A/\langle\eta'\rangle^*]\!]_\Delta$
By induction hypothesis.

(e) $A{\downarrow_{\mathrm{refl}}}/\langle\eta\rangle^* = A/\langle\eta'\rangle^*$

i. $A{\downarrow_{\mathrm{refl}}}/\langle\eta\rangle^* = A/\langle{\downarrow}\mathrm{refl}\rangle^*/\langle\eta\rangle^*$

ii. $\cdots = A/(\lambda(\gamma, a) \to \gamma) \circ \langle(\eta', \alpha)\rangle^*$

iii. $\cdots = A/(\lambda(\gamma, a) \to \gamma) \circ (\lambda\delta \to (\langle\eta'\rangle^*\,\delta, \alpha\,\delta))$

iv. $\cdots = A/\langle\eta'\rangle^*$

(f) Thus $(\!|t|\!)\,\eta \in \mathcal{P}[\![A{\downarrow_{\mathrm{refl}}}/\langle\eta\rangle^*]\!]_\Delta$.
By (c), (d), (e).

3. Case $t = \mathrm{con}\,c$, where $t : \mathrm{Tm}\,\Gamma\,\tau(c){\downarrow_i}$, $i : \varnothing \subseteq \Gamma$ and $i = \epsilon_\Gamma$.

(a) Let $i' = \epsilon_\Delta : \varnothing \subseteq \Delta$.

(b) $(\!|t|\!)\,\eta = (\lambda(x : \top) \to \langle c\rangle^{\mathrm{con}}) \nearrow i'$

i. $(\!|t|\!)\,\eta = \langle \mathrm{con}\,c\rangle \circ \langle\eta\rangle^*$

ii. $\langle \mathrm{con}\,c\rangle \circ \langle\eta\rangle^* = \lambda(\delta : [\Delta]^*) \to \langle c\rangle^{\mathrm{con}}$
By definition of $\langle\_\rangle$.

iii. $(\lambda(\delta : [\Delta]^*) \to \langle c\rangle^{\mathrm{con}}) = (\lambda(t : \top) \to \langle c\rangle^{\mathrm{con}}) \circ \langle i'\rangle^{\mathrm{weak}}$

(c) $(\lambda(x : \top) \to \langle c\rangle^{\mathrm{con}}) \in \mathcal{P}[\![\tau(c)]\!]_\varnothing$.
Because $\mathcal{P}[\![\_]\!]$ is a Kripke predicate.

(d) Thus $(\!|t|\!)\,\eta \in \mathcal{P}[\![\tau(c){\downarrow_{i'}}]\!]_\Delta$
From (b), (c) and monotonicity of $\mathcal{P}$ (mon).

4. Case $t = \mathrm{abs}\,t'$, where $t' : \mathrm{Tm}\,(\Gamma, A)\,B$.

(a) It suffices to show $\mathrm{apply}\,((\!|t|\!)\,\eta \nearrow i)\,a \in \mathcal{P}[\![((B/\mathrm{lift}\,\langle\eta\rangle^*){\downarrow_{\uparrow i}})/_x a]\!]_\Omega$, given an $a : [\![(A/\langle\eta\rangle^*){\downarrow_i}]\!]$ in a future context Ω with $i : \Gamma \subseteq \Omega$ such that $a \in \mathcal{P}[\![(A/\langle\eta\rangle^*){\downarrow_i}]\!]_\Omega$.
By the 'Kripke function-space' property of $\mathcal{P}$.

(b) $\mathrm{apply}\,(\llparenthesis t \rrparenthesis\,\eta \nearrow i)\,a = \llparenthesis t' \rrparenthesis\,(\eta \nearrow i, a)$

i. $\mathrm{apply}\,(\llparenthesis t \rrparenthesis\,\eta \nearrow i)\,a = \mathrm{apply}\,(\langle t\rangle \circ \langle \eta\rangle^* \circ \langle i\rangle^{\mathrm{weak}})\,a$

ii. $\cdots = \mathrm{apply}\,(\langle t\rangle \circ \langle \eta \nearrow i\rangle^*)\,a$
By transp-env-den.

iii. $\cdots = \lambda\omega \to \langle t\rangle\,(\langle \eta \nearrow i\rangle^*\,\omega)\,(a\,\omega)$

iv. $\cdots = \lambda\omega \to \langle \mathrm{abs}\,t'\rangle\,(\langle \eta \nearrow i\rangle^*\,\omega)\,(a\,\omega)$

v. $\cdots = \lambda\omega \to \langle t'\rangle\,(\langle \eta \nearrow i\rangle^*\,\omega, a\,\omega)$

vi. $\cdots = \llparenthesis t' \rrparenthesis\,(\eta \nearrow i, a)$

(c) $\eta \nearrow i \in \mathcal{P}[\![\Gamma]\!]^*_\Omega$
By monotonicity of $\mathcal{P}$.

(d) $(\eta \nearrow i, a) \in \mathcal{P}[\![\Gamma, A]\!]^*_\Omega$
By (a), (c).

(e) $\llparenthesis t' \rrparenthesis\,(\eta \nearrow i, a) \in \mathcal{P}[\![B/\langle \eta \nearrow i, a\rangle^*]\!]_\Omega$
By induction hypothesis.

(f) $((B/\mathrm{lift}\,\langle\eta\rangle^*)\downarrow_{\uparrow i})/_x a = B/\langle \eta \nearrow i, a\rangle^*$

i. $((B/\mathrm{lift}\,\langle\eta\rangle^*)\downarrow_{\uparrow i})/_x a = ((B/\lambda(\delta, x) \to (\langle\eta\rangle^*\,\delta, x))\downarrow_{\uparrow i})/_x a$

ii. $\cdots = ((B/\lambda(\delta, x) \to (\langle\eta\rangle^*\,\delta, x))/\lambda(\gamma, x) \to (\langle i\rangle^{\mathrm{weak}}\,\gamma, x))/_x a$

iii. $\cdots = (B/\lambda(\gamma, x) \to (\langle\eta\rangle^*\,(\langle i\rangle^{\mathrm{weak}}\,\gamma), x))/_x a.$
By sub-comp.

iv. $\cdots = (B/\lambda(\gamma, x) \to (\langle\eta\rangle^*\,(\langle i\rangle^{\mathrm{weak}}\,\gamma), x))/\lambda\gamma \to (\gamma, a\,\gamma)$

v. $\cdots = B/\lambda\gamma \to (\langle\eta\rangle^*\,(\langle i\rangle^{\mathrm{weak}}\,\gamma), a\,\gamma).$
By sub-comp.

vi. $\cdots = B/\lambda\gamma \to (\langle \eta \nearrow i\rangle^*\,\gamma, a\,\gamma).$
By transp-env-den.

vii. $\cdots = B/\langle \eta \nearrow i, a\rangle^*$

(g) Thus $\llparenthesis t \rrparenthesis\,\eta \in \mathcal{P}[\![(\mathrm{pi}\,A\,B)/\langle\eta\rangle^*]\!]_\Omega$.
By (a), (b), (e), (f).

5. Case $t = \mathrm{app}\,t'\,u'$, where $t' : \mathrm{Tm}\,\Gamma\,(\mathrm{pi}\,A\,B)$ and $u' : \mathrm{Tm}\,\Gamma\,A$.

(a) $\llparenthesis \mathrm{app}\,t'\,u' \rrparenthesis\,\eta = \mathrm{apply}\,(\llparenthesis t' \rrparenthesis\,\eta)\,(\llparenthesis u' \rrparenthesis\,\eta)$

i. $\mathrm{apply}\,(\llparenthesis t' \rrparenthesis\,\eta)\,(\llparenthesis u' \rrparenthesis\,\eta) = \lambda\delta \to \llparenthesis t' \rrparenthesis\,\eta\,\delta\,(\llparenthesis u' \rrparenthesis\,\eta\,\delta)$

ii. $\cdots = \lambda\delta \to \langle t'\rangle\,(\langle\eta\rangle^*\,\delta)\,(\langle u'\rangle\,(\langle\eta\rangle^*\,\delta))$

iii. $\cdots = (\lambda\gamma \to \langle t'\rangle\,\gamma\,(\langle u'\rangle\,\gamma)) \circ \langle\eta\rangle^*$

iv. $\cdots = \langle \mathrm{app}\,t'\,u'\rangle \circ \langle\eta\rangle^*$

v. $\cdots = \llparenthesis \mathrm{app}\,t'\,u' \rrparenthesis\,\eta$

(b) $\llparenthesis t' \rrparenthesis\,\eta \in \mathcal{P}[\![\mathrm{pi}\,(A/\langle\eta\rangle^*)\,(B/\mathrm{lift}\,\langle\eta\rangle^*)]\!]_\Gamma$, $\llparenthesis u' \rrparenthesis\,\eta \in \mathcal{P}[\![A/\langle\eta\rangle^*]\!]_\Gamma$
By induction hypothesis.

(c) $\mathrm{apply}\,(\llparenthesis t' \rrparenthesis\,\eta \nearrow \mathrm{refl})\,(\llparenthesis u' \rrparenthesis\,\eta) \in \mathcal{P}[\![((B/\mathrm{lift}\,\langle\eta\rangle^*)\downarrow_{\uparrow\mathrm{refl}})/_x(\llparenthesis u' \rrparenthesis\,\eta)]\!]_\Gamma$
By (b) and the 'Kripke function-space' property of $\mathcal{P}$.

(d) $((B/\mathrm{lift}\,\langle\eta\rangle^*)\downarrow_{\uparrow\mathrm{refl}})/_x(\llparenthesis u' \rrparenthesis\,\eta) = (B/_x\langle u'\rangle)/\langle\eta\rangle^*$

i. $(B/\mathrm{lift}\,\langle\eta\rangle^*)\downarrow_{\uparrow\mathrm{refl}} = (B/\mathrm{lift}\,\langle\eta\rangle^*)/\langle\uparrow \mathrm{refl}\rangle^{\mathrm{weak}}$
By weak-den.

ii. $\cdots = (B/\mathrm{lift}\,\langle\eta\rangle^*)/\mathrm{id}$

iii. $\cdots = B/\mathrm{lift}\,\langle\eta\rangle^*$
By sub-id.
iv. $(B/\mathrm{lift}\,\langle\eta\rangle^*)/_x(\Vert u'\Vert\,\eta)$
$= (B/(\lambda(\delta, a) \to (\langle\eta\rangle^*\,\delta, a)))/(\lambda\delta \to \delta, \langle u'\rangle\,(\langle\eta\rangle^*\,\delta))$
By unfolding lift, $\_/_x\_$ and $\Vert\_\Vert$.
v. $\cdots = B/(\lambda\delta \to (\langle\eta\rangle^*\,\delta, \langle u'\rangle\,(\langle\eta\rangle^*\,\delta)))$
By sub-comp.
vi. $\cdots = B/((\lambda\gamma \to (\gamma, \langle u'\rangle\,\gamma)) \circ \langle\eta\rangle^*)$
vii. $\cdots = (B/_x\langle u'\rangle)/\langle\eta\rangle^*$
By sub-comp, folding $\_/_x\_$.

(e) $\mathrm{apply}\,(\Vert t'\Vert\,\eta \nearrow \mathrm{refl})\,(\Vert u'\Vert\,\eta) = \mathrm{apply}\,(\Vert t'\Vert\,\eta)\,(\Vert u'\Vert\,\eta)$
Since $\_ \nearrow \mathrm{refl} = \_ \circ \langle\mathrm{refl}\rangle^{\mathrm{weak}} = \_ \circ \mathrm{id}$.

(f) Thus $\Vert t\Vert\,\eta \in \mathcal{P}[\![(B/_x\langle u'\rangle)/\langle\eta\rangle^*]\!]_\Gamma$
From (a), (c), (d) and (e).

□

The following two special cases of the fundamental lemma allow us to relate Kripke predicates to the standard semantics of LF, which we are ultimately interested in.

Lemma (fund-closed)**.** *Given a closed term* $t : \mathrm{Tm}\,\varnothing\,A$ *and Kripke predicate* $\mathcal{P}[\![\_]\!]$ *we have* $\langle t\rangle \in \mathcal{P}[\![A]\!]_\varnothing$.

Proof. (Proof also in Agda.)
The trivial environment $\mathrm{tt} : [\![\varnothing]\!]^*_\varnothing$ is by definition contained in any predicate, i.e. $\mathrm{tt} \in \mathcal{P}[\![\varnothing]\!]^*_\varnothing$. The fundamental theorem therefore gives us that $\Vert t\Vert\,\mathrm{tt} \in \mathcal{P}[\![A/\langle\mathrm{tt}\rangle^*]\!]_\varnothing$, which, since $\langle\mathrm{tt}\rangle^* : [\varnothing]^* \to [\varnothing]^*$ is the identity function, and since $\Vert t\Vert\,\mathrm{tt} = \langle t\rangle \circ \langle\mathrm{tt}\rangle^*$ by definition, gives us that $\langle t\rangle \in \mathcal{P}[\![A]\!]_\varnothing$. □

Lemma (fund-id-env)**.** *Given a term* $t : \mathrm{Tm}\,\Gamma\,A$ *and Kripke predicate* $\mathcal{P}[\![\_]\!]$*, if* $\mathcal{P}$ *contains the identity environment* $\mathrm{fresh}_\Gamma \in \mathcal{P}[\![\Gamma]\!]^*_\Gamma$ *then* $\langle t\rangle \in \mathcal{P}[\![A]\!]_\Gamma$.

Proof. (Proof also in Agda.)
From the fundamental lemma we have $\Vert t\Vert\,\mathrm{fresh}_\Gamma \in \mathcal{P}[\![A/\langle\mathrm{fresh}_\Gamma\rangle^*]\!]_\Gamma$. Since the identity environment $\mathrm{fresh}_\Gamma : [\![\Gamma]\!]^*_\Gamma$ evaluates to the identity function, $\langle\mathrm{fresh}_\Gamma\rangle^* = \mathrm{id} : [\Gamma]^* \to [\Gamma]^*$, and since $\Vert t\Vert\,\mathrm{fresh}_\Gamma = \langle t\rangle \circ \langle\mathrm{fresh}_\Gamma\rangle^*$ by definition, this gives us $\langle t\rangle \in \mathcal{P}[\![A]\!]_\Gamma$. □

7.2 Definability as a Kripke Predicate

The Agda code corresponding to the definitions and lemmas in this section can be found in NbE.agda.

In this section we construct a particular Kripke predicate that captures exactly the LF-definable objects, i.e. we will show that $x \in \mathcal{P}[\![A]\!]_\Gamma$ holds if and only if x is LF-definable. Note that this already gives us a possible characterisation of LF-definability, but due to the inherent *syntactic* nature of the 'definability predicate' (as we will see in the following) such an answer would hardly be satisfying from a semantic perspective. To construct $\mathcal{P}$ we follow the same recipe as outlined in Section 5.2.

7.2.1 Normal Forms

Note that the question of definability is superficially not at all concerned with normal forms—it only asks for the existance of some (not necessarily normal) defining term. However, since we are going to use NbE at the heart of the definability proof we will need normal forms anyway. This means that we obtain a sound-by-construction normalisation algorithm as a neat by-product of our efforts.

We define subsets of terms $\mathrm{Ne}, \mathrm{Nf} : (\Gamma : \mathrm{Cxt}) \to (A : \mathrm{Fam}\,\Gamma) \to \mathrm{Tm}\,\Gamma\,A \to \mathrm{Set}$ that correspond to the neutral and normal forms. Remember that we use subset syntax informally, e.g. writing $a \in P$ to stand for $P\,a$.

$$\frac{c : \mathrm{const} \quad \Gamma : \mathrm{Cxt}}{\mathrm{con}\,c \in \mathrm{Ne}\,\Gamma\,\tau(c)\downarrow_{\epsilon_\Gamma}} \qquad \frac{v : \mathrm{Var}\,\Gamma\,A}{\mathrm{var}\,v \in \mathrm{Ne}\,\Gamma\,A} \qquad \frac{t \in \mathrm{Ne}\,\Gamma\,(\mathrm{pi}\,A\,B) \quad u \in \mathrm{Nf}\,\Gamma\,A}{\mathrm{app}\,t\,u \in \mathrm{Ne}\,\Gamma\,(B/_x\langle u\rangle)}$$

$$\frac{t \in \mathrm{Ne}\,\Gamma\,(\mathrm{dat}\,C\,\sigma)}{t \in \mathrm{Nf}\,\Gamma\,(\mathrm{dat}\,C\,\sigma)} \qquad \frac{t \in \mathrm{Nf}\,(\Gamma, A)\,B}{\mathrm{abs}\,t \in \mathrm{Nf}\,\Gamma\,(\mathrm{pi}\,A\,B)}$$

We also prove that the neutral terms are closed under weakening.

Lemma (ne-weak)**.** *Given $t \in \mathrm{Ne}\,\Gamma\,A$ and $i : \Gamma \subseteq \Delta$, then $t\downarrow_i \in \mathrm{Ne}\,\Delta\,A\downarrow_i$.*

Proof. In Agda. □

7.2.2 The NbE Predicate

We define the NbE predicate (or 'definability predicate') as a straight-forward translation of the STLC definition:

$$x \in \mathcal{P}[\![\mathrm{dat}\,C\,\sigma]\!]_\Gamma = \Sigma(t \in \mathrm{Ne}\,\Gamma\,(\mathrm{dat}\,C\,\sigma))(x = \langle t\rangle)$$
$$f \in \mathcal{P}[\![\mathrm{pi}\,A\,B]\!]_\Gamma = \forall(\Delta : \mathrm{Cxt})(i : \Gamma \subseteq \Delta)(x \in \mathcal{P}[\![A\downarrow_i]\!]_\Delta) \to \mathrm{apply}\,(f \nearrow i)\,x \in \mathcal{P}[\![(B\downarrow_{\uparrow i})/_x x]\!]_\Delta$$

Note that this is indeed a Kripke predicate (ne-weak proves monotonicity, while the definition at pi-types is exactly the Kripke function space property).

We now prove reflection and reification by mutual induction. Note that we implicitly make use of the fact that weakening and substitution do not affect the size (in number of constructors) of types.[12] This fact is particular to LF and STLC, and does *not* readily extend to more advanced calculi (like $\lambda\Pi\omega$).

Lemma 1 (reflect)**.** *Whenever $t \in \mathrm{Ne}\,\Gamma\,A$ we have $\langle t\rangle \in \mathcal{P}[\![A]\!]_\Gamma$.*

Proof. By induction on the size (in number of Fam constructors) of the type A. We are given $t \in \mathrm{Ne}\,\Gamma\,A$ and need to show$\langle t\rangle \in \mathcal{P}[\![A]\!]_\Gamma$.

1. Case $t : \mathrm{Tm}\,\Gamma\,(\mathrm{dat}\,C\,\sigma)$.
   (a) $\langle t\rangle \in \mathcal{P}[\![\mathrm{dat}\,C\,\sigma]\!]_\Gamma$ iff $\Sigma(t' \in \mathrm{Ne}\,\Gamma\,(\mathrm{dat}\,C\,\sigma))(\langle t'\rangle = \langle t\rangle)$
   (b) Thus $\langle t\rangle \in \mathcal{P}[\![\mathrm{dat}\,C\,\sigma]\!]_\Gamma$ is witnessed by $t \in \mathrm{Ne}\,\Gamma\,A$.
2. Case $t : \mathrm{Tm}\,\Gamma\,(\mathrm{pi}\,A\,B)$.

[12] See Section 9 of the appendix for a short elaboration of the termination argument.

(a) It suffices to show for every Δ, $i : \Gamma \subseteq \Delta$ and $a \in \mathcal{P}[\![A{\downarrow_i}]\!]_\Delta$ that $\mathrm{apply}\,(\langle t\rangle \nearrow i)\,a \in \mathcal{P}[\![(B{\downarrow_{\uparrow i}})/_x a]\!]_\Delta$.

(b) There is a $u \in \mathrm{Nf}\,\Delta\,A{\downarrow_i}$ such that $a = \langle u\rangle$.
By reification at $A{\downarrow_i}$. (Note that $A{\downarrow_i}$ is structurally smaller than $\mathrm{pi}\,A\,B$.)

(c) $\mathrm{apply}\,(\langle t\rangle \nearrow i)\,a = \langle \mathrm{app}\,(t{\downarrow_i})\,u\rangle$

i. $\mathrm{apply}\,(\langle t\rangle \nearrow i)\,a = \lambda\delta \to (\langle t\rangle \circ \langle i\rangle^{\mathrm{weak}})\,\delta\,(a\,\delta)$

ii. $\big(\lambda\delta \to (\langle t\rangle \circ \langle i\rangle^{\mathrm{weak}})\,\delta\,(a\,\delta)\big) = \lambda\delta \to (\langle t{\downarrow_i}\rangle)\,\delta\,(a\,\delta)$.
By weak-term-den.

iii. $(\lambda\delta \to \langle t{\downarrow_i}\rangle\,\delta\,(a\,\delta)) = \langle \mathrm{app}\,(t{\downarrow_i})\,u\rangle$

(d) $t{\downarrow_i} \in \mathrm{Ne}\,\Delta\,(\mathrm{pi}\,(A{\downarrow_i})\,(B{\downarrow_{\uparrow i}}))$
By assumption, ne-weak.

(e) $\mathrm{app}\,(t{\downarrow_i})\,u \in \mathrm{Ne}\,\Delta\,((B{\downarrow_{\uparrow i}})/_x\langle u\rangle)$
By (b), (d).

(f) $\langle \mathrm{app}\,(t{\downarrow_i})\,u\rangle \in \mathcal{P}[\![(B{\downarrow_{\uparrow i}})/_x\langle u\rangle]\!]_\Delta$
By reflecting at $(B{\downarrow_{\uparrow i}})/_x a$ (has the same size as B and is therefore structurally smaller than $\mathrm{pi}\,A\,B$).

(g) Thus $\mathrm{apply}\,(\langle t\rangle \nearrow i)\,a \in \mathcal{P}[\![(B{\downarrow_{\uparrow i}})/_x a]\!]_\Delta$.
By (b), (c), (f).

□

Lemma 2 (reify)**.** *For every $x \in \mathcal{P}[\![A]\!]_\Gamma$ there exists a $t \in \mathrm{Nf}\,\Gamma\,A$ such that $x = \langle t\rangle$.*

Proof. By induction on the size (in the number of Fam constructors) of the type A. We are given $x \in \mathcal{P}[\![A]\!]_\Gamma$ and need to construct $t : \mathrm{Nf}\,\Gamma\,A$ such that $x = \langle t\rangle$.

1. Case $x : [\![\mathrm{dat}\,C\,\sigma]\!]_\Gamma$.

   (a) There exists a $t \in \mathrm{Ne}\,\Gamma\,(\mathrm{dat}\,C\,\sigma)$ such that $x = \langle t\rangle$.
   By definition of $\mathcal{P}$ at type family constants.

   (b) Thus also $t \in \mathrm{Nf}\,\Gamma\,(\mathrm{dat}\,C\,\sigma)$.

2. Case $x : [\![\mathrm{pi}\,A\,B]\!]_\Gamma$.

   (a) Let $i = {\downarrow}\,\mathrm{refl} : \Gamma \subseteq (\Gamma, A)$.

   (b) Let $u = \mathrm{var}\,\mathrm{vzero} : \mathrm{Tm}\,(\Gamma, A)\,A{\downarrow_i}$.

   (c) $\langle u\rangle \in \mathcal{P}[\![A{\downarrow_i}]\!]_{(\Gamma,A)}$
   By reflection at $A{\downarrow_i}$.

   (d) $(B{\downarrow_{\uparrow i}})/_x\langle u\rangle = B$

   i. $(B{\downarrow_{\uparrow i}})/_x\langle u\rangle = (B{\downarrow_{\uparrow i}})/(\lambda(\gamma, a) \to (\gamma, a), \langle \mathrm{var}\,\mathrm{vzero}\rangle\,(\gamma, a))$
   By unfolding $\_/_x\_$ and u.

   ii. $\cdots = (B{\downarrow_{\uparrow i}})/(\lambda(\gamma, a) \to (\gamma, a), a)$

   iii. $B{\downarrow_{\uparrow i}} = B/\langle\uparrow{\downarrow}\,\mathrm{refl}\rangle^{\mathrm{weak}}$
   By weak-sub and unfolding of i.

   iv. $\cdots = (B/(\lambda((\gamma, a_1), a_2) \to (\gamma, a_2)))$

v. $(B{\downarrow_{\uparrow i}})/_x\langle u\rangle = B/(\lambda(\gamma, a) \to (\gamma, a))$
By (i)–(ii), (iii–iv) and sub-comp.

vi. $\cdots = B$
By sub-id.

(e) $\mathrm{apply}\,(x \nearrow i)\,\langle u\rangle \in \mathcal{P}[\![B]\!]_{(\Gamma,A)}$
By the definition of $\mathcal{P}$ at pi types and (d).

(f) Have some $t \in \mathrm{Nf}\,(\Gamma, A)\,B$ such that $\mathrm{apply}\,(x \nearrow i)\,\langle u\rangle = \langle t\rangle$
By (e) and reification at B.

(g) $x = \langle \mathrm{abs}\, t\rangle$

i. $\langle \mathrm{abs}\, t\rangle = \lambda(\gamma : [\Gamma]^*)(a : [A]\,\gamma) \to \langle t\rangle\,(\gamma, a)$

ii. $\cdots = \lambda(\gamma : [\Gamma]^*)(a : [A]\,\gamma) \to \mathrm{apply}\,(x \nearrow i)\,\langle u\rangle\,(\gamma, a)$

iii. $\cdots = \lambda(\gamma : [\Gamma]^*)(a : [A]\,\gamma) \to ((x \nearrow i)\,(\gamma, a))\,(\langle u\rangle\,(\gamma, a))$

iv. $\cdots = \lambda(\gamma : [\Gamma]^*)(a : [A]\,\gamma) \to x\,\gamma\,a$
using $i =\downarrow \mathrm{refl}$ and $u = \mathrm{var}\,\mathrm{vzero}$.

v. $\cdots = x$

□

Using reflection we can show that our predicate contains the identity environment $\mathrm{fresh}_\Gamma : [\![\Gamma]\!]^*_\Gamma$.

Lemma (fresh-nbe)**.** *For each* $\Gamma : \mathrm{Cxt}$, $\mathrm{fresh}_\Gamma \in \mathcal{P}[\![\Gamma]\!]^*_\Gamma$.

Proof. In Agda. □

We can now give a correct-by-construction NbE algorithm for LF. Note that we do not actually need the full strength of this result, in particular the fact that the terms are in normal form is of no relevance for the definability theorem; this normalisation proof is merely an interesting by-product of the project.

Theorem (norm)**.** *Given a term* $t : \mathrm{Tm}\,\Gamma\,A$ *there exists a normal form* $t' \in \mathrm{Nf}\,\Gamma\,A$ *with the same denotation,* $\langle t\rangle = \langle t'\rangle$.

Proof. (Proof in Agda.) □

7.3 The LF-Definability Theorem

Having set up all of the pieces of the puzzle, we are now able to state and prove our main theorem. As in the STLC-definability theorem in Section 5.3 one direction of the proof is a corollary of the fundamental theorem of Kripke predicates, while the other direction follows as a by-product of the normalisation-by-evaluation algorithm.

Theorem 3. *At* $A : \mathrm{Fam}\,\varnothing$, $x : [A]\,\mathrm{tt}$ *is definable in LF iff it satisfies all Kripke predicates. In other words, there exists a* $t : \mathrm{Tm}\,\varnothing\,A$ *such that* $(\lambda\_ \to x) = \langle t\rangle$, *iff for every Kripke logical predicate* $\mathcal{P}$ *we have* $(\lambda\_ \to x) \in \mathcal{P}[\![A]\!]_\varnothing$.

Proof. (Proof also in Agda, in Definability.agda.)

If $t : \mathrm{Tm}\,\varnothing\,A$ defines x the fact that x satisfies all LF Kripke predicates follows trivially from the fundamental lemma of Kripke predicates. We proved this in the lemma fund-closed.

In the other direction, if x satisfies all Kripke predicates it also satisfies our NbE predicate, i.e. $(\lambda\_ \to x) \in \mathcal{P}[\![A]\!]_{\varnothing}$. The reification lemma then gives us a term (in normal form) $t : \mathrm{Tm}\,\varnothing\,A$ with $(\lambda\_ \to x) = \langle t \rangle$. □

Chapter 8

Conclusion

In Section 7.3 we derived a characterisation of LF-definability (i.e. definability in the Edinburgh Logical Framework) in terms of Kripke predicates, adapting Jung and Tiuryn's work on STLC-definability. The fact that convertible types in our version of the LF calculus were only equal up to *propositional* (rather than *definitional*) equality created a lot of overhead in the Agda code. In discussion with Nils Anders Danielsson he suggested that this could have been avoided by following McBride's approach more closely [19]; he demonstrates this by deriving a definability result following our approach in one of his own Agda developments [13].

Along the way, in Section 7.2, we obtained a correct-by-construction normalisation procedure for LF. Most of the proofs are fully formalised in Agda, barring a few holes and termination arguments that are dealt with informally in the Appendix. Danielsson previously formalised NbE for (a subtly different formulation of) LF [14]; more recently, normalisation using NbE has been shown for variants of Martin-Löf type theory by Altenkirch and Kaposi [3], as well as Wieczorek and Biernacki [24].

Looking back, the 'semi-semantic' representation of LF we used in deriving our main result shares a certain similarity with McBride's language in 'Outrageous but Meaningful Coincidences' [19]; in both calculi the aim is to avoid having to deal with type equality by appealing to the propositional equality of the meta-language. Concerning the related problem of encoding different type theories in some meta-type theory, Barras and Werner formalised the calculus of constructions ($\lambda\Pi\omega$) in Coq [7], and more recently Anand and Rahli formalised Nuprl in Coq [5]; even more recently, Abel, Öhman and Vezzosi managed to formalise a full dependent type theory in Agda [2].

Recent work on full completeness (see for example [9]) also seems to be related to our line of work, but I have not determined exactly how.

Bibliography

[1] Andreas Abel. *On Typed Lambda Definability and Normalization by Evaluation*. `www.cse.chalmers.se/~abela/talkAIM27.pdf`. Talk at the Initial Types Seminar. June 2018.

[2] Andreas Abel, Joakim Öhman and Andrea Vezzosi. "Decidability of conversion for type theory in type theory". In: *PACMPL* 2.POPL (2018), 23:1–23:29. DOI: 10.1145/3158111. URL: https://doi.org/10.1145/3158111.

[3] Thorsten Altenkirch and Ambrus Kaposi. "Normalisation by Evaluation for Type Theory, in Type Theory". In: *Logical Methods in Computer Science* 13.4 (2017). DOI: 10.23638/LMCS-13(4:1)2017. URL: https://doi.org/10.23638/LMCS-13(4:1)2017.

[4] Thorsten Altenkirch and Bernhard Reus. "Monadic Presentations of Lambda Terms Using Generalized Inductive Types". In: *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*. Ed. by Jörg Flum and Mario Rodríguez-Artalejo. Vol. 1683. Lecture Notes in Computer Science. Springer, 1999, pp. 453–468. DOI: 10.1007/3-540-48168-0_32. URL: https://doi.org/10.1007/3-540-48168-0%5C_32.

[5] Abhishek Anand and Vincent Rahli. "Towards a Formally Verified Proof Assistant". In: *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*. Ed. by Gerwin Klein and Ruben Gamboa. Vol. 8558. Lecture Notes in Computer Science. Springer, 2014, pp. 27–44. DOI: 10.1007/978-3-319-08970-6_3. URL: https://doi.org/10.1007/978-3-319-08970-6%5C_3.

[6] Henk Barendregt. "Introduction to Generalized Type Systems". In: *J. Funct. Program.* 1.2 (1991), pp. 125–154.

[7] Bruno Barras and Benjamin Werner. *Coq in Coq*. Tech. rep. 1997.

[8] Ulrich Berger and Helmut Schwichtenberg. "An Inverse of the Evaluation Functional for Typed lambda-calculus". In: *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*. IEEE Computer Society, 1991, pp. 203–211. DOI: 10.1109/LICS.1991.151645. URL: https://doi.org/10.1109/LICS.1991.151645.

[9] Valentin Blot and Jim Laird. "Extensional and Intensional Semantic Universes: A Denotational Model of Dependent Types". In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*. Ed. by Anuj Dawar and Erich Grädel. ACM, 2018, pp. 95–104. DOI: 10.1145/3209108.3209206. URL: https://doi.org/10.1145/3209108.3209206.

[10] Ana Bove, Peter Dybjer and Ulf Norell. "A Brief Overview of Agda – A Functional Language with Dependent Types". In: *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*. Ed. by Stefan Berghofer et al. Vol. 5674. Lecture Notes in Computer Science. Springer, 2009, pp. 73–78. DOI: 10.1007/978-3-642-03359-9_6. URL: https://doi.org/10.1007/978-3-642-03359-9%5C_6.

[11] N.G. de Bruijn. "Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem". In: *Indagationes Mathematicae (Proceedings)* 75.5 (1972), pp. 381–392. ISSN: 1385-7258. DOI: https://doi.org/10.1016/1385-7258(72)90034-0. URL: http://www.sciencedirect.com/science/article/pii/1385725872900340.

[12] Catarina Coquand. "A Formalised Proof of the Soundness and Completeness of a Simply Typed Lambda-Calculus with Explicit Substitutions". In: *Higher-Order and Symbolic Computation* 15.1 (2002), pp. 57–90. DOI: 10.1023/A:1019964114625. URL: https://doi.org/10.1023/A:1019964114625.

[13] Nils Anders Danielsson. *A Definability Result*. http://www.cse.chalmers.se/~nad/listings/dependently-typed-syntax/README.DependentlyTyped.Definability.html. Personal Agda Development. July 2019.

[14] Nils Anders Danielsson. "A Formalisation of a Dependently Typed Language as an Inductive-Recursive Family". In: *Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*. Ed. by Thorsten Altenkirch and Conor McBride. Vol. 4502. Lecture Notes in Computer Science. Springer, 2006, pp. 93–109. DOI: 10.1007/978-3-540-74464-1_7. URL: https://doi.org/10.1007/978-3-540-74464-1%5C_7.

[15] Robert Harper, Furio Honsell and Gordon D. Plotkin. "A Framework for Defining Logics". In: *J. ACM* 40.1 (1993), pp. 143–184. DOI: 10.1145/138027.138060. URL: https://doi.org/10.1145/138027.138060.

[16] Robert Harper and Frank Pfenning. "On equivalence and canonical forms in the LF type theory". In: *ACM Trans. Comput. Log.* 6.1 (2005), pp. 61–101. DOI: 10.1145/1042038.1042041. URL: https://doi.org/10.1145/1042038.1042041.

[17] Achim Jung and Jerzy Tiuryn. "A New Characterization of Lambda Definability". In: *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*. Ed. by Marc Bezem and Jan Friso Groote. Vol. 664. Lecture Notes in Computer Science.

Springer, 1993, pp. 245–257. DOI: 10.1007/BFb0037110. URL: https://doi.org/10.1007/BFb0037110.

[18] Per Martin-Löf. "An Intuitionistic Theory of Types: Predicative Part". In: (1975). Ed. by H. E. Rose and J. C. Shepherdson, pp. 73–118.

[19] Conor McBride. "Outrageous but Meaningful Coincidences: Dependent Type-Safe Syntax and Evaluation". In: *Proceedings of the ACM SIGPLAN Workshop on Generic Programming, WGP 2010, Baltimore, MD, USA, September 27-29, 2010*. Ed. by Bruno C. d. S. Oliveira and Marcin Zalewski. ACM, 2010, pp. 1–12. DOI: 10.1145/1863495.1863497. URL: https://doi.org/10.1145/1863495.1863497.

[20] John C. Mitchell and Eugenio Moggi. "Kripke-Style Models for Typed lambda Calculus". In: *Ann. Pure Appl. Logic* 51.1-2 (1991), pp. 99–124. DOI: 10.1016/0168-0072(91)90067-V. URL: https://doi.org/10.1016/0168-0072(91)90067-V.

[21] Gordon David Plotkin. *Lambda-definability and Logical Relations*. Tech. rep. SAI-RM-4. School of Artificial Intelligence, University of Edinburgh, Oct. 1973.

[22] Alan M. Turing. "Computability and λ-Definability". In: *J. Symb. Log.* 2.4 (1937), pp. 153–163. DOI: 10.2307/2268280. URL: https://doi.org/10.2307/2268280.

[23] Various. *Agda's Documentation*. https://agda.readthedocs.io/en/latest/index.html.

[24] Pawel Wieczorek and Dariusz Biernacki. "A Coq formalization of normalization by evaluation for Martin-Löf type theory". In: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*. Ed. by June Andronick and Amy P. Felty. ACM, 2018, pp. 266–279. DOI: 10.1145/3167091. URL: https://doi.org/10.1145/3167091.

Part III

Appendix

Chapter 9

Termination Problems

9.1 Termination Measures

Define the following measures on type families and weakenings, respectively.

$$\begin{aligned} \text{size} &: \forall\{\Gamma\}(A : \text{Fam}\,\Gamma) \to \mathbb{N} \\ \text{size}\,(\text{dat}\,C\,\sigma) &= 1 \\ \text{size}\,(\text{pi}\,A\,B) &= 1 + \text{size}\,A + \text{size}\,B \end{aligned}$$

$$\begin{aligned} \text{size}^{\subseteq} &: \forall\{\Gamma\,\Delta\}(i : \Gamma \subseteq \Delta) \to \mathbb{N} \\ \text{size}^{\subseteq}\,\text{refl} &= 1 \\ \text{size}^{\subseteq}\,(\downarrow i) &= 1 + \text{size}\,i \\ \text{size}^{\subseteq}\,(\uparrow i) &= \text{size}\,A + \text{size}\,i, \quad \text{where } A \text{ is fixed by} \uparrow i : (\Gamma, A) \to (\Delta, A{\downarrow_i}) \end{aligned}$$

Note that, by their definition, our sizes are compatible with structural recursion, i.e. if B is a structurally smaller type than A, then $\text{size}\,B < \text{size}\,A$, and the same holds for weakenings.

9.2 Weakening.agda: Weakening of Type Families

The mutually recursive Agda definitions of $\_{\downarrow}\_$, $\langle\_\rangle^{\text{weak}}$ and weak-sub contain the following recursive calls:

- $(\text{pi}\,A\,B){\downarrow_i} \longrightarrow A \downarrow i$ is structurally recursive.
- $(\text{pi}\,A\,B){\downarrow_i} \longrightarrow B{\downarrow_{\uparrow i}}$, where $\uparrow i : (\Gamma, A) \subseteq (\Delta, A)$ for some Γ and Δ. Here the size of the arguments decreases since

$$\begin{aligned} \text{size}\,(\text{pi}\,A\,B) + \text{size}\,i &= 1 + \text{size}\,A + \text{size}\,B + \text{size}\,i \\ &> \text{size}\,B + \text{size}\,(\uparrow i) = \text{size}\,A + \text{size}\,B + \text{size}\,i. \end{aligned}$$

- $(\text{dat}\,C\,\sigma){\downarrow_i} \longrightarrow \langle i \rangle^{\text{weak}}$. Here $\text{size}\,(\text{dat}\,C\,\sigma) + \text{size}\,i = 1 + \text{size}\,i > \text{size}\,i$.

- $\langle \downarrow i \rangle^{\text{weak}} \longrightarrow \langle i \rangle^{\text{weak}}$ is structurally recursive.
- $\langle \uparrow i \rangle^{\text{weak}} \longrightarrow \langle i \rangle^{\text{weak}}$ is structurally recursive.
- $\langle \uparrow i \rangle^{\text{weak}} \longrightarrow \text{weak-sub}\, A\, i$, where $\uparrow i : (\Gamma, A) \subseteq (\Delta, A)$ for some Γ and Δ. Here the size of the arguments stays the same, since

  $$\text{size}\,(\uparrow i) = \text{size}\, A + \text{size}\, i$$

  by definition. This is not a problem, since the termination measure decreases in all other recursive calls (and hence there are no non-decreasing cycles).
- $\text{weak-sub}\,(\text{pi}\, A\, B)\, i \longrightarrow A{\downarrow_i}$ is structurally recursive.
- $\text{weak-sub}\,(\text{pi}\, A\, B)\, i \longrightarrow B{\downarrow_{\uparrow i}}$, where $\uparrow i : (\Gamma, A) \subseteq (\Delta, A)$. Here we have again

  $$\begin{aligned}\text{size}\,(\text{pi}\, A\, B) + \text{size}\, i &= 1 + \text{size}\, A + \text{size}\, B + \text{size}\, i \\ &> \text{size}\, B + \text{size}\,(\uparrow i) = \text{size}\, A + \text{size}\, B + \text{size}\, i.\end{aligned}$$

9.3 KripkePredicate.agda: Kripke Predicates

We use the fact that weakening preserves our size measure of types.

Lemma. *For every type family* $A : \text{Fam}\,\Gamma$ *and order-preserving embedding* $i : \Gamma \subseteq \Delta$, *size* $(A{\downarrow_i}) =$ *size* A.

Proof. The fact that substitution preserves sizes is immediate from the definition of $\_/\_$. Since weak-den gives us that $A{\downarrow_i} = A/\langle i \rangle^{\text{weak}}$ we have that weakening also preserves sizes. □

The definition of $P[\![\__2]\!]_{\__1} : (\Gamma : \text{Cxt}) \to \Gamma(A : \text{Fam}\,\Gamma) \to [\![A]\!]_\Gamma \to \text{Set}$ can now be easily seen to be terminating since the recursive calls smaller arguments:

- $\mathcal{P}[\![\text{pi}\, A\, B]\!]_\Gamma = \ldots \mathcal{P}[\![A{\downarrow_i}]\!]_\Delta \ldots$ Here $\text{size}\,(A{\downarrow_i}) = \text{size}\, A$.
- $\mathcal{P}[\![\text{pi}\, A\, B]\!]_\Gamma = \ldots \mathcal{P}[\![(B{\downarrow_{\uparrow i}})/_x a]\!]_\Delta \ldots$ Recall that $\_/_x\, a$ was just syntactic sugar for the substitution $\_/(\lambda\gamma \to (\gamma, a\,\gamma))$, and therefore $\text{size}\,((B{\downarrow_{\uparrow i}})/_x a) = \text{size}\, B$.

Chapter 10

Paper Solutions to the Remaining Holes in the Agda Proofs

10.1 TermWeakening.agda: weak-var-den

Goal #0, line 93: It remains to show (ignoring intensional wrestling with equalities) that

$$\pi_2 = \pi_2 \circ (\langle i \rangle^{\mathrm{weak}} \times \mathrm{id}).$$

Note that the two projections have different implicit arguments: on the left-hand side we have

$$\pi_2 : (p : \Sigma\, [\Delta]^*\, [A{\downarrow_i}]) \to [A{\downarrow_i}]\, (\pi_1\, p),$$

while on the right-hand side

$$\pi_2 : (p : \Sigma\, [\Gamma]^*\, [A]) \to [A]\, (\pi_1\, p).$$

In our extensional on-paper type theory the missing equality holds trivially:

$$\begin{aligned} \pi_2 \circ (\langle i \rangle^{\mathrm{weak}} \times \mathrm{id}) &= \pi_2 \circ (\lambda p \to \langle i \rangle^{\mathrm{weak}}\, (\pi_1\, p), \pi_2\, p) \\ &= (\lambda p \to \pi_2\, p) \\ &= \pi_2 \end{aligned}$$

Goal #1, line 132: It remains to show (again the projections have different implicit arguments):

$$\langle v \rangle^{\mathrm{var}} \circ \langle i \rangle^{\mathrm{weak}} \circ \pi_1 = \langle v \rangle^{\mathrm{var}} \circ \pi_1 \circ (\langle i \rangle^{\mathrm{weak}} \times \mathrm{id})$$

Again trivial extensionally:

$$\begin{aligned} &\langle v \rangle^{\mathrm{var}} \circ \pi_1 \circ (\langle i \rangle^{\mathrm{weak}} \times \mathrm{id}) \\ &= \langle v \rangle^{\mathrm{var}} \circ \pi_1 \circ (\lambda p \to (\langle i \rangle^{\mathrm{weak}}\, (\pi_1\, p), \pi_2\, p)) \\ &= \langle v \rangle^{\mathrm{var}} \circ (\lambda p \to \langle i \rangle^{\mathrm{weak}}\, (\pi_1\, p)) \\ &= \langle v \rangle^{\mathrm{var}} \circ \langle i \rangle^{\mathrm{weak}} \circ \pi_1 \end{aligned}$$

10.2 KripkePredicate.agda: mon

It remains to show

$$\text{apply}\,(f \nearrow (i{\cdot}i'))\,x \in \mathcal{P}[\![(B{\downarrow_{\uparrow(i\cdot i')}})/_x x]\!]_\Omega = \text{apply}\,(f \nearrow i \nearrow i')\,x \in \mathcal{P}[\![(B{\downarrow_{\uparrow i}}{\downarrow_{\uparrow i'}})/_x x]\!]_\Omega.$$

This follows from the following two observations:

1. $f \nearrow (i \cdot i') = f \nearrow i \nearrow i'$
   (a) $f \nearrow (i \cdot i') = f \circ \langle i \cdot i' \rangle^{\text{weak}}$
   (b) $\cdots = f \circ \langle i \rangle^{\text{weak}} \circ \langle i' \rangle^{\text{weak}}$
   By weak-comp-den.
   (c) $\cdots = f \nearrow i \nearrow i'$
2. $B{\downarrow_{\uparrow(i\cdot i')}} = B{\downarrow_{\uparrow i}}{\downarrow_{\uparrow i'}}$
   (a) $B{\downarrow_{\uparrow(i\cdot i')}} = B{\downarrow_{(\uparrow i)\cdot(\uparrow i')}}$
   (b) $B{\downarrow_{(\uparrow i)\cdot(\uparrow i')}} = B{\downarrow_{\uparrow i}}{\downarrow_{\uparrow i'}}$
   By weak-comp.