LILY Operational Semantics and Models of Linear Abadi-Plotkin Logic

A Technical Report at the ITU

L. Birkedal
R.L. Petersen
R.E. Møgelberg
C. Varming

IT University Technical Report Series

TR-2006-83

ISSN 1600–6100 **February 2006**

ISSN 1600–6100

ISBN 87-7949-122-7

Copies may be obtained by contacting:

IT University of Copenhagen
Glentevej 67
DK-2400 Copenhagen NV
Denmark

Telephone: **+45 38 16 88 88**
Telefax: **+45 38 16 88 99**
Web `www.itu.dk`

Abstract

This technical report includes complementary material to the article of the same name submitted for LICS. It has been assembled mostly to aid referees reading the article, and to demonstrate that all calculations have been carried out in detail.

Chapter 1

Introduction

The combination of parametric polymorphism and recursion on the level of terms yields a type theory expressive enough to solve general recursive type equations[15, 6]. However, as realized by Plotkin [15], for this combination to give a consistent theory, the parametricity principle must be weakened, and so he suggested studying a dual intuitionistic / linear type theory in combination with parametric polymorphism, in which for example the parametricity principle would apply to graphs of linear, but not intuitionistic maps.

Based on these ideas, in 2000 Bierman, Pitts, and Russo [3] presented the programming language LILY, a polymorphic intuitionistic / linear lambda calculus endowed with an operational semantics. In their paper, they sketched how to carry out Plotkin's ideas in the operational setting. The formulation of the parametricity principle was based on the notion of $\top\top$-closed relations, and the idea was to use the strong connection between these and ground contextual equivalence, to show correctness of recursive type encodings up to ground contextual equivalence. In their paper, however, only correctness of the encodings of sum types was proved.

In recent work the first three authors [6, 9] have presented an adaptation of Abadi & Plotkin's logic for parametricity [14] to PILL_Y, a polymorphic dual intuitionistic / linear type theory with fixed points. The resulting logic — called LAPL for Linear Abadi and Plotkin Logic — contains an axiomatized abstract notion of admissible relations on which the formulation of the parametricity principle is based. Admissible relations give the necessary weakening of the parametricity principle. In *loc. cit.* we showed in detail, following Plotkin's suggestions, that LAPL can be used to define a wide range of types, including general nested recursive types. We also defined a sound and complete class of categorical models for LAPL called LAPL-structures,

In this paper we present a concrete LAPL-structure constructed from the operational semantics of LILY. This defines formally an interpretation of LAPL into LILY, transferring the general results proved abstractly in LAPL to LILY. The construction of the LAPL-structure involves showing that the $\top\top$-closed relations define a notion of admissible relations, i.e., formally showing that they can be used for reasoning about parametricity.

This new model of LAPL based on the operational semantics of LILY is of interest for the following reasons:

- It shows that *our definition of LAPL in [6] is* indeed a *general* one: In [6] we presented a model based on admissible partial equivalence relations over a universal model of the untyped lambda calculus and in [12] we presented a model based on synthetic domain theory.

- The present model is *simpler* than our previous models mentioned above in that it does not require any

domain theory, realizability, or synthetic domain theory. We hope this will make it more accessible. To make the model accessible also for readers who are not that familiar with category theory we deliberately choose to show not only the necessary categorical properties required of an LAPL-structure but also sketch in more explicit terms the interpretation of LAPL in our new model.

- The previous models mentioned above were constructed via a so-called parametric completion process, which roughly means that they were constructed in two steps: first a simple non-parametric model was constructed and then it was made parametric by filtering out all the non-parametric elements. (See [9] for more on parametric completions.) The present model is the first such[1] that is *not* based on a parametric completion process.

- It allows us to conclude that a wide range of types are definable up to ground contextual equivalence in LILY, as claimed but not formally proved in [3] (except for the simple case of finite coproducts). Hence our model can be used to prove correct program transformations based on parametricity for a language with general recursive types, an improvement over earlier work [7], which only dealt with algebraic data types.

  In fact, we also proved this LILY definability of types in [12] from the model based on synthetic domain theory by combining it with an adequate denotational semantics of LILY, but here we can do it by much simpler techniques.

- The model we present here is, to our knowledge, the first model of general recursive types as formalized via algebraic compactness based on operational semantics. Algebraic compactness is a categorical formulation of what it means to solve recursive domain equations; it ensures that the solutions are universal in an appropriate categorical sense, thus allowing for the derivation of (mixed) inductive-coinductive reasoning principles, c.f. [13]. In earlier work by the first author and Harper [4] a reasoning principle similar to the one derived here was presented for a recursive type, but it only worked for a single top-level recursive type. For other related work on operational models of recursive types, e.g., [1, 8], one may probably also establish useful reasoning principles for the recursive types but as far as we know it has not been done and, at any rate, it is pleasing that the reasoning principles presented here are an immediate consequence of general results about LAPL-structures.

The remainder of this report is organized as follows.

In Chapter 2 we recall the notion of an LAPL structure.

In Chapter 3 we then derive reasoning principles for recursive types definable in such LAPL-structures.

In Chapter 4 we extend the LILY language with tensor types and associated terms. Moreover, we extend the operational properties of LILY studied in [3] to the new language and slightly generalize the results in [3] (to work for terms with free term and type variables, e.g.) and include a few new ones needed for the model construction. We have included tensor types, even though we in the end show that they are definable using parametricity, because it eases the construction of the model.

In Chapter 5 we construct a PILL_Y-model from the operational semantics of our extension of LILY. This is the first step in the construction of the LAPL-structure. The next step in the construction is the logic fibration. The interpretation of the logic is basically set theoretic, interpreting propositions on types, for example, as subsets of sets of ground contextual equivalence classes of terms. As mentioned, the $\top\top$-closed relations play the role of admissible relations in LAPL, and thus need to satisfy certain closure properties. These are also established in Chapter 5.

[1] Except for the syntactic one constructed to prove completeness of LAPL-structures.

The last step in the construction of the LAPL-structure is the relational interpretation of types. We show how the inductive definition of the interpretation of LILY-types as $\perp\!\!\perp$-closed relations presented in [3] defines a relational interpretation of types satisfying the requirements for LAPL-structures. Finally, it is shown that the parametricity schema does indeed hold in the constructed LAPL-structure.

The chapter concludes with a description of the interpretation of PILL_Y into the model we have constructed out of the operational semantics of LILY.

We then conclude in Chapter 6

Chapter 2

LAPL-structures

The equational theory PILL_Y is a variant of DILL [2] extended with polymorphism and fixed points given by a fixed point combinator of type $\prod \alpha.(\alpha \to \alpha) \to \alpha$, where in general we use $\sigma \to \tau$ as notation for $!\sigma \multimap \tau$.

We start off by sketching the notion of LAPL-structure as described in [6]. Some readers may prefer to first look at the concrete LAPL-structure constructed in Chapter 5 and the explicit description of the interpretation in the end and then refer back to this section later. LAPL-structures model a variant of Abadi & Plotkin's logic for parametricity [16, 15] designed for reasoning about PILL_Y. Propositions in the logic exist in contexts of free type variables, free variables of PILL_Y and free relational variables. The free relational variables may be relations or admissible relations. Propositions are written as

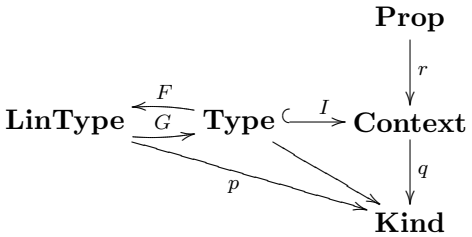
$$\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \mid \vec{R}\colon \mathsf{Rel}(\vec{\sigma}, \vec{\sigma}'), \vec{S}\colon \mathsf{AdmRel}(\vec{\tau}, \vec{\tau}') \vdash \phi\colon \mathsf{Prop}.$$

The vector $\vec{\alpha}$ is a list of free type variables. We will not describe the logic in details, but only mention a few main points. The variables $\vec{x}\colon \vec{\sigma}$ are treated intuitionistically in the logic. We may reason about linear terms by for example using variables of type $\sigma \multimap \tau$, but the reasoning about the terms is purely intuitionistic.

The logic comes equipped with a notion of admissible relations, which is required to be closed under certain constructions. For example, equality relations (relating equal elements of some type) are required to be admissible, and admissible relations must be closed under reindexing along *linear* maps and universal quantification.

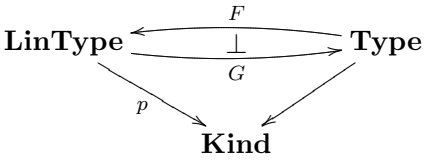
For any type $\vec{\alpha} \vdash \sigma(\vec{\alpha})\colon \mathbf{Type}$ with n free variables, and any n-vector of *admissible* relations $\vec{R}\colon \mathsf{AdmRel}(\vec{\tau}, \vec{\tau}')$, we can form an admissible relation $\sigma[\vec{R}]\colon \mathsf{AdmRel}(\sigma(\vec{\tau}), \sigma(\vec{\tau}'))$. This is called the relational interpretation of σ, and it is important for reasoning about parametricity. For example we can express the *identity extension schema* [17] as $\sigma[eq_{\vec{\alpha}}] \equiv eq_{\sigma(\vec{\alpha})}$, which we use as our definition of parametricity.

A **pre-LAPL**-structure is a schema of categories and functors



(2.1)

such that the diagram



(2.2)

is a model of PILL_Y [10] (a fibred version of a model of DILL [2], with generic object $\Omega \in \mathbf{Kind}$ for p, simple products modeling polymorphism in p, and a term modeling the fixed point combinator).

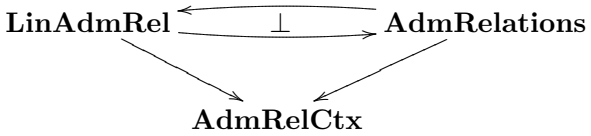
We further require that the fibration q has fibred products and that I is a faithful map preserving fibred products. The pair of fibrations (r, q) is an indexed first-order logic fibration which has products and coproducts with respect to projections $\Xi \times \Omega \to \Xi$ in $\mathbf{Kind}$ [5], meaning that each fibre of r over an object Ξ in $\mathbf{Kind}$ is a first-order logic fibration with structure preserved under reindexing, and that the logic models quantification along the mentioned projections in $\mathbf{Kind}$.

Finally, there should exist a fibred functor U mapping pairs of types σ, τ in the same fibre of $\mathbf{LinType}$ to an object $U(\sigma, \tau)$ in $\mathbf{Context}$ acting as an object of all relations from $IG(\sigma)$ to $IG(\tau)$ in the logic of $\mathbf{Prop}$.

A notion of admissible relations for a pre-LAPL-structure is a subfunctor V of U closed under the constructions for admissible relations in the logic.

A pre-LAPL-structure models Abadi & Plotkin's logic for parametricity, except for the relational interpretation of types. The contexts of the logic are modeled in $\mathbf{Context}$ using U, V to model the sets of all relations and admissible relations between types respectively. The propositions of the logic are modeled in $\mathbf{Prop}$.

From a pre-LAPL-structure with a notion of admissible relations one can define a PILL model (a PILL_Y model that does not necessarily model Y)



(2.3)

of admissible relations. There exists two maps of PILL-models ∂_0, ∂_1 from (2.3) to (2.2) basically mapping a relation to its domain and codomain respectively. An LAPL-structure is a pre-LAPL-structure with a notion of admissible relations and a map of PILL-models J from (2.2) to (2.3) such that

$$\partial_0 \circ J = \partial_1 \circ J = \mathsf{id}.$$

The functor J models the relational interpretation of types. It enables us to talk about parametricity at all types in the model, not just the interpretations of types in *pure* PILL_Y.

A **parametric LAPL-structure** is an LAPL-structure satisfying the identity extension schema in the internal logic. Moreover the extensionality schemes

$$\forall x\colon \sigma. f(x) =_\tau g(x) \supset f =_{\sigma\to\tau} g$$
$$\forall \alpha\colon \mathbf{Type}. t\ \alpha =_\sigma u\ \alpha \supset t =_{\prod \alpha.\sigma} u,$$

should hold and the model should have very strong equality. The latter means that if two terms of PILL_Y are provably equal in the logic, then they are in fact equal in the model.

Parametric LAPL-structures are interesting because we can reason about the contained PILL_Y-model using parametricity. In particular, we can solve a large class of domain equations in parametric LAPL-structures, and show that a large class of endo-functors have initial algebras and final coalgebras. These results are presented in Chapter 3.

Chapter 3

Reasoning Principles

In this chapter we include a note by Rasmus Møgelberg deriving reasoning principles for recursive types, valid in any parametric LAPL-structure.

Induction and coinduction principles for recursive types in LAPL

Rasmus E. Møgelberg*
mogel@itu.dk
DISI, Universitá di Genova
Italy

September 12, 2005

Abstract

We investigate induction and coinduction principles for inductive, coinductive and recursive types encoded in LAPL using parametricity. Using a closure operator on relations giving the least admissible relation containing the given relation, we show that coinduction principles work for general relations and not just admissible relations.

1 Introduction

In this document, we will assume that we are given a parametric LAPL-structure.

Note for readers: Section 3 seems a bit out of place here. Consider skipping it on arst reading.

2 A closure operator on relations

Lemma 2.1

If ρ is an admissible relation, and ρ' is any relation, $\rho' \multimap \rho$ is admissible.

*This work is sponsored by Danish Research Agency stipend no. 272-05-0031

Proof. The proof from [1, Prop 2.3] goes through without the assumption that ρ' is admissible. □

Lemma 2.1 shows that inductively constructed types induce what Pitts calls admissible actions on relations [6, Def 4.6]. In fact the statement of Lemma 2.1 is much stronger than that.

We will now define an operation on relations, associating to a relation the smallest admissible relation containing it. First we need some general definitions for the formulation of the theorem. Define the indexed category $\mathbf{Relations} \to \mathbf{Kind}$, to have as objects over Ξ triples (σ, τ, R), where $\Xi \vdash R\colon \mathrm{Rel}(\sigma, \tau)$, and as maps pairs of maps preserving relations. The indexed category $\mathbf{AdmRelations} \to \mathbf{Kind}$ is the restriction of $\mathbf{Relations} \to \mathbf{Kind}$ to admissible relations.

Suppose $R\colon \mathrm{Rel}(\sigma, \tau)$ is any relation. By Lemma 2.1 above,

$$\forall \alpha, \beta, S\colon \mathrm{AdmRel}(\alpha, \beta).\, (R \multimap S) \multimap S \tag{1}$$

is an admissible relation from $\prod \alpha.\, (\sigma \multimap \alpha) \multimap \alpha$ to $\prod \alpha.\, (\tau \multimap \alpha) \multimap \alpha$. By [1, Prop 3.4], in a parametric LAPL-structure $\sigma \cong \prod \alpha.\, (\sigma \multimap \alpha) \multimap \alpha$, and so we may pull back (1) along the isomorphisms to get an admissible relation $\Phi(R)\colon \mathrm{AdmRel}(\sigma, \tau)$.

Theorem 2.2. *The association $R \mapsto \Phi(R)$ defines a fibred left adjoint to the forgetful functor $\mathbf{AdmRelations} \to \mathbf{Relations}$. In other words, for R any relation, $\Phi(R)$ is the smallest admissible relation containing R.*

Proof. The map Φ is functorial since S appears only positively in (1).

We show that $R \subset \Phi(R)$. The isomorphism $\sigma \multimap \prod \alpha.\, (\sigma \multimap \alpha) \multimap \alpha$ is given by the lambda expression $\xi = \lambda^\circ x\colon \sigma.\, \Lambda\alpha.\, \lambda^\circ f\colon \sigma \multimap \alpha.\, f\ x$. If $R(x, y)$ and $(f, g)\colon R \multimap S$ then $(f\ x, g\ y) \in S$, and so $R(x, y)$ implies

$$(\xi\ x, \xi\ y) \in \forall \alpha, \beta, S\colon \mathrm{AdmRel}(\alpha, \beta).\, (R \multimap S) \multimap S.$$

This shows that $R \subset \Phi(R)$.

Finally, since the types $\beta \vdash \beta$ and $\beta \vdash \prod \alpha.\, (\beta \multimap \alpha) \multimap \alpha$ are isomorphic, their relational interpretations are isomorphic, and so for any admissible relation S:

$$S = (\xi, \xi)^*(\forall \alpha, \beta, S\colon \mathrm{AdmRel}(\alpha, \beta).\, (R \multimap S) \multimap S) = \Phi(S).$$

Thus, if $R\colon \mathrm{Rel}(\sigma, \tau)$ is any relation and $S\colon \mathrm{AdmRel}(\sigma, \tau)$ is admissible then $R \subset S$ implies $\Phi(R) \subset \Phi(S) = S$, and on the other hand, $\Phi(R) \subset S$ implies $R \subset S$ since $R \subset \Phi(R)$. □

3 Existensial types

In [8] Plotkin sketched how to define existensial types from polymorphism using parametricity as

$$\coprod \alpha.\, \sigma(\alpha) = \prod \beta.\, (\prod \alpha.\, \sigma(\alpha) \multimap \beta) \multimap \beta$$

In [1, Sec 3.9] working in the logic LAPL, Birkedal, Møgelberg and Petersen show that this definition is correct in the categorical sense, by showing that it defines a left adjoint to weakening. Put more precisely, this means that if $\Xi, \alpha \vdash \sigma$ is a type, and $\Xi \vdash \tau$ is a type, there is a natural bijective correspondence between terms $\Xi \vdash t\colon \coprod \alpha.\, \sigma \multimap \tau$ and terms $\Xi, \alpha \vdash \hat{t}\colon \sigma \multimap \tau$. The goal of this section is to deduce the natural reasoning principle for existensial types.

Before we state the reasoning principle, we recall the constructor for existensial types:

$$\mathit{pack}\colon \prod \alpha.\, \sigma(\alpha) \multimap \coprod \alpha.\, \sigma(\alpha).$$

Basically, *pack* takes a type α and an element of type $\sigma(\alpha)$ and packs it into the existensial type. It is defined as follows:

$$\mathit{pack} = \Lambda\alpha.\, \lambda^\circ x\colon \sigma(\alpha).\, \Lambda\beta.\, \lambda^\circ f\colon \prod \gamma.\, \sigma(\gamma) \multimap \beta.\, f\ \alpha\ x$$

Theorem 3.1. *For $x, y\colon \coprod \alpha.\, \sigma(\alpha)$ the following is equivalent to internal equality of x and y.*

$$\exists \alpha, \beta, R\colon \mathit{AdmRel}(\alpha, \beta), x'\colon \sigma(\alpha), y\colon \sigma(\beta).\, x = \mathit{pack}\alpha x' \wedge y = \mathit{pack}\beta y' \wedge \sigma[R](x', y').$$

As a special case we get the following principle:

$$\forall x\colon \coprod \alpha.\, \sigma(\alpha).\, \exists \alpha, x'\colon \sigma(\alpha).\, x =_{\coprod \alpha.\sigma(\alpha).\exists \alpha} \mathit{pack}\ \alpha\ x'$$

Proof. We first show the case of admissible relations.

Recall from [1, Sec 3.9] that if $\Xi \vdash t\colon \coprod \alpha.\, \sigma \multimap \tau$, then $\Xi, \alpha \vdash \hat{t}\colon \sigma \multimap \tau$ is defined as $\hat{t}(x) = t(\mathit{pack}\ \alpha\ x)$. On the other hand, if $\Xi, \alpha \vdash s\colon \sigma \multimap \tau$ then $\Xi \vdash \hat{s}\colon \coprod \alpha.\, \sigma \multimap \tau$ is defined as $\hat{s}(x) = x\ \tau\ (\Lambda\alpha.\, s)$.

Let us for simplicity write $\chi(R)$ for

$$(x, y).\, \exists \alpha, \beta, R\colon \mathrm{AdmRel}(\alpha, \beta), x'\colon \sigma(\alpha), y\colon \sigma(\beta).\, x = \mathit{pack}\alpha x' \wedge y = \mathit{pack}\beta y' \wedge \sigma[R](x', y')$$

Now, our aim is to prove that for any pair of types τ, τ' and any admissible relation $S\colon \mathrm{AdmRel}(\tau, \tau')$, and any pair of maps t, t' we have

$$(t, t')\colon \mathit{eq}_{\coprod \alpha.\sigma} \multimap S$$

iff

$$(t.t')\colon \chi(R) \multimap S$$

and the first part of the theorem will follow as an application of the Yoneda Lemma.

First notice that

$$\frac{\Xi \mid x, y \vdash \chi(R)(x,y) \supset S(t(x), t'(y))}{\dfrac{\Xi, \alpha, \beta \mid R\colon \mathrm{AdmRel}(\alpha,\beta) \mid x, y, x', y' \vdash \sigma[R](x',y') \supset S(t(\mathit{pack}\,\alpha\,x'), t'(\mathit{pack}\,\beta\,y'))}{\Xi, \alpha, \beta \mid R\colon \mathrm{AdmRel}(\alpha,\beta) \mid x', y' \vdash \sigma[R](x',y') \supset S(\hat{t}(x'), \hat{t}'(y'))}}$$

so it suffices to show that

$$\frac{\Xi \mid x, y \vdash x =_{\coprod \alpha.\sigma(\alpha)} y \supset S(t(x), t'(y))}{\Xi, \alpha, \beta \mid R\colon \mathrm{AdmRel}(\alpha,\beta) \mid x', y' \vdash \sigma[R](x',y') \supset S(\hat{t}(x'), \hat{t}'(y'))}$$

i.e., that (t, t') preserve relations iff $(\hat{t}, \hat{t}')$ do.

First assume (t, t') preserve relations. By parametricity of *pack*,

$$(\mathit{pack}\,\alpha, \mathit{pack}\,\beta)\colon \sigma[R] \multimap \mathit{eq},$$

and so since $\hat{t} = t \circ (\mathit{pack}\,\alpha)$ and $\hat{t}' = t' \circ (\mathit{pack}\,\beta)$ the pair $(\hat{t}, \hat{t}')$ preserve relations. On the other hand, if $(\hat{t}, \hat{t}')$ preserve relations then

$$(\Lambda\alpha.\,\hat{t}, \Lambda\beta.\,\hat{s})\colon \forall\alpha, \beta, R\colon \mathrm{AdmRel}(\alpha,\beta).\,\sigma[R] \multimap S,$$

and so by parametricity, if $\mathit{eq}_{\coprod \alpha.\sigma(\alpha)}(x, y)$ then

$$(t(x), t'(y)) = (x \coprod \alpha.\,\sigma(\alpha)\ (\Lambda\alpha.\,\hat{t}), y \coprod \alpha.\,\sigma(\alpha)\ (\Lambda\beta.\,\hat{t}')) \in S$$

□

Remark 3.2. It may seem more natural to first prove the reasoning principle for existensial types, and then use that for proving the categorical properties. I believe that this is what Plotkin and Abadi did in the case of the second order lambda calculus [9]. In this paper we have done the opposite, namely used the categorical properties to prove the reasoning principle. The reason for this is that admissible relations are not assumed to be closed under existensial quantification, and so we cannot induce the reasoning principle by applying parametricity to a relation like $\chi(R)$.

4 Induction and coinduction priciples

In this section we establish induction and coinduction principles for respectively inductive and coinductive types encoded using parametricity. This supplements the treatment of inductive and coinductive types in [1], where only categorical properties of these types were discussed.

We start with inductive types. Suppose $\alpha \vdash \sigma(\alpha)$ is a type expression in which α occurs only positively, such that σ induces a functor from types to types. In [1], following [8] we define the type

$$\mu\alpha.\,\sigma(\alpha) = \prod \alpha.\,(\sigma(\alpha) \multimap \alpha) \to \alpha$$

and a term

$$\mathit{in}\colon \sigma(\mu\alpha.\,\sigma(\alpha)) \multimap \mu\alpha.\,\sigma(\alpha)$$

which we show is an initial algebra for the functor induced by σ. We now show the following (relational) induction principle.

Theorem 4.1 (Induction). *Suppose* $R\colon \mathit{AdmRel}(\mu\alpha.\,\sigma(\alpha), \mu\alpha.\,\sigma(\alpha))$ *satisfies*

$$(\mathit{in}, \mathit{in})\colon \sigma[R] \multimap R.$$

Then

$$\forall x\colon \mu\alpha.\,\sigma(\alpha).\,R(x, x)$$

Remark 4.2. The induction principle speaks about relations since it is obtained as a consequence of binary parametricity. In case one also has unary parametricity available (for some notion of admissible propositions), applying the proof of Theorem 4.1 to unary parametricity will yield the well-known propositional induction principle: If ϕ is an *admissible* proposition on $\mu\alpha.\,\sigma(\alpha)$, then

$$(\forall x\colon \sigma(\mu\alpha.\,\sigma(\alpha)).\,\sigma[\phi](x) \supset \phi(\mathit{in}\ x)) \supset \forall x\colon \mu\alpha.\,\sigma(\alpha).\,\phi(x)$$

Proof of Theorem 4.1. By parametricity, for any $x\colon \mu\alpha.\,\sigma(\alpha)$,

$$x(\forall \alpha, \beta, R\colon \mathrm{AdmRel}(\alpha, \beta).\,(\sigma[R] \multimap R) \to R)x$$

The assumption states that $(\mathit{in}, \mathit{in})\colon \sigma[R] \multimap R$ and so by [1, Lemma 2.31],

$$(!\mathit{in}, !\mathit{in})\colon !(\sigma[R] \multimap R).$$

Thus

$$R(x\ \mu\alpha.\,\sigma(\alpha)\ !\mathit{in}, x\ \mu\alpha.\,\sigma(\alpha)\ !\mathit{in}).$$

Finally, [1, Lemma 3.25] tells us that $x\mu\alpha.\,\sigma(\alpha)!\mathit{in} = x$, which proves the theorem. □

We now turn to coinduction principles for coinductive types. Again we assume that we are given a type $\alpha \vdash \sigma(\alpha)$ with α appearing only positively. Coinductive types are treated in [1, Sec 3.11], where — following [8] — we define the type

$$\nu\alpha.\,\sigma(\alpha) = \coprod \alpha.\,!(\alpha \multimap \sigma(\alpha)) \otimes \alpha$$

and the term

$$\mathit{out}\colon \nu\alpha.\,\sigma(\alpha) \multimap \sigma(\nu\alpha.\,\sigma(\alpha))$$

which we prove is a final coalgebra for the functor induced by σ.

Theorem 4.3 (Coinduction). *Suppose* $R\colon$ *AdmRel*$(\nu\alpha.\,\sigma(\alpha), \nu\alpha.\,\sigma(\alpha))$ *is such that* $(\mathit{out}, \mathit{out})\colon R \multimap \sigma[R]$*, then*

$$\forall x, y\colon \nu\alpha.\,\sigma(\alpha).\, R(x,y) \supset x =_{\nu\alpha.\sigma(\alpha)} y$$

Proof. Suppose $R\colon$ AdmRel$(\nu\alpha.\,\sigma(\alpha), \nu\alpha.\,\sigma(\alpha))$ is such that $(\mathit{out}, \mathit{out})\colon R \multimap \sigma[R]$ and $R(x,y)$. By parametricity of

$$\mathit{pack}\colon \textstyle\prod \alpha.\, !(\alpha \multimap \sigma(\alpha)) \otimes \alpha \multimap \nu\alpha.\,\sigma(\alpha)$$

we have

$$\mathit{pack}\ \nu\alpha.\,\sigma(\alpha)\ !\mathit{out} \otimes x =_{\nu\alpha.\sigma(\alpha)} \mathit{pack}\ \nu\alpha.\,\sigma(\alpha)\ !\mathit{out} \otimes y$$

and by [1, Lem 3.29],

$$\begin{array}{c} \mathit{pack}\ \nu\alpha.\,\sigma(\alpha)\ !\mathit{out} \otimes x =_{\nu\alpha.\sigma(\alpha)} x \\ \mathit{pack}\ \nu\alpha.\,\sigma(\alpha)\ !\mathit{out} \otimes y =_{\nu\alpha.\sigma(\alpha)} y \end{array}$$

which proves the theorem. □

The next theorem is an interesting generalisation of Theorem 4.3, stating that the assumption of admissibility in the coinduction principle is unnecessary. A similar result was proved by Pitts in the setting of coinductive types in the category of domains [6].

Theorem 4.4 (General coinduction principle). *Suppose* $R\colon$ *Rel*$(\nu\alpha.\,\sigma(\alpha), \nu\alpha.\,\sigma(\alpha))$ *is such that* $(\mathit{out}, \mathit{out})\colon R \multimap \sigma[R]$*, then*

$$\forall x, y\colon \nu\alpha.\,\sigma(\alpha).\, R(x,y) \supset x =_{\nu\alpha.\sigma(\alpha)} y$$

Proof. Suppose $R\colon$ Rel$(\nu\alpha.\,\sigma(\alpha), \nu\alpha.\,\sigma(\alpha))$ is any relation satisfying $(\mathit{out}, \mathit{out})\colon R \multimap \sigma[R]$. The idea of the proof is to use Theorem 4.3 on the admissible relation $\Phi(R)$. Since Φ is a functor,

$$(\mathit{out}, \mathit{out})\colon \Phi(R) \multimap \Phi(\sigma[R]),$$

and since $\sigma[\Phi(R)]$ is an admissible relation containing $\sigma[R]$, and $\Phi(\sigma[R])$ is the smallest such, $\Phi(\sigma[R]) \subset \sigma[\Phi(R)]$ and so

$$(\mathit{out}, \mathit{out})\colon \Phi(R) \multimap \sigma[\Phi(R)].$$

Now, the coindution principle for admissible relations gives us

$$\forall x, y\colon \nu\alpha.\,\sigma(\alpha).\, \Phi(R)(x,y) \supset R(x,y)$$

and so the theorem follows from $R \subset \Phi(R)$. □

5 Recursive types

In this section we consider recursive types defined using parametricity as in [8] and [1, Sec 3.12]. For the moment we only consider recursive types with no parameters. So we consider the situation, where we are given an inductively constructed type

$$\alpha, \beta \vdash \sigma(\alpha, \beta)$$

in which α occurs only negatively and β only positively. Such a type induces a functor in two variables, contravariant in the first and covariant in the second, on the category of types. In *loc. cit.* a type rec $\alpha.\, \sigma(\alpha, \alpha)$ is constructed satisfying

$$\sigma(\text{rec } \alpha.\, \sigma(\alpha, \alpha), \text{rec } \alpha.\, \sigma(\alpha, \alpha)) \cong \text{rec } \alpha.\, \sigma(\alpha, \alpha).$$

This is done using parametricity in combination with Freyds theory of algebraically compact categories [3, 2, 4] as suggested first by Plotkin [8, 7]. In fact, what is proved about rec $\alpha.\, \sigma(\alpha, \alpha)$ is a bit stronger. We prove that it is an initial dialgebra for the functor induced by σ, i.e., that given any pair of types ω, ω' and terms $t\colon \omega \multimap \sigma(\omega', \omega)$,$t'\colon \sigma(\omega, \omega') \multimap \omega'$, there exists unique h, h' making the diagrams

$$\begin{array}{ccc}
\sigma(\text{rec } \alpha.\, \sigma(\alpha, \alpha), \text{rec } \alpha.\, \sigma(\alpha, \alpha)) & \multimap & \text{rec } \alpha.\, \sigma(\alpha, \alpha) \\
{\scriptstyle \sigma(h,h')}\downarrow & & \downarrow{\scriptstyle h'} \\
\sigma(\omega, \omega') & \xrightarrow{\quad t' \quad} & \omega' \\
\\
\omega & \xrightarrow{\quad t \quad} & \sigma(\omega', \omega) \\
{\scriptstyle h}\downarrow & & \downarrow{\scriptstyle \sigma(h',h)} \\
\text{rec } \alpha.\, \sigma(\alpha, \alpha) & \multimap & \sigma(\text{rec } \alpha.\, \sigma(\alpha, \alpha), \text{rec } \alpha.\, \sigma(\alpha, \alpha))
\end{array} \tag{2}$$

commute.

Here we prove the following reasoning principle for the recursive type rec $\alpha.\, \sigma(\alpha, \alpha)$. This principle is the same as the one obtained by Pitts for recursive types in the category domains [6, Cor 4.10]

Theorem 5.1. *Suppose*

$$\begin{array}{l} R^-\colon \mathit{Rel}(\mathit{rec}\ \alpha.\, \sigma(\alpha, \alpha), \mathit{rec}\ \alpha.\, \sigma(\alpha, \alpha)) \text{ and} \\ R^+\colon \mathit{AdmRel}(\mathit{rec}\ \alpha.\, \sigma(\alpha, \alpha), \mathit{rec}\ \alpha.\, \sigma(\alpha, \alpha)) \end{array}$$

are relations. Then the following principle holds

$$\frac{(i^{-1}, i^{-1})\colon R^- \multimap \sigma(R^+, R^-) \qquad (i, i)\colon \sigma(R^-, R^+) \multimap R^+}{R^- \subset \mathit{eq}_{\mathit{rec}\ \alpha.\sigma(\alpha,\alpha)} \subset R^+}$$

where i denotes the isomorphism

$$\sigma(\mathit{rec}\ \alpha.\, \sigma(\alpha, \alpha), \mathit{rec}\ \alpha.\, \sigma(\alpha, \alpha)) \multimap \mathit{rec}\ \alpha.\, \sigma(\alpha, \alpha).$$

Proof. We first prove the rule in the case of both relations being admissible. The proof in this case is a surprisingly simple consequence of parametricity.

The proof that rec $\alpha.\,\sigma(\alpha,\alpha)$ is an initial dialgebra [1, Sec 3.12] is constructive in the sence that there is a construction of the unique maps h, h' satisfying the diagrams (2) above from the given types ω, ω' and terms t, t'. In fact, from the proof we can derive terms

$$\begin{array}{l} k\colon \prod \omega,\omega'.\,(\sigma(\omega,\omega') \multimap \omega') \multimap (\omega \multimap \sigma(\omega',\omega)) \multimap \omega' \multimap \text{rec } \alpha.\,\sigma(\alpha,\alpha) \\ k'\colon \prod \omega,\omega'.\,(\sigma(\omega,\omega') \multimap \omega') \multimap (\omega \multimap \sigma(\omega',\omega)) \multimap \text{rec } \alpha.\,\sigma(\alpha,\alpha) \multimap \omega' \end{array}$$

such that the maps h, h' can be obtained as

$$\begin{array}{rcl} h & = & k\;\omega\;\omega'\;t'\;t \\ h' & = & k'\;\omega\;\omega'\;t'\;t \end{array}$$

The exact constructions of k, k' are not of interest us right now — what matters to us is that we can use the assumption of parametricity on them. We consider the case $\omega = \omega' = \text{rec } \alpha.\,\sigma(\alpha,\alpha)$ and $t = i^{-1}$ and $t' = i$. In this case of course $h = h' = id$. If we use parametricity of k by substituting the relation R^+ for the type ω and R^- for ω' then we get since

$$id = k \text{ rec } \alpha.\,\sigma(\alpha,\alpha) \text{ rec } \alpha.\,\sigma(\alpha,\alpha)\; i^{-1}\; i$$

$(id, id)\colon R^- \multimap \mathit{eq}_{\text{rec } \alpha.\sigma(\alpha,\alpha)}$. Likewise, using parametricity of k' we get

$$(id, id)\colon \mathit{eq}_{\text{rec } \alpha.\sigma(\alpha,\alpha)} \multimap R^+$$

which proves the theorem in the case of R^- being admissible.

For the general case, we just need a simple application of the closure operator of Theorem 2.2. So assume again

$$\begin{array}{l} (i^{-1}, i^{-1})\colon R^- \multimap \sigma(R^+, R^-), \\ (i, i)\colon \sigma(R^-, R^+) \multimap R^+, \end{array}$$

and R^+ is admissible, but R^- may not be. The idea is to use the case above on $\Psi(R^-)$ and R^+ which are both admissible, but we need to check that the hypothesis still holds for this case. First, by Ψ being a functor

$$(i^{-1}, i^{-1})\colon \Psi(R^-) \multimap \Psi(\sigma(R^+, R^-)).$$

But, since $\sigma(R^+, \Psi(R^-))$ is an admissible relation containing $\sigma(R^+, R^-)$,

$$\Psi(\sigma(R^+, R^-) \subset \sigma(R^+, \Psi(R^-))$$

and so

$$(i^{-1}, i^{-1})\colon \Psi(R^-) \multimap \sigma(R^+, \Psi(R^-)). \tag{3}$$

Since $\sigma(\Psi(R^-), R^+) \subset \sigma(R^-, R^+)$ we also have

$$(i, i)\colon \sigma(\Psi(R^-), R^+) \multimap R^+. \tag{4}$$

Using the case of admissible relation proved above on (3) and (4), we get

$$\Psi(R^-) \subset \mathit{eq}_{\mathrm{rec}\,\alpha.\sigma(\alpha,\alpha)} \subset R^+$$

which together with $R^- \subset \Psi(R^-)$ proves the theorem in the general case. □

6 Recursive types with parameters

Recursive types with parameters are also treated in [1, Sec 3.13]. Let us start by recalling the results of *loc. cit.*.

Suppose $\vec{\alpha}, \vec{\beta}, \alpha, \beta \vdash \sigma(\vec{\alpha}, \vec{\beta}, \alpha, \beta)$ is a type in which the variables $\vec{\alpha}, \alpha$ occur only negatively and the variables $\vec{\beta}, \beta$ occur only positively. Then there exists a type $\vec{\alpha}, \vec{\beta} \vdash \tau(\vec{\alpha}, \vec{\beta})$ in which the variables $\vec{\alpha}$ occur only negatively and the variables $\vec{\beta}$ only positively, and an isomorphism:

$$\vec{\alpha}, \vec{\beta} \vdash i\colon \sigma(\vec{\alpha}, \vec{\beta}, \tau(\vec{\beta}, \vec{\alpha}), \tau(\vec{\alpha}, \vec{\beta})) \multimap \tau(\vec{\alpha}, \vec{\beta})$$

In fact, also a parametrized version of (2) is proved, which we repeat here, as it is needed for the proof of Theorem 6.1 below.

For any pair of types ω, ω' and pair of maps

$$\begin{aligned} g &\colon \omega \multimap \sigma(\vec{\beta}, \vec{\alpha}, \omega', \omega) \\ g' &\colon \sigma(\vec{\alpha}, \vec{\beta}, \omega, \omega') \multimap \omega' \end{aligned}$$

there exists unique maps

$$\begin{aligned} h &\colon \omega \multimap \tau(\vec{\beta}, \vec{\alpha}) \\ h' &\colon \tau(\vec{\alpha}, \vec{\beta}) \multimap \omega' \end{aligned}$$

making the diagrams

$$\begin{array}{ccc} \omega & \xrightarrow{\quad g \quad} & \sigma(\vec{\beta}, \vec{\alpha}, \omega', \omega) \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle \sigma(\vec{\beta},\vec{\alpha},h',h)} \\ \tau(\vec{\beta}, \vec{\alpha}) & \xrightarrow{\quad i \quad} & \sigma(\vec{\beta}, \vec{\alpha}, \tau(\vec{\alpha}, \vec{\beta}), \tau(\vec{\beta}, \vec{\alpha})) \end{array}$$

$$\begin{array}{ccc} \sigma(\vec{\alpha}, \vec{\beta}, \tau(\vec{\beta}, \vec{\alpha}), \tau(\vec{\alpha}, \vec{\beta})) & \xrightarrow{\quad i \quad} & \tau(\vec{\alpha}, \vec{\beta}) \\ {\scriptstyle \sigma(\vec{\alpha},\vec{\beta},h,h')}\downarrow & & \downarrow{\scriptstyle h'} \\ \sigma(\vec{\alpha}, \vec{\beta}, \omega, \omega') & \xrightarrow{\quad g' \quad} & \omega' \end{array} \tag{5}$$

commute.

Theorem 6.1. *Suppose $\vec{R}_+ :$ AdmRel$(\vec{\omega}_+, \vec{\omega}'_+)$ and $\vec{R}_- :$ AdmRel$(\vec{\omega}_-, \vec{\omega}'_-)$ are vectors of admissible relations, and*

$$\begin{array}{c} S_+ : \textit{AdmRel}(\tau(\vec{\omega}_-, \vec{\omega}_+), \tau(\vec{\omega}'_-, \vec{\omega}'_+)) \\ S_- : \textit{Rel}(\tau(\vec{\omega}_+, \vec{\omega}_-), \tau(\vec{\omega}'_+, \vec{\omega}'_-)) \end{array}$$

are relations. Then the following rule holds:

$$\frac{(i^{-1}, i^{-1}) \colon S_- \multimap \sigma(\vec{R}_+, \vec{R}_-, S_+, S_-) \qquad (i, i) \colon \sigma(\vec{R}_-, \vec{R}_+, S_-, S_+) \multimap S_+}{S_- \subset \tau(\vec{R}_+, \vec{R}_-) \qquad \tau(\vec{R}_-, \vec{R}_+) \subset S_+}$$

Proof. The proof proceeds as the proof of Theorem 5.1, and we start by considering the case where S_- is admissible. This time the terms generating h, h' have types

$$\begin{array}{rl} k \colon & \prod \vec{\alpha}, \vec{\beta}. \prod \omega, \omega'. (\sigma(\vec{\alpha}, \vec{\beta}, \omega, \omega') \multimap \omega') \multimap (\omega \multimap \sigma(\vec{\beta}, \vec{\alpha}, \omega', \omega)) \multimap \omega \multimap \tau(\vec{\beta}, \vec{\alpha}) \\ k' \colon & \prod \vec{\alpha}, \vec{\beta}. \prod \omega, \omega'. (\sigma(\vec{\alpha}, \vec{\beta}, \omega, \omega') \multimap \omega') \multimap (\omega \multimap \sigma(\vec{\beta}, \vec{\alpha}, \omega', \omega)) \multimap \tau(\vec{\alpha}, \vec{\beta}) \multimap \omega' \end{array}$$

Now, notice first that

$$k\, \vec{\omega}_+\, \vec{\omega}_-\, \tau(\vec{\omega}_+, \vec{\omega}_-)\, \tau(\vec{\omega}_-, \vec{\omega}_+)\, i\, i^{-1} = id_{\tau(\vec{\omega}_+, \vec{\omega}_-)} \tag{6}$$

$$k'\, \vec{\omega}_+\, \vec{\omega}_-\, \tau(\vec{\omega}_+, \vec{\omega}_-)\, \tau(\vec{\omega}_-, \vec{\omega}_+)\, i\, i^{-1} = id_{\tau(\vec{\omega}_-, \vec{\omega}_+)} \tag{7}$$

$$k\, \vec{\omega}'_+\, \vec{\omega}'_-\, \tau(\vec{\omega}'_+, \vec{\omega}'_-)\, \tau(\vec{\omega}'_-, \vec{\omega}'_+)\, i\, i^{-1} = id_{\tau(\vec{\omega}'_+, \vec{\omega}'_-)} \tag{8}$$

$$k'\, \vec{\omega}'_+\, \vec{\omega}'_-\, \tau(\vec{\omega}'_+, \vec{\omega}'_-)\, \tau(\vec{\omega}'_-, \vec{\omega}'_+)\, i\, i^{-1} = id_{\tau(\vec{\omega}'_-, \vec{\omega}'_+)} \tag{9}$$

because the identities make the diagrams (5) commute.

The theorem will follow from instantiating the parametricity schema of k, k' with $\vec{R}_-$ substituted for $\vec{\alpha}$, $\vec{R}_-$ substituted for $\vec{\beta}$ and S_- for ω and S_+ for ω'. This tells us that if

$$\begin{array}{c} (i^{-1}, i^{-1}) \colon S_- \multimap \sigma(\vec{R}_+, \vec{R}_-, S_+, S_-) \\ (i, i) \colon \sigma(\vec{R}_-, \vec{R}_+, S_-, S_+) \multimap S_+ \end{array}$$

then (using (6)-(9) above)

$$\begin{array}{c} (id_{\tau(\vec{\omega}_+, \vec{\omega}_-)}, id_{\tau(\vec{\omega}'_+, \vec{\omega}'_-)}) \colon S_- \multimap \tau(\vec{R}_+, \vec{R}_-) \\ (id_{\tau(\vec{\omega}_-, \vec{\omega}_+)}, id_{\tau(\vec{\omega}'_-, \vec{\omega}'_+)}) \colon \tau(\vec{R}_-, \vec{R}_+) \multimap S_+ \end{array}$$

which was what we needed to prove.

For the general case, dropping the assumtion that S_- is admissible, the proof proceeds exactly as in Theorem 5.1. □

References

[1] L. Birkedal, R. E. Møgelberg, and R. L. Petersen. Parametric domain-theoretic models of linear Abadi & Plotkin logic. 2005. Submitted. 2, 2, 3, 3, 4, 4, 4, 5, 5, 6

[2] P.J. Freyd. Algebraically complete categories. In A. Carboni, M. C. Pedicchio, and G. Rosolini, editors, *Category Theory. Proceedings, Como 1990*, volume 1488 of *Lecture Notes in Mathematics*, pages 95–104. Springer-Verlag, 1990. 5

[3] P.J. Freyd. Recursive types reduced to inductive types. In *Proceedings of the fifth IEEE Conference on Logic in Computer Science*, pages 498–507, 1990. 5

[4] P.J. Freyd. Remarks on algebraically compact categories. In M. P. Fourman, P.T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science. Proceedings of the LMS Symposium, Durham 1991*, volume 177 of *London Mathematical Society Lecture Note Series*, pages 95–106. Cambridge University Press, 1991. 5

[5] R. E. Møgelberg. *Categorical and domain theoretic models of parametric polymorphism*. PhD thesis, IT University of Copenhagen, 2005. Revised Aug. 2005.

[6] A.M. Pitts. Relational properties of domains. *Information and Computation*, 1995. To Appear. 2, 4, 5

[7] G. D. Plotkin. Type theory and recursion (extended abstract). In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, page 374, Montreal, Canada, 19–23 June 1993. IEEE Computer Society Press. 5

[8] G.D. Plotkin. Second order type theory and recursion. Notes for a talk at the Scott Fest, February 1993. 3, 4, 4, 5

[9] Gordon Plotkin and Martín Abadi. A logic for parametric polymorphism. In *Typed lambda calculi and applications (Utrecht, 1993)*, volume 664 of *Lecture Notes in Comput. Sci.*, pages 361–375. Springer, Berlin, 1993. 3.2

Chapter 4

Operational Semantics of LILY

In this chapter we include a student project by Carsten Varming extending the operational semantics of LILY with tensor and open types, deriving a number of properties of contextual equivalence.

Operational approach to Parametricity
Part II

Carsten Varming
cpr: 230980-2635

16th February 2006

Supervisor: Lars Birkedal

1 Preface

This is the second part of my project about parametricity. In includes the first part as it will be part of a technical report as a whole.

Some things have changed since the first part and other thing are new. All the stuff to do with the tensor product, unit and the interpreter is new. These additions are scattered around everywhere, but in particular the sections 5.3, 7 and 8 are new. Section 6 is the same as in part I. In the other sections I have tried to mark new parts with a * and changed parts with a †. Things without a mark are as in the first part except for the additions to handle tensor product and unit.

Quite a things are new in section 4 as I have changed some invariants used in the proofs of equivalence of the given semantics. Likewise many things have changed quite a bit as I have spent much time elaborating the proofs of the first part.

2 Introduction

In [BPR00] Bierman, Russo and Pitts gave an operational semantics of a functional language (Lily) with a polymorphic linear type system. With only four type constructs the type system is very simple, but it still provides us with powerful concepts such as linear use of variables and parametricity. The type system is strong enough to allow us to prove a strictness theorem, that in a sense unifies call-by-value and call-by-name semantics.

In this student project it is my intension to give a coherent and detailed examination of the theorems and definitions published in [BPR00]. I will start with a brief introduction to the static and dynamic semantics of the language. Then I will develop a structural operational semantics of Lily, as in [Pit02] to prove the strictness theorem. Along the way I will prove the ciu theorem for closed Lily-terms. I will also show an unwinding theorem for Lily.

Next I will look at four different relations, that turns out to be equivalent. One of these is contextual equivalence, and thus we end up with a valuable description of contextual equivalence. This part follows the outline of [Pit00] closely, but as the language is quite different, almost all the details are different.

In the end I will show how contextual equivalence can be used to define a category and I will give an example of how to encode natural numbers in Lily and prove that they behave as such. The idea behind the proof of the example is taken from [BMP05].

3 The language

The language without unit and tensor is given in [BPR00]. The language is given by the following grammar, type and evaluation relation.

Definition In Lily the raw terms and types are defined by:

Raw types	$\tau ::=$	α	type variables
		$\tau \multimap \tau'$	linear function type
		$\forall\alpha.\tau$	$\forall$-type
		$!\tau$	exponential type
		I	unit
		$\tau \otimes \tau$	tensor product
Raw terms	$M ::=$	a	linear variable
		x	intuitionistic variable
		$\lambda a : \tau.M$	abstraction
		$M_1 M_2$	application
		$\Lambda\alpha.M$	generalization
		$M\tau$	specialization
		$!(x = M : \tau)$	recursively defined thunk
		let $!x = M_1$ **in** M_2	exponential eliminator
		$*$	unit
		let $* = M_1$ **in** M_2	unit eliminator
		$M_1 \otimes M_2$	tensor
		let $a_1 \otimes a_1 \ = M_1$ **in** M_2	tensor eliminator

where α ranges over a denumerable infinite set of type variables, a ranges over a denumerable infinite set of linear program variables and x ranges over a denumerable infinite set of intuitionistic program variables.

In Lily $\forall\alpha.(\_)$, $\lambda a : \tau.(\_)$, $\Lambda\alpha.(\_)$, $!(x = (\_) : \tau)$, **let** $!x = M$ **in** $(\_)$ and **let** $a_1 \otimes a_2 = M_1$ **in** $(\_)$ are variable-binding constructs. I will denote the set of free type variables in τ with $ftv(\tau)$, free intuitionistic variables in M with $fiv(M)$ and free linear variables in M with $flv(M)$.

I will identify raw terms modulo renaming of bound variables. Likewise types are defined up to renaming of bound type variables. *

Given raw terms N, M and a linear variable a we define $M[N/a]$ as the result of capture avoiding substitution of N into all free occurrences of a in M. Likewise, given a intuitionistic variable x we define $M[N/x]$ as the result of capture avoiding substitution of N into all free occurrences of x in M. Given a type τ and a type variable α we define $M[\tau/\alpha]$ as the result of a capture avoiding substitution of τ into all free occurrences of α in M. *

Definition A type environment is a partial function with finite domain, from variables to types. Let Γ we a type environment. We then write $\Gamma, x : \tau$ for the type environment that maps x to τ and any variable y different from x to $\Gamma(y)$. We denote the empty type environment $\emptyset$. We abbreviate $ftv(\mathtt{im}(\Gamma))$ as $ftv(\Gamma)$. We say a type environment is linear (resp. intuitionistic) if it maps linear (resp. intuitionistic) variables to types.

We say a raw term M is well-typed and has type τ if and only if there is a set of type variables $\vec{\alpha}$, an intuitionistic Γ and linear Δ type environment such that $\Gamma, \Delta, \vec{\alpha}, \tau, M$ are in the relation

defined by the following rules.

$$\frac{ftv(\Gamma,\tau)\subseteq\vec{\alpha} \quad x\notin\mathrm{dom}(\Gamma)}{\Gamma,x:\tau;\emptyset\vdash_{\vec{\alpha}} x:\tau} \qquad \frac{ftv(\Gamma,\tau)\subseteq\vec{\alpha}}{\Gamma;a:\tau\vdash_{\vec{\alpha}} a:\tau}$$

$$\frac{\Gamma;\Delta,a:\tau\vdash_{\vec{\alpha}} M:\tau' \quad a\notin\mathrm{dom}(\Delta)}{\Gamma;\Delta\vdash_{\vec{\alpha}}\lambda a:\tau.M:\tau\multimap\tau'}$$

$$\frac{\Gamma;\Delta_1\vdash_{\vec{\alpha}} M_1:\tau\multimap\tau' \quad \Gamma;\Delta_2\vdash_{\vec{\alpha}} M_2:\tau \quad \mathrm{dom}(\Delta_1)\cap\mathrm{dom}(\Delta_2)=\emptyset}{\Gamma;\Delta_1,\Delta_2\vdash_{\vec{\alpha}} M_1M_2:\tau'}$$

$$\frac{\Gamma;\Delta\vdash_{\vec{\alpha},\alpha} M:\tau \quad \alpha\notin\vec{\alpha}\cup ftv(\Gamma,\Delta)}{\Gamma;\Delta\vdash_{\vec{\alpha}}\Lambda\alpha.M:\forall\alpha.\tau} \qquad \frac{\Gamma;\Delta\vdash_{\vec{\alpha}} M:\forall\alpha.\tau \quad ftv(\tau')\subseteq\vec{\alpha}}{\Gamma;\Delta\vdash_{\vec{\alpha}} M\tau':\tau[\tau'/\alpha]}$$

$$\frac{\Gamma,x:\tau;\emptyset\vdash_{\vec{\alpha}} M:\tau \quad x\notin\mathrm{dom}(\Gamma)}{\Gamma;\emptyset\vdash_{\vec{\alpha}}!(x=M:\tau):!\tau}$$

$$\frac{\Gamma;\Delta_1\vdash_{\vec{\alpha}} M_1:!\tau \quad \Gamma,x:\tau;\Delta_2\vdash_{\vec{\alpha}} M_2:\tau' \quad x\notin\mathrm{dom}(\Gamma) \quad \mathrm{dom}(\Delta_1)\cap\mathrm{dom}(\Delta_2)=\emptyset}{\Gamma;\Delta_1,\Delta_2\vdash_{\vec{\alpha}}\mathbf{let}\ !x=M_1\ \mathbf{in}\ M_2:\tau'}$$

$$\frac{}{\Gamma;\emptyset\vdash_{\vec{\alpha}} *:I}$$

$$\frac{\Gamma;\Delta_1\vdash_{\vec{\alpha}} M_1:I \quad \Gamma;\Delta_2\vdash_{\vec{\alpha}} M_2:\tau \quad \mathrm{dom}(\Delta_1)\cap\mathrm{dom}(\Delta_2)=\emptyset}{\Gamma;\Delta_1,\Delta_2\vdash_{\vec{\alpha}}\mathbf{let}\ *=M_1\ \mathbf{in}\ M_2:\tau}$$

$$\frac{\Gamma;\Delta_1\vdash_{\vec{\alpha}} M_1:\tau_1 \quad \Gamma;\Delta_2\vdash_{\vec{\alpha}} M_2:\tau_2 \quad \mathrm{dom}(\Delta_1)\cap\mathrm{dom}(\Delta_2)=\emptyset}{\Gamma;\Delta_1,\Delta_2\vdash_{\vec{\alpha}} M_1\otimes M_2:\tau_1\otimes\tau_2}$$

$$\frac{\begin{array}{c}\Gamma;\Delta_1\vdash_{\vec{\alpha}} M_1:\tau_1\otimes\tau_2 \qquad \Gamma;\Delta_2,a_1:\tau_1,a_2:\tau_2\vdash_{\vec{\alpha}} M_2:\sigma\\ \mathrm{dom}(\Delta_1)\cap\mathrm{dom}(\Delta_2)=\emptyset \quad a_1,a_2\notin\mathrm{dom}(\Delta_1)\cup\mathrm{dom}(\Delta_2) \quad a_1\neq a_2\end{array}}{\Gamma;\Delta_1,\Delta_2\vdash_{\vec{\alpha}}\mathbf{let}\ a_1\otimes a_2=M_1\ \mathbf{in}\ M_2:\sigma}$$

Note how the rules ensure linearity of Δ, by disallowing weakening and contraction.

In what is to come we will only consider raw terms that are well-typed. Thus terms are defined as well-typed raw terms.

Finally, we have two dynamic semantics. One gives a call-by-value semantics ($\Downarrow_s$) and one gives a call-by-name semantics ($\Downarrow_n$).

Definition Let $\Downarrow$ denote either $\Downarrow_s$ or $\Downarrow_n$. We then define $\Downarrow_s$ and $\Downarrow_n$ as *evaluation* relations given by:

$$\mathrm{v}_1:\frac{}{\lambda a:\tau.M\Downarrow\lambda a:\tau.M} \qquad \mathrm{v}_2:\frac{}{\Lambda\alpha.M\Downarrow\Lambda\alpha.M} \qquad \mathrm{v}_3:\frac{}{!(x=M:\tau)\Downarrow!(x=M:\tau)}$$

$$\mathrm{v}_4:\frac{}{*\Downarrow *} \qquad \mathrm{v}_5:\frac{}{M_1\otimes M_2\Downarrow M_1\otimes M_2}$$

$$\mathrm{app}_t:\frac{M\Downarrow\Lambda\alpha.M' \quad M'[\tau/\alpha]\Downarrow V}{M\tau\Downarrow V}$$

$$\mathrm{rec}:\frac{M_1\Downarrow!(x=M:\tau) \quad M_2[(\mathbf{let}\ !x=!(x=M:\tau)\ \mathbf{in}\ M)/y]\Downarrow V}{\mathbf{let}\ !y=M_1\ \mathbf{in}\ M_2\Downarrow V}$$

$$\mathrm{c}_{val}:\frac{M_1\Downarrow_s\lambda a:\tau.M \quad M_2\Downarrow_s V' \quad M[V'/a]\Downarrow_s V}{M_1M_2\Downarrow_s V}$$

$$\mathrm{c}_{name}:\frac{M_1\Downarrow_n\lambda a:\tau.M \quad M[M_2/a]\Downarrow_n V}{M_1M_2\Downarrow_n V}$$

$$\mathrm{unit}:\frac{M_1\Downarrow * \quad M_2\Downarrow V}{\mathbf{let}\ *=M_1\ \mathbf{in}\ M_2\Downarrow V}$$

$$\mathrm{tensor}:\frac{M_1\Downarrow N_1\otimes N_2 \quad M_2[N_1,N_2/a_1,a_2]\Downarrow V}{\mathbf{let}\ a_1\otimes a_2=M_1\ \mathbf{in}\ M_2\Downarrow V}$$

We say a term M evaluates to V in a call-by-value (resp. call-by-name) semantics if and only if $M\Downarrow_s V$ (resp. $M\Downarrow_n V$).

The set of values of type τ, denoted $Val(\tau)$ is a subset of the set of terms of type τ such that $V \in Val(\tau)$ if and only if $V = \lambda a : \tau'.M$, $V = \Lambda\alpha.M$, $V = !(x = M : \tau')$, $V = *$ or $V = M_1 \otimes M_2$ for some type variable α, type τ', intuitionistic variable x and terms M, M_1, M_2. I will use M for arbitrary terms and V for values.

I will abbreviate **let** $!x = !(x = M : \tau)$ **in** M to **fix** $x : \tau.M$ as the evaluation of this term is defined recursively and behaves like a fix point (have a look at the unwinding theorem [theorem 4.28]). I will also abbreviate the following non-recursive thunk $!(x = M : \tau)$, $x \notin fiv(M)$ to $!M$.

Lemma 3.1 (Rule inversion of the type system) *For all type environments Γ, Δ, set of type variables $\vec{\alpha}$, terms M and types τ, if $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$ then one and only one rule must have been applied as the last rule in the derivation of $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$.* *

Proof To any syntactic construct in Lily, only one rule can result in a derivation with that syntax. ∎

Lemma 3.2 *For all type environments Δ, Γ, terms M and types τ_1, τ_2.* *

$$\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau_1 \quad \wedge \quad \Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau_2 \quad \Rightarrow \quad \tau_1 = \tau_2$$

Proof By induction on $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau_1$. ∎

Lemma 3.3 (Rule inversion of the dynamic semantics) *For all closed terms M and values V, if $M \Downarrow V$ then one and only one rule must have been applied as the last rule in the derivation of $M \Downarrow V$.* *

Proof To any syntactic construct of a closed term only one rule can result in a derivation of that syntax. Note how no rules are given for variables as they are never closed. ∎

Lemma 3.4 *For all closed terms M and values V,* *

$$M \Downarrow V \wedge M \Downarrow V' \Rightarrow V = V'.$$

Proof By induction on $M \Downarrow V$. ∎

Lemma 3.5 *If $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$ and $\alpha \notin \vec{\alpha}$ then $\Gamma; \Delta \vdash_{\vec{\alpha},\alpha} M : \tau$.* *

Proof By induction on $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$. ∎

Lemma 3.6 *If $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$, $x \notin \mathrm{dom}(\Gamma)$, $ftv(\tau') \subseteq \vec{\alpha}$ then $\Gamma, x : \tau'; \Delta \vdash_{\vec{\alpha}} M : \tau$.* *

Proof By induction on $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$. ∎

Lemma 3.7 *For all types τ, terms M, type environments Γ, Δ and set of type variable $\vec{\alpha}$,* *

$$\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau \Rightarrow ftv(\Gamma, \Delta, \tau) \subseteq \vec{\alpha}.$$

Proof By induction on $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$. ∎

Lemma 3.8 (Substitution lemma for types) *Given types τ, τ' such that $\alpha \in ftv(\tau)$ then $\tau[\tau'/\alpha]$ is a type such that $ftv(\tau[\tau'/\alpha]) = (ftv(\tau) \backslash \{\alpha\}) \cup ftv(\tau')$.* *

Proof By structural induction on τ, using the fact that types are identified up to bound variables in the $\forall$ case. ∎

Lemma 3.9 (Substitution lemma) *For all type environments Γ, Δ, set of type variables $\vec{\alpha}$, terms M, N and types τ, τ'.* *

$$\frac{\Gamma; \Delta \vdash_{\vec{\alpha},\alpha} M : \tau \quad \alpha \notin \vec{\alpha} \quad ftv(\tau') \subseteq \vec{\alpha}}{\Gamma[\tau'/\alpha]; \Delta[\tau'/\alpha] \vdash_{\vec{\alpha}} M[\tau'/\alpha] : \tau[\tau'/\alpha]}$$

$$\frac{\Gamma, x : \tau; \Delta \vdash_{\vec{\alpha}} M : \tau' \quad \Gamma; \emptyset \vdash_{\vec{\alpha}} N : \tau \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma; \Delta \vdash_{\vec{\alpha}} M[N/x] : \tau'}$$

$$\frac{\Gamma; \Delta_1, a : \tau \vdash_{\vec{\alpha}} M : \tau' \quad \Gamma; \Delta_2 \vdash_{\vec{\alpha}} N : \tau \quad a \notin \mathrm{dom}(\Delta_1) \quad \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset}{\Gamma; \Delta_1, \Delta_2 \vdash_{\vec{\alpha}} M[N/a] : \tau'.}$$

Proof By induction over the derivation of $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau'$. Let begin with substitutivity of types.

If $M = x$ then $\Gamma, x : \tau; \emptyset \vdash_{\vec{\alpha},\alpha} x : \tau$, $\alpha \notin \vec{\alpha}$, $ftv(\tau') \subseteq \vec{\alpha}$, $ftv(\Gamma, \tau) \subseteq \vec{\alpha}, \alpha$ and $x \notin \mathrm{dom}(\Gamma)$, thus $ftv(\Gamma[\tau'/\alpha], \tau[\tau'/\alpha]) \subseteq \vec{\alpha}$, $x \notin \mathrm{dom}(\Gamma[\tau'/\alpha])$, whence $(\Gamma, x : \tau)[\tau'/\alpha]; \emptyset[\tau'/\alpha] \vdash_{\vec{\alpha}} x[\tau'/\alpha] : \tau[\tau'/\alpha]$.

If $M = a$ then $\Gamma, a : \tau \vdash_{\vec{\alpha},\alpha} a : \tau$, $\alpha \notin \vec{\alpha}$, $ftv(\tau') \subseteq \vec{\alpha}$, $ftv(\Gamma, \tau) \subseteq \vec{\alpha}, \alpha$, thus $ftv(\Gamma[\tau'/\alpha], \tau[\tau'/\alpha]) \subseteq \vec{\alpha}$, whence $\Gamma[\tau'/\alpha], a : \tau[\tau'/\alpha] \vdash_{\vec{\alpha}} a : \tau[\tau'/\alpha]$.

If $M = \lambda a : \sigma.M'$ then $\Gamma; \Delta \vdash_{\vec{\alpha},\alpha} \lambda a : \sigma.M' : \sigma \multimap \sigma'$, thus $\Gamma; \Delta, a : \sigma \vdash_{\vec{\alpha},\alpha} M' : \sigma'$, $a \notin \mathrm{dom}(\Delta)$, $\alpha \notin \vec{\alpha}$, $ftv(\tau') \subseteq \vec{\alpha}$. By induction $\Gamma[\tau'/\alpha], \Delta[\tau'/\alpha], a : \sigma[\tau'/\alpha] \vdash_{\vec{\alpha}} M'[\tau'/\alpha] : \sigma'[\tau'/\alpha]$, whence $\Gamma[\tau'/\alpha], \Delta[\tau'/\alpha] \vdash_{\vec{\alpha}} (\lambda a : \sigma.M')[\tau'/\alpha] : \sigma'[\tau'/\alpha]$.

If $M = N\sigma$ then $\Gamma; \Delta \vdash_{\vec{\alpha},\alpha} N\sigma : \tau[\sigma/\beta]$, where $\Gamma; \Delta \vdash_{\vec{\alpha},\alpha} N : \forall\beta.\tau$, $ftv(\sigma) \subseteq \vec{\alpha}, \alpha$, $\alpha \notin \vec{\alpha}$ and $ftv(\tau') \subseteq \vec{\alpha}$, thus $ftv(\sigma[\tau'/\alpha]) \subseteq \vec{\alpha}$). By induction $\Gamma[\tau'/\alpha]; \Delta[\tau'/\alpha] \vdash_{\vec{\alpha}} N[\tau'/\alpha] : (\forall\beta.\tau)[\tau'/\alpha]$. As types are identified modulo bound variables β can be made different from α in τ. Then $(\forall\beta.\tau)[\tau'/\alpha] = \forall\beta.\tau[\tau'/\alpha]$ and $(N\sigma)[\tau'/\alpha] = N[\tau'/\alpha]\sigma[\tau'/\alpha]$, thus

$$\Gamma[\tau'/\alpha]; \Delta[\tau'/\alpha] \vdash_{\vec{\alpha}} (N\sigma)[\tau'/\alpha] : (\tau[\tau'/\alpha])[\sigma[\tau'/\alpha]/\beta].$$

As β is different from α, $(\tau[\tau'/\alpha])[\sigma[\tau'/\alpha]/\beta] = (\tau[\sigma/\beta])[\tau'/\alpha]$.

If $M = \Lambda\beta.N$ then $\Gamma; \Delta \vdash_{\vec{\alpha},\alpha} \Lambda\beta.N : \forall\beta.\tau$, $\alpha \notin \vec{\alpha}$, $ftv(\tau') \subseteq \vec{\alpha}$, thus $\Gamma; \Delta \vdash_{\vec{\alpha},\alpha,\beta} N : \tau$ and $\beta \notin \vec{\alpha} \cup \{\alpha\} \cup ftv(\Gamma, \Delta)$, whence $\alpha \neq \beta$, $\beta \notin \vec{\alpha} \cup ftv(\Gamma[\tau'/\alpha], \Delta[\tau'/\alpha])$, $\alpha \notin \vec{\alpha} \cup \{\beta\}$ and $ftv(\tau') \subseteq \vec{\alpha} \cup \{\beta\}$. By induction $\Gamma[\tau'/\alpha]; \Delta[\tau'/\alpha] \vdash_{\vec{\alpha},\beta} N[\tau'/\alpha] : \tau[\tau'/\alpha]$. Hence $\Gamma[\tau'/\alpha]; \Delta[\tau'/\alpha] \vdash_{\vec{\alpha}} (\Lambda\beta.N)[\tau'/\alpha] : (\forall\beta.\tau)[\tau'/\alpha]$

The other cases goes through with the same reasoning.

Now to substitutivity of intuitionistic variables.

If $M = x$ then $\Gamma, x : \tau; \emptyset \vdash_{\vec{\alpha}} x : \tau$, $\Gamma; \emptyset \vdash_{\vec{\alpha}} N : \tau$, $x \notin \mathrm{dom}(\Gamma)$. Now $x[N/x] = N$, thus $\Gamma; \emptyset \vdash_{\vec{\alpha}} x[N/x] : \tau$. If $M = y \neq x$ then as $y[N/x] = y$ and $y \in \mathrm{dom}(\Gamma)$, $\Gamma(y) = \tau$, $\Gamma; \emptyset \vdash_{\vec{\alpha}} y]N/x] : \tau$.

If $M = a$ then $x \neq a$. As before $\Gamma; \Delta \vdash_{\vec{\alpha}} a[N/x] : \tau'$.

If $M = \mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2$ then $\Gamma, x : \tau; \Delta_1, \Delta_2 \vdash_{\vec{\alpha}} \mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2 : \tau'$ and $x \notin \mathrm{dom}(\Gamma)$, thus $\Gamma, x : \tau; \Delta_1 \vdash_{\vec{\alpha}} M_1 :!\sigma$, $\Gamma, x : \tau, y : \sigma; \Delta_2 \vdash_{\vec{\alpha}} M_2 : \tau'$, $\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset$ and $y \notin \mathrm{dom}(\Gamma) \cup \{x\}$, thus $x \neq y$. By induction $\Gamma; \Delta_1 \vdash_{\vec{\alpha}} M_1[N/x] :!\sigma$ and $\Gamma, y : \sigma; \Delta_2 \vdash_{\vec{\alpha}} M_2[N/x] : \tau'$. Hence $\Gamma; \Delta \vdash_{\vec{\alpha}} (M_1 M_2)[N/x] : \tau'$.

If $M =!(y = M' : \sigma)$ then $\tau' =!\sigma$ and $\Gamma, x : \tau; \emptyset \vdash_{\vec{\alpha}} !(y = M' : \sigma) :!\sigma$, thus $\Gamma, x : \tau, y : \sigma; \emptyset \vdash_{\vec{\alpha}} M' : \sigma$ and $y \notin \mathrm{dom}(\Gamma) \cup \{x\}$, thus $x \neq y$. By induction $\Gamma, y : \sigma, \emptyset \vdash_{\vec{\alpha}} M'[N/x] : \sigma$, thus $\Gamma; \emptyset \vdash_{\vec{\alpha}} !(y = M' : \sigma)[N/x] : \tau'$.

If $M = M_1 M_2$ then $\Gamma, x : \tau; \Delta_1, \Delta_2 \vdash_{\vec{\alpha}} M_1 M_2 : \sigma'$ and $\Gamma; \emptyset \vdash_{\vec{\alpha}} N : \tau$ and $x \notin \mathrm{dom}(\Gamma)$, thus $\Gamma, x : \tau; \Delta_1 \vdash_{\vec{\alpha}} M_1 : \sigma \multimap \sigma'$, $\Gamma, x : \tau; \Delta_2 \vdash_{\vec{\alpha}} M_2 : \sigma$, $\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset$. By induction $\Gamma; \Delta_1 \vdash_{\vec{\alpha}} M_1[N/x] : \sigma \multimap \sigma$, $\Gamma; \Delta_2 \vdash_{\vec{\alpha}} M_2[N/x] : \sigma$ and $\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset$, thus as $(M_1 M_2)[N/x] = M_1[N/x] M_2[N/x]$, $\Gamma; \Delta_1, \Delta_2 \vdash_{\vec{\alpha}} (M_1 M_2)[N/x] : \sigma'$.

The other cases are like the ones above.

The linear case is proved using the same reasoning as above. ∎

Lemma 3.10 *For all closed terms M such that $\emptyset;\emptyset \vdash_\emptyset M : \tau$,* *

$$M \Downarrow V \Rightarrow \emptyset;\emptyset \vdash_\emptyset V : \tau \wedge V \in Val(\tau).$$

Proof By induction on $M \Downarrow V$, using rule inversion of the type system and the substitution lemma. ∎

In [BPR00] we have the following definition of contextual equivalence.

Definition (Ground contextual equivalence of closed terms) Let $Typ = \{\tau | ftv(\tau) = \emptyset\}$ be the set of *closed* types. Given $\tau \in Typ$, let $Term(\tau)$ be the set $\{M \mid \emptyset;\emptyset \vdash_\emptyset M : \tau\}$ of *closed* terms of type τ. Given $M, M' \in Term(\tau)$, we have $M =_{\text{gnd}} M' : \tau$ if and only if

$$\begin{array}{lcl} \forall x, N, \tau' & : & x : \tau;\emptyset \vdash_\emptyset N : \tau' \Rightarrow (N[M/x] \Downarrow_n \Leftrightarrow N[M'/x] \Downarrow_n) \\ \forall a, N, \tau' & : & \emptyset; a : \tau \vdash_\emptyset N : \tau' \Rightarrow (N[M/a] \Downarrow_n \Leftrightarrow N[M'/a] \Downarrow_n) \end{array}$$

4 An abstract machine for Lily

In order to prove the strictness theorem I follow the outline given in [BPR00] and [Pit02]. This implies defining a structural operational semantics for Lily and showing that it is equivalent to the semantics given above.

Definition An *evaluation frame* $\mathcal{F}$ is a term with a *hole* $-$, given by the grammar

$$\begin{array}{rcl} \mathcal{F} & ::= & \mathbf{let}\ !y = - \ \mathbf{in}\ M \\ & | & -M \\ & | & -\tau \\ & | & \mathbf{let}\ * = - \ \mathbf{in}\ M \\ & | & \mathbf{let}\ a_1 \otimes a_2 = - \ \mathbf{in}\ M \end{array}$$

where M is a term and τ is a type.

We write $\mathcal{F}[M]$ for the term given by replacing the hole in $\mathcal{F}$ by the term M. We can compose evaluation frames into a *frame stack* $\mathcal{F}s$, and we define

$$\begin{array}{rcll} \mathcal{F}s & ::= & \mathcal{I}d & \text{Empty} \\ & | & \mathcal{F}s \circ \mathcal{F} & \text{non-empty} \end{array}$$

and $\mathcal{I}d[M] = M$ and $(\mathcal{F}s \circ \mathcal{F})[M] = \mathcal{F}s[\mathcal{F}[M]]$.

A *closed* evaluation frame is an evaluation frame $\mathcal{F}$, such that $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}[a] : \tau'$ for some closed types τ and τ'. A *closed* frame stack is a frame stack $\mathcal{F}s$ where all evaluation frames are closed. By definition $\mathcal{I}d$ is closed.

We define a typing relation on closed frame stacks, by †

$$\frac{}{\mathcal{I}d : \tau \multimap \tau} \qquad \frac{\emptyset; a : \tau \vdash_\emptyset \mathcal{F}[a] : \tau'' \quad \mathcal{F}s : \tau'' \multimap \tau'}{\mathcal{F}s \circ \mathcal{F} : \tau \multimap \tau'}$$

I will only consider closed frame stacks, unless stated otherwise.

Lemma 4.1 *Given a closed frame stack $\mathcal{F}s$ of type $\tau \multimap \tau'$ and a term M such that $\Gamma;\Delta \vdash_{\vec{\alpha}} M : \tau$ then $\Gamma;\Delta \vdash_{\vec{\alpha}} \mathcal{F}s[M] : \tau'$.* †

Proof By induction on $\mathcal{F}s$.

If $\mathcal{F}s = \mathcal{I}d : \tau \multimap \tau$ then $\mathcal{F}s[M] = M$, thus by assumption $\Gamma;\Delta \vdash_{\vec{\alpha}} \mathcal{F}s[M] : \tau'$.

If $\mathcal{F}s = \mathcal{F}s' \circ \mathcal{F}$ then $\mathcal{F}s' : \tau'' \multimap \tau'$ and $\emptyset; a : \tau \vdash_{\emptyset} \mathcal{F}[a] : \tau''$. By Lemma 3.6 and Lemma 3.5, $\Gamma; a : \tau \vdash_{\vec{\alpha}} \mathcal{F}[a] : \tau''$, thus by the substitution lemma $\Gamma;\Delta \vdash_{\vec{\alpha}} \mathcal{F}[M] : \tau''$ and by induction $\Gamma;\Delta \vdash_{\vec{\alpha}} \mathcal{F}s'[\mathcal{F}[M]] : \tau'$, whence $\Gamma;\Delta \vdash_{\vec{\alpha}} \mathcal{F}s[M] : \tau'$. ∎

Now we can define an evaluation relation using frame stacks, in which a term evaluate relative to a given frame stack. Please note how non-values reduce by pushing an evaluation frame on the frame stack and how only values can pop an evaluation frame from the frame stack. This is the key to Lemma 4.4.

Definition Let $\rightarrow$ be the relation in $\{\langle \mathcal{F}s, M\rangle \mid \mathcal{F}s : \tau \multimap \tau', M : \tau\}^2$ where $\mathcal{F}s$ ranges of closed frame stacks and M ranges over closed terms, given by:

$$\begin{gathered}
\langle \mathcal{F}s, \mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2\rangle \rightarrow \langle \mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, M_1\rangle \\
\langle \mathcal{F}s, M_1 M_2\rangle \rightarrow \langle \mathcal{F}s \circ -M_2, M_1\rangle \\
\langle \mathcal{F}s, M\tau\rangle \rightarrow \langle \mathcal{F}s \circ -\tau, M\rangle \\
\langle \mathcal{F}s, \mathbf{let}\ * = M_1\ \mathbf{in}\ M_2\rangle \rightarrow \langle \mathcal{F}s \circ \mathbf{let}\ * = -\ \mathbf{in}\ M_2, M_1\rangle \\
\langle \mathcal{F}s, \mathbf{let}\ a_1 \otimes a_2 = M_1\ \mathbf{in}\ M_2\rangle \rightarrow \langle \mathcal{F}s \circ \mathbf{let}\ a_1 \otimes a_2 = -\ \mathbf{in}\ M_2, M_1\rangle \\
\langle \mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, !(x = M : \tau)\rangle \rightarrow \langle \mathcal{F}s, M_2[(\mathbf{let}\ !x = !(x = M : \tau)\ \mathbf{in}\ M)/y]\rangle \\
\langle \mathcal{F}s \circ -M_2, \lambda a : \tau.M\rangle \rightarrow \langle \mathcal{F}s, M[M_2/a]\rangle \\
\langle \mathcal{F}s \circ -\tau, \Lambda\alpha.M\rangle \rightarrow \langle \mathcal{F}s, M[\tau/\alpha]\rangle \\
\langle \mathcal{F}s \circ \mathbf{let}\ * = -\ \mathbf{in}\ M, *\rangle \rightarrow \langle \mathcal{F}s, M\rangle \\
\langle \mathcal{F}s \circ \mathbf{let}\ a_1 \otimes a_2 = -\ \mathbf{in}\ M, N_1 \otimes N_2\rangle \rightarrow \langle \mathcal{F}s, M[N_1, N_2/a_1, a_2]\rangle
\end{gathered}$$

In the following I will make use of the lemma below without further notice, the proof is omitted since it is a straightforward exercise.

Lemma 4.2 *The semantics is deterministic: For all closed frame stacks $\mathcal{F}s : \tau \multimap \tau'$ and terms $M \in Term(\tau)$, if $\langle \mathcal{F}s, M\rangle \rightarrow \langle \mathcal{F}s', M'\rangle$ and $\langle \mathcal{F}s, M\rangle \rightarrow \langle \mathcal{F}s'', M''\rangle$ then $\mathcal{F}s' = \mathcal{F}s''$ and $M' = M''$.*

Proof By inspection. ∎

Lemma 4.3 *Given a closed frame stack $\mathcal{F}s : \tau \multimap \tau'$ and a closed term $M : \tau$ such that $\langle \mathcal{F}s, M\rangle \rightarrow \langle \mathcal{F}s', M'\rangle$, then $\mathcal{F}s' : \tau'' \multimap \tau'$ is closed and $M' : \tau''$ is closed for some type τ''.* *

Proof The proof is a case analysis of the derivation of $\langle \mathcal{F}s, M\rangle \rightarrow \langle \mathcal{F}s', M'\rangle$.

Assume we have the derivation

$$\langle \mathcal{F}s, \mathbf{let}\ a_1 \otimes a_2 = M_1\ \mathbf{in}\ M_2\rangle \rightarrow \langle \mathcal{F}s \circ \mathbf{let}\ a_1 \otimes a_2 = -\ \mathbf{in}\ M_2, M_1\rangle.$$

By assumption $\emptyset;\emptyset \vdash_{\emptyset} \mathbf{let}\ a_1 \otimes a_2 = M_1\ \mathbf{in}\ M_2 : \tau$, thus $\emptyset;\emptyset \vdash_{\emptyset} M_1 : \sigma \otimes \sigma'$ and $\emptyset; a_1 : \sigma, a_2 : \sigma' \vdash_{\emptyset} M_2 : \tau$. Thus M_1 is closed. As $\emptyset; a : \sigma \otimes \sigma' \vdash_{\emptyset} a : \sigma \otimes \sigma'$, $\emptyset; a : \sigma \otimes \sigma' \vdash_{\emptyset} \mathbf{let}\ a_1 \otimes a_2 = a\ \mathbf{in}\ M_2 : \tau$. Thus $\mathbf{let}\ a_1 \otimes a_2 = -\ \mathbf{in}\ M_2$ is a closed evaluation frame. Hence $\mathcal{F}s \circ \mathbf{let}\ a_1 \otimes a_2 = -\ \mathbf{in}\ M_2$ is a closed frame stack of type $\sigma \otimes \sigma' \multimap \tau'$ and M_1 is closed of type $\sigma \otimes \sigma'$.

Assume we have the derivation

$$\langle \mathcal{F}s' \circ \mathbf{let}\ a_1 \otimes a_2 = - \ \mathbf{in}\ M_2, N_1 \otimes N_2 \rangle \to \langle \mathcal{F}s', M_2[N_1, N_2/a_1, a_2] \rangle.$$

$\mathcal{F}s'$ is closed of type $\tau \multimap \tau''$ by assumption. As $\mathbf{let}\ a_1 \otimes a_2 = - \ \mathbf{in}\ M_2$ is closed, $\tau = \sigma \otimes \sigma'$ for some closed types σ, σ', $N_1 : \sigma$, $N_2 : \sigma'$ and $\emptyset; a_1 : \sigma_1, a_2 : \sigma' \vdash_\emptyset M_2 : \tau''$. By the substitution lemma $\emptyset; a_2 : \sigma' \vdash_\emptyset M_2[N_1/a_1] : \tau''$ and $\emptyset; \emptyset \vdash_\emptyset M_2[N_1, N_2/a_1, a_2]$ is closed of type τ''.

The other cases are handled the same way. ∎

The next lemma is in the heart of evaluation using frame stacks, since it points out the idea of evaluating terms in frames.

Lemma 4.4 *For all closed frame stacks $\mathcal{F}s : \tau \multimap \tau'$ and closed terms $M : \tau$,*

$$\langle \mathcal{I}d, \mathcal{F}s[M] \rangle \to^* \langle \mathcal{F}s, M \rangle,$$

where $\to^*$ is the transitive *and* reflexive *closure of $\to$.*

Proof By induction on the length n of the frame stack $\mathcal{F}s$.

$n = 0$: Given a closed frame stack $\mathcal{F}s$ it must be $\mathcal{I}d$ thus $\langle \mathcal{I}d, \mathcal{F}s[M] \rangle = \langle \mathcal{I}d, M \rangle \to^* \langle \mathcal{I}d, M \rangle = \langle \mathcal{F}s, M \rangle$ as $\to^*$ is reflexive.

$n > 0$: Given a closed frame stack $\mathcal{F}s$ is must be $\mathcal{F}s' \circ \mathcal{F}$ for some closed frame stack $\mathcal{F}s' : \tau'' \multimap \tau'$ and closed evaluation frame $\mathcal{F}$, such that $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}[a] : \tau''$. By the substitution lemma $\mathcal{F}[M] : \tau''$, thus by induction

$$\langle \mathcal{I}d, \mathcal{F}s[M] \rangle = \langle \mathcal{I}d, \mathcal{F}s'[\mathcal{F}[M]] \rangle \to^* \langle \mathcal{F}s', \mathcal{F}[M] \rangle \xrightarrow{\star} \langle \mathcal{F}s' \circ \mathcal{F}, M \rangle = \langle \mathcal{F}s, M \rangle.$$

We get $\star$ by case analysis. ∎

Lemma 4.5 *For all closed evaluation frames $\mathcal{F}$, and closed terms $M : \tau$, such that $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}[a] : \tau'$ and values $V : \tau$,* *

$$\mathcal{F}[M] \Downarrow_n V \Leftrightarrow \exists V'.M \Downarrow_n V' \wedge \mathcal{F}[V'] \Downarrow_n V.$$

Proof The proof is by case analysis of $\mathcal{F}$.

If $\mathcal{F} = -M_2$ then $\mathcal{F}[M] = MM_2$. Assume $MM_2 \Downarrow_n V$, then $M \Downarrow_n \lambda a : \sigma.M'$ and $M'[M_2/a] \Downarrow_n V$ for some M', a, σ. As $\lambda a : \sigma.M' \Downarrow_n \lambda a : \sigma.M'$, $(\lambda a : \sigma.M')M_2 \Downarrow_n V$.

Assume there exists V' such that $M \Downarrow_n V' \wedge V'M_2 \Downarrow_n V$. By the substitution lemma $MM_2 : \tau'$, thus by type preservation $V' = \lambda a : \sigma.M'$ for some a, σ, M'. From $(\lambda a : \sigma.M')M_2 \Downarrow_n V$, $M'[M_2/a] \Downarrow_n V$ and as $M \Downarrow_n \lambda a : \sigma.M'$ we have $MM_2 \Downarrow_n V$.

The other cases are handled in the same manner. ∎

Lemma 4.6 *For all closed frame stacks $\mathcal{F}s : \tau \multimap \tau'$ and closed terms $M : \tau$ and values $V : \tau'$,* *

$$\mathcal{F}s[M] \Downarrow_n V \Leftrightarrow \exists V'.M \Downarrow_n V' \wedge \mathcal{F}s[V'] \Downarrow_n V.$$

Proof The proof is by induction on length n of $\mathcal{F}s$.

If $n = 0$ then $\mathcal{F}s = \mathcal{I}d$, thus $\Rightarrow$ is trivial and as $V' \Downarrow_n V \Rightarrow V' = V$ we have $\Leftarrow$.

If $n > 0$ then $\mathcal{F}s = \mathcal{F}s' \circ \mathcal{F}$ for some closed frame stack $\mathcal{F}s' : \tau'' \multimap \tau'$ and closed evaluation frame $\mathcal{F}$, such that $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}[a] : \tau''$. By the substitution lemma $\mathcal{F}[M] : \tau''$. By induction,

$$\mathcal{F}s'[\mathcal{F}[M]] \Downarrow_n V \Leftrightarrow \exists V'.\mathcal{F}[M] \Downarrow_n V' \wedge \mathcal{F}s'[V'] \Downarrow_n V$$

Assume $\mathcal{F}s[M] \Downarrow_n V$ then $\mathcal{F}s'[\mathcal{F}[M]] \Downarrow_n V$, thus $\mathcal{F}[M] \Downarrow_n V' \wedge \mathcal{F}s'[V'] \Downarrow_n V$ for some V'. As $\mathcal{F}$ and M are closed, by Lemma 4.5 $M \Downarrow_n V''$ and $\mathcal{F}[V''] \Downarrow_n V'$ for some value V''. As the length of $\mathcal{F}s'$ is less than the length of $\mathcal{F}s$ we get by induction $\mathcal{F}s'[\mathcal{F}[V'']] \Downarrow_n V$, thus $M \Downarrow_n V'' \wedge \mathcal{F}s[V''] \Downarrow_n V$ for some V''.

Assume $M \Downarrow_n V'' \wedge \mathcal{F}s[V''] \Downarrow_n V$ for some V''. Then $\mathcal{F}s'[\mathcal{F}[V'']] \Downarrow_n V$ thus by induction $\mathcal{F}[V''] \Downarrow_n V'''$ and $\mathcal{F}s'[V'''] \Downarrow_n V$ for some V'''. By Lemma 4.5 $\mathcal{F}[M] \Downarrow_n V'''$. Hence $\exists V''.\mathcal{F}[M] \Downarrow_n V'' \wedge \mathcal{F}s'[V''] \Downarrow_n V$, thus $\mathcal{F}s[M] \Downarrow_n V$. ∎

Lemma 4.7 *For all closed terms $M : \tau, M' : \tau'$ and closed frame stacks $\mathcal{F}s : \tau \multimap \tau'', \mathcal{F}s' : \tau' \multimap \tau'''$* *

$$\langle \mathcal{F}s, M\rangle \to \langle \mathcal{F}s', M'\rangle \Rightarrow \forall V.\mathcal{F}s'[M'] \Downarrow_n V \Rightarrow \mathcal{F}s[M] \Downarrow_n V.$$

Proof The proof is by case analysis of $\langle \mathcal{F}s, M\rangle \to \langle \mathcal{F}s', M'\rangle$

If M is not a value then $\mathcal{F}s[M] = \mathcal{F}s'[M']$ and the result is trivial as the semantics is deterministic.

Let $M = \lambda a : \sigma.M_1'$ and $\mathcal{F}s = \mathcal{F}s'' \circ -M_2$, then $\mathcal{F}s'' = \mathcal{F}s'$ and $M' = M_1'[M_2/a]$. By lemma 4.3 $\mathcal{F}s' : \tau'' \multimap \tau'$ and $M' : \tau''$ for some closed type τ''. By Lemma 4.6

$$\mathcal{F}s'[M'] = \mathcal{F}s'[M_1'[M_2/a]] \Downarrow_n V \Leftrightarrow \exists V'.M_1'[M_2/a] \Downarrow_n V' \wedge \mathcal{F}s'[V'] \Downarrow_n V.$$

As $M = \lambda a : \sigma.M_1'$, $M \Downarrow_n M$, thus $MM_2 \Downarrow_n V'$ and $\mathcal{F}s'[V'] \Downarrow_n V$. As $\mathcal{F}s : \tau \multimap \tau''$, $\emptyset; a : \tau \vdash_\emptyset aM_2 : \tau'$, thus by the substitution lemma MM_2 is closed of type τ'. By Lemma 4.6 $\mathcal{F}s'[MM_2] \Downarrow_n V$ and as $\mathcal{F}s'[MM_2] = \mathcal{F}s[M]$, $\mathcal{F}s[M] \Downarrow_n V$.

In the other cases the reasoning is the same. ∎

Lemma 4.8 *Given a closed term $M : \tau$, then for all closed frame stacks $\mathcal{F}s : \tau \multimap \tau'$,* †

$$M \Downarrow_n V \Rightarrow \langle \mathcal{F}s, M\rangle \to^* \langle \mathcal{F}s, V\rangle,$$

Proof I will prove the $\Rightarrow$, by induction on $M \Downarrow_n V$.

$\mathbf{v}_i$: $M \equiv V$**.** Trivial as $\to^*$ is reflexive.

app$_t$: $M \equiv M_1\sigma$**.** From $M_1\sigma \Downarrow_n V$ we know $M_1 \Downarrow_n \Lambda\alpha.M_1'$ and $M_1'[\tau/\alpha] \Downarrow_n V$ for some α and M_1'. As $M_1\sigma$ is closed of type τ, we get $\emptyset;\emptyset \vdash_\emptyset M_1 : \forall\alpha.\tau''$ from

$$\frac{\emptyset;\emptyset \vdash_\emptyset M_1 : \forall\alpha.\tau'' \qquad ftv(\sigma) \in \emptyset}{\emptyset;\emptyset \vdash_\emptyset M_1\sigma : \tau}.$$

By type preservation $\Lambda\alpha.M_1' : \forall\alpha.\tau''$, thus $\emptyset;\emptyset \vdash_{\{\alpha\}} M_1' : \tau''$ and by substitutivity $M_1'[\sigma/\alpha]$ is a closed term of type τ. By induction we have for all closed frame stacks $\mathcal{F}s'', \mathcal{F}s'$ that $\langle \mathcal{F}s'', M_1\rangle \to^* \langle \mathcal{F}s'', \Lambda\alpha.M_1'\rangle$ and $\langle \mathcal{F}s', M_1'[\tau/\alpha]\rangle \to^* \langle \mathcal{F}s', V\rangle$. Thus given $\mathcal{F}s$,

$$\langle \mathcal{F}s, M_1\tau\rangle \to \langle \mathcal{F}s \circ -\tau, M_1\rangle \to^* \langle \mathcal{F}s \circ -\tau, \Lambda\alpha.M_1'\rangle \to \langle \mathcal{F}s, M_1'[\tau/\alpha]\rangle \to^* \langle \mathcal{F}s, V\rangle.$$

rec: $M \equiv \mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2$**.** From $\mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2 \Downarrow_n V$ we know $M_1 \Downarrow_n !(x = M_1' : \tau)$ and $M_2[(\mathbf{fix}\ x : \tau.M_1')/y] \Downarrow_n V$. By type preservation and substitutivity of the type system $M_2[(\mathbf{fix}\ \Downarrow_n !(x = M_1' : \tau)$ is closed, thus by induction we have for all closed frame stacks $\mathcal{F}s, \mathcal{F}s'$ that $\langle \mathcal{F}s, M_1\rangle \to^* \langle \mathcal{F}s, !(x = M' : \tau)\rangle$ and $\langle \mathcal{F}s, M_2[(\mathbf{fix}\ x : \tau.M')/y]\rangle \to^* \langle \mathcal{F}s, V\rangle$. Thus given $\mathcal{F}s$,

$$\begin{aligned}\langle \mathcal{F}s, \mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2\rangle &\to \langle \mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, M_1\rangle\\ &\to^* \langle \mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, !(x = M' : \tau)\rangle\\ &\to \langle \mathcal{F}s, M_2[(\mathbf{fix}\ x : \tau.M')/y]\rangle\\ &\to^* \langle \mathcal{F}s, V\rangle\end{aligned}$$

In the other cases the reasoning is the same. ∎

Lemma 4.9 *For all closed frame stacks $\mathcal{F}s : \tau \multimap \tau'$ and closed terms $M : \tau$* †

$$\mathcal{F}s[M] \Downarrow_n V \Leftrightarrow \langle \mathcal{F}s, M\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle.$$

Proof By Lemma 4.1 $\mathcal{F}s[M]$ is closed. Assume $\mathcal{F}s[M] \Downarrow_n V$ then by Lemma 4.8, $\langle \mathcal{I}d, \mathcal{F}s[M]\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle$. By Lemma 4.4, $\langle \mathcal{I}d, \mathcal{F}s[M]\rangle \rightarrow^* \langle \mathcal{F}s, M\rangle$. As the system is deterministic and $\langle \mathcal{I}d, V\rangle$ is stuck, $\langle \mathcal{I}d, \mathcal{F}s[M]\rangle \rightarrow^* \langle \mathcal{F}s, M\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle$. Thus $\langle \mathcal{F}s, M\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle$.

The $\Leftarrow$ is by induction on the length n of the derivation. Assume $\langle \mathcal{F}s, M\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle$.

If $n = 0$ then $M = V$ and $\mathcal{F}s = \mathcal{I}d$ and as $V = \mathcal{I}d[V] = \mathcal{F}s[M]$, $\mathcal{F}s[M] \Downarrow_n V$.

If $n > 0$ then $\langle \mathcal{F}s, M\rangle \rightarrow \langle \mathcal{F}s', M'\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle$, $\langle \mathcal{F}s', M'\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle$ is a shorter derivation and by Lemma 4.3 $\mathcal{F}s'$ and M' are closed and $\mathcal{F}s' : \tau'' \multimap \tau'$, $M' : \tau''$ for some closed type τ'', thus by induction $\mathcal{F}s'[M'] \Downarrow_n V$. By Lemma 4.7 $\mathcal{F}s[M] \Downarrow_n V$. ∎

Corollary 4.10 *For all closed terms $M : \tau$,*

$$M \Downarrow_n V \Leftrightarrow \langle \mathcal{I}d, M\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle.$$

The next definition is very important as it gives an inductive definition of termination. This makes it easier to prove theorems concerning termination, such as contextual equivalence of terms, etc.

Keeping the evaluation relation using frame stacks in mind, the definition is not surprising.

Definition Let $\searrow$ be a relation on closed frame stacks and closed terms given by

$$\mathrm{base}\frac{}{\langle \mathcal{I}d, V\rangle \searrow} \qquad \mathrm{app}_1\frac{\langle \mathcal{F}s \circ -M_2, M_1\rangle \searrow}{\langle \mathcal{F}s, M_1M_2\rangle \searrow} \qquad \mathrm{spec}_1\frac{\langle \mathcal{F}s \circ -\tau, M\rangle \searrow}{\langle \mathcal{F}s, M\tau\rangle \searrow}$$

$$\mathrm{rec}_1\frac{\langle \mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, M_1\rangle \searrow}{\langle \mathcal{F}s, \mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2\rangle \searrow} \qquad \mathrm{spec}_2\frac{\langle \mathcal{F}s, M[\tau/\alpha]\rangle \searrow}{\langle \mathcal{F}s \circ -\tau, \Lambda\alpha.M\rangle \searrow}$$

$$\mathrm{rec}_2\frac{\langle \mathcal{F}s, M_2[(\mathbf{let}\ !x = !(x = M : \tau)\ \mathbf{in}\ M)/y]\rangle \searrow}{\langle \mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, !(x = M : \tau)\rangle \searrow} \qquad \mathrm{app}_2\frac{\langle \mathcal{F}s, M[M_2/a]\rangle \searrow}{\langle \mathcal{F}s \circ -M_2, \lambda a : \tau.M\rangle \searrow}$$

$$\mathrm{unit}_1\frac{\langle \mathcal{F}s \circ\ \mathbf{let} * = -\ \mathbf{in}\ M_2, M_1\rangle \searrow}{\langle \mathcal{F}s, \mathbf{let} * = M_1\ \mathbf{in}\ M_2\rangle \searrow} \qquad \mathrm{unit}_2\frac{\langle \mathcal{F}s, M_2\rangle \searrow}{\langle \mathcal{F}s\ \mathbf{let} * = -\ \mathbf{in}\ M_2, *\rangle \searrow}$$

$$\mathrm{tn}_1\frac{\langle \mathcal{F}s \circ\ \mathbf{let}\ a_1 \otimes a_2 = -\ \mathbf{in}\ M_2, M_1\rangle \searrow}{\langle \mathcal{F}s, \mathbf{let}\ a_1 \otimes a_2 = M_1\ \mathbf{in}\ M_2\rangle \searrow} \qquad \mathrm{tn}_2\frac{\langle \mathcal{F}s, M_2[N_1, N_2/a_1, a_2]\rangle \searrow}{\langle \mathcal{F}s \circ \mathbf{let}\ a_1 \otimes a_2 = -\ \mathbf{in}\ M_2, N_1 \otimes N_2\rangle \searrow}$$

Remark Note how a derivation of $\searrow$ is unique, thus we have rule inversion.

Lemma 4.11 *For all closed terms $M \in Term(\tau)$ and closed frame stacks $\mathcal{F}s : \tau \multimap \tau'$ such that $\langle \mathcal{F}s, M\rangle \searrow$, if $\langle \mathcal{F}s, M\rangle \searrow$ is derived from $\langle \mathcal{F}s', M'\rangle \searrow$ using the rules of the definition of $\langle \mathcal{F}s, M\rangle \searrow$ then $\mathcal{F}s'$ is closed of type $\tau'' \multimap \tau'$ and M' is closed of type τ'' for some closed type τ''.* *

Proof By induction of the derivation of $\langle \mathcal{F}s, M\rangle \searrow$.

The base case is vacuously true.

In the other cases Lemma 4.3 gives the result. ∎

Lemma 4.12 *For all closed terms $M \in Term(\tau)$ and closed frame stacks $\mathcal{F}s : \tau \multimap \tau'$,*

$$\langle \mathcal{F}s, M\rangle \searrow \Leftrightarrow \exists V \in Val(\tau').\langle \mathcal{F}s, M\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle$$

and $\langle \mathcal{I}d, M\rangle \searrow \Leftrightarrow M \Downarrow_n$

Proof We start with $\Rightarrow$, by induction on $\langle \mathcal{F}s, M\rangle \searrow$.

base: Trivial as $\rightarrow^*$ is reflexive.

app$_1$: By Lemma 4.11 the induction hypothesis is applicable on $\langle \mathcal{F}s \circ -M_2\rangle \searrow$, thus by induction, there exists a $V \in Val(\tau')$ such that $\langle \mathcal{F}s \circ -M_2, M_1\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle$. As $\langle \mathcal{F}s, M_1M_2\rangle \rightarrow \langle \mathcal{F}s \circ -M_2, M_1\rangle$, $\langle \mathcal{F}s, M_1M_2\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle$.

app$_2$: We must have $\mathcal{F}s = \mathcal{F}s' \circ -M_1$ and $M = \lambda a : \tau.M'$ for some $\mathcal{F}s', M_1, \tau, M'$. By induction, $\exists V.\langle \mathcal{F}s', M'[M_1/a]\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle$. Hence

$$\langle \mathcal{F}s' \circ -M_1, \lambda a : \tau.M'\rangle \rightarrow \langle \mathcal{F}s', M'[M_1/a]\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle.$$

The other cases goes just like the three cases given above.

Now to $\Leftarrow$, which we prove by induction on the number of reductions in $\langle \mathcal{F}s, M\rangle \rightarrow^* \langle \mathcal{I}d, V\rangle$.

$n = 0$: We must have $\mathcal{F}s = \mathcal{I}d$ and $M = V$, so by the base rule we have $\langle \mathcal{F}s, M\rangle \searrow$.

$n > 0$: We must have at least one reduction so we proceed by case analysis of the first reduction step:

Assume the first reduction step is

$$\langle \mathcal{F}s, \mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2\rangle \rightarrow \langle \mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, M_1\rangle.$$

By Lemma 4.3 the induction hypothesis is applicable on $\langle \mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, M_1\rangle$, thus $\langle \mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, M_1\rangle \searrow$, and $\langle \mathcal{F}s, M\rangle \searrow$.

The other cases are similar.

The last bit is obvious from Corollary 4.10. ∎

The next lemma expresses that termination is not affected by evaluation, that is, given a term that terminates, anything that term evaluates to also terminates. Likewise, given a term that terminates, anything that evaluates to that term also terminates. This is not surprising as the semantics is deterministic.

Lemma 4.13 *For all closed frame stacks $\mathcal{F}s : \tau \multimap \tau''$, $\mathcal{F}s' : \tau' \multimap \tau''$ and closed terms $M : \tau$, $M' : \tau'$,*

$$\langle \mathcal{F}s, M\rangle \rightarrow^* \langle \mathcal{F}s', M'\rangle \Rightarrow (\langle \mathcal{F}s, M\rangle \searrow \Leftrightarrow \langle \mathcal{F}s', M'\rangle \searrow).$$

Moreover $M \Downarrow_n V \Rightarrow (\langle \mathcal{F}s, M\rangle \searrow \Leftrightarrow \langle \mathcal{F}s, V\rangle \searrow)$.

Proof By induction on the length n of the derivation $\rightarrow^*$. If $n = 0$ then there is nothing to prove as $\mathcal{F}s = \mathcal{F}s'$ and $M = M'$. If $n > 0$ then we proceed by case analysis of the first evaluation step:

(i) $\mathcal{F}s = \mathcal{F}s'' \circ -M_2$ for some $\mathcal{F}s'', M_2$ and $M = \lambda a : \tau_1.M_1$. Then

$$\langle \mathcal{F}s, M\rangle = \langle \mathcal{F}s'' \circ -M_2, \lambda a : \tau_1.M_1\rangle \rightarrow \langle \mathcal{F}s'', M_1[M_2/a]\rangle \rightarrow^* \langle \mathcal{F}s', M'\rangle.$$

Assume $\langle \mathcal{F}s'' \circ -M_2, \lambda a : \tau_1.M_1\rangle \searrow$. Then $\langle \mathcal{F}s'', M_1[M_2/a]\rangle \searrow$ and $\mathcal{F}s'', M_1[M_2/a]$ are closed, thus by induction, $\langle \mathcal{F}s', M'\rangle \searrow$. The $\Leftarrow$ is proved in the same way.

(ii) The other cases are just like the one above.

By Lemma 4.8 we get the last bit. ∎

Lemma 4.14 *For all closed frame stacks $\mathcal{F}s : \tau \multimap \tau'$ and closed terms $M : \tau$, $\langle \mathcal{I}d, \mathcal{F}s[M]\rangle \searrow\Leftrightarrow \langle \mathcal{F}s, M\rangle \searrow$ and $\langle \mathcal{F}s, M\rangle \searrow\Leftrightarrow \mathcal{F}s[M] \Downarrow_n$.*

Proof By Lemma 4.4 $\langle \mathcal{I}d, \mathcal{F}s[M]\rangle \rightarrow^* \langle \mathcal{F}s, M\rangle$, thus by Lemma 4.13, $\langle \mathcal{I}d, \mathcal{F}s[M]\rangle \searrow\Leftrightarrow \langle \mathcal{F}s, M\rangle \searrow$. By Lemma 4.12 $\langle \mathcal{I}d, \mathcal{F}s[M]\rangle \searrow\Leftrightarrow \mathcal{F}s[M] \Downarrow_n$. ∎

The next theorem is the heart of contextual equivalence in Lily. This is so because it states that the call-by-value and the call-by-name semantics coincide when observing terms of exponential type. This is the first non-obvious theorem so far and it shows the power of the type system, as in the next lemma the base case is handled entirely by it and the other cases are handled almost directly by the induction hypothesis.

The intuition for why this theorem holds, is the following. Assume $N[M/a] \Downarrow_n V :!\tau$. Then V must be a thunk and this thunk must be a residual of one of the original thunks in $N[M/a]$. Since none of those thunks have free linear variables the residual thunk must have had all the terms substituted into such variables evaluated.

Theorem 4.15 (Strictness Theorem) *For all $\tau', \tau \in Typ, M \in Term(\tau)$, and open terms, such that $\emptyset; a : \tau \vdash_\emptyset N :!\tau'$*

$$N[M/a] \Downarrow_n \Leftrightarrow \exists V. M \Downarrow_s V \wedge N[V/a] \Downarrow_n \tag{4.1}$$

The proof of the theorem (see page 19) is obtained by a series of lemmas.

Lemma 4.16 *For all frame stacks $\mathcal{F}s$ and terms M, $flv(M) \subseteq flv(\mathcal{F}s[M])$.* *

Proof By induction on $\mathcal{F}s$.

If $\mathcal{F}s = \mathcal{I}d$ then $\mathcal{F}s[M] = M$ thus $flv(\mathcal{F}s[M]) = flv(M)$.

If $\mathcal{F}s = \mathcal{F}s' \circ \mathcal{F}$ for some frame stack $\mathcal{F}s'$ and frame $\mathcal{F}$ then as $\mathcal{F}s[M] = \mathcal{F}s'[\mathcal{F}[M]]$ and $\mathcal{F}[M]$ is a term, by induction $flv(\mathcal{F}[M]) \subseteq flv(\mathcal{F}s[M])$. As no frame binds variables in the hole $flv(M) \subseteq flv(\mathcal{F}[M])$ and $flv(M) \subseteq flv(\mathcal{F}s[M])$. ∎

Lemma 4.17 *For all terms M, type environments Γ, Δ and types τ.* *

$$\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau \quad \Rightarrow \quad flv(M) = \mathrm{dom}(\Delta).$$

Proof By induction on $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$. We have the following cases of the derivation of $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$.

$M \equiv x$ Here $\Delta = \emptyset$ and $flv(M) = \emptyset = \mathrm{dom}(\Delta)$.

$M \equiv a$ Here $\Delta = a : \tau$ and $flv(M) = \{a\} = \mathrm{dom}(\Delta)$

$M \equiv \lambda a : \sigma. M'$ By induction $flv(M') = \mathrm{dom}(\Delta) \cup \{a\}$. Now $flv(\lambda a : \sigma. M') = \mathrm{dom}(\Delta)$ as $a \notin \mathrm{dom}(\Delta)$.

$M \equiv M_1 M_2$ By induction $flv(M_1) = \mathrm{dom}(\Delta_1)$ and $flv(M_2) = \mathrm{dom}(\Delta_2)$ where $\Delta = \Delta_1, \Delta_2$ and $\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset$. Now $flv(M) = flv(M_1) \cup flv(M_2) = \mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2) = \mathrm{dom}(\Delta)$.

$M \equiv \Lambda\alpha. M'$ By induction $flv(M') = \mathrm{dom}(\Delta)$, thus as $flv(M') = flv(\Lambda\alpha. M')$, $flv(\Lambda\alpha. M') = \mathrm{dom}(\Delta)$.

$M \equiv M'\tau'$ By induction $flv(M') = \mathrm{dom}(\Delta)$, thus $flv(M) = \mathrm{dom}(\Delta)$.

$M \equiv !(x = M' : \sigma)$ Here $\Delta = \emptyset$ and by induction $flv(M') = \emptyset$, thus $flv(M) = \emptyset = \mathrm{dom}(\Delta)$.

$M \equiv \mathbf{let}\ !x = M_1\ \mathbf{in}\ M_2$ By induction $flv(M_1) = \mathrm{dom}(\Delta_1)$ and $flv(M_2) = \mathrm{dom}(\Delta_2)$ for some Δ_1, Δ_2 such that $\Delta_1, \Delta_2 = \Delta$. As $flv(M) = flv(M_1) \cup flv(M_2)$, $flv(M) = \mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2) = \mathrm{dom}(\Delta_1, \Delta_2)$.

$M \equiv *$ Here $\Delta = \emptyset$ and $flv(*) = \emptyset = \mathrm{dom}(\Delta)$.

$M \equiv \mathbf{let}\ * = M_1\ \mathbf{in}\ M_2$ By induction $flv(M_1) = \mathrm{dom}(\Delta_1)$ and $flv(M_2) = \mathrm{dom}(\Delta_2)$ for some Δ_1, Δ_2 such that $\Delta = \Delta_1, \Delta_2$. Thus $flv(M) = flv(M_1) \cup flv(M_2) = \mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2) = \mathrm{dom}(\Delta)$.

$M \equiv M_1 \otimes M_2$ By induction $flv(M_1) = \mathrm{dom}(\Delta_1)$ and $flv(M_2) = \mathrm{dom}(\Delta_2)$ for some Δ_1, Δ_2 such that $\Delta = \Delta_1, \Delta_2$. Thus $flv(M) = flv(M_1) \cup flv(M_2) = \mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2) = \mathrm{dom}(\Delta)$.

$M \equiv\ \mathbf{let}\ a_1 \otimes a_2 = M_1\ \mathbf{in}\ M_2$ By induction $flv(M_1) = \mathrm{dom}(\Delta_1)$ and $flv(M_2) = \mathrm{dom}(\Delta_2) \cup \{a_1, a_2\}$ where $\Delta = \Delta_1, \Delta_2$ and $a \neq b$ and $a, b \notin \mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2)$. As $flv(M) = flv(M_1) \cup (flv(M_2) \backslash \{a, b\})$, $flv(M) = \mathrm{dom}(\Delta)$. ∎

Lemma 4.18 *For all terms M, type environments Γ, Δ, frame stacks $\mathcal{F}s$ and types τ.* *

$$\Gamma; \Delta \vdash_{\vec{\alpha}} \mathcal{F}s[M] : \tau \wedge flv(M) = \mathrm{dom}(\Delta) \quad \Rightarrow \quad \exists \tau'. \Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau'$$

Proof By induction on $\mathcal{F}s$.

If $\mathcal{F}s = \mathcal{I}d$ then the result is trivial.

If $\mathcal{F}s = \mathcal{F}s' \circ \mathcal{F}$ for some frame stack $\mathcal{F}s'$ and frame $\mathcal{F}$, then

$$\mathrm{dom}(\Delta) = flv(M) \overset{*}{\subseteq} flv((\mathcal{I}d \circ \mathcal{F})[M]) = flv(\mathcal{F}[M]) \overset{*}{\subseteq} flv(\mathcal{F}s'[\mathcal{F}[M]]) = flv(\mathcal{F}s[M]) \overset{\star}{=} \mathrm{dom}(\Delta).$$

$*$ by Lemma 4.16 and $\star$ by Lemma 4.17. Thus $flv(\mathcal{F}[M]) = \mathrm{dom}(\Delta)$.

By induction there exists a τ'' such that $\Gamma; \Delta \vdash_{\vec{\alpha}} \mathcal{F}[M] : \tau''$. By inspection and Lemma 4.17, $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau'$ for some τ'. ∎

Lemma 4.19 *For all terms M, type environments Δ, frame stacks $\mathcal{F}s$ and types τ.* *

$$\emptyset; \Delta \vdash_{\emptyset} \mathcal{F}s[M] : \tau \wedge flv(M) = \mathrm{dom}(\Delta) \Rightarrow \exists \tau'. \emptyset; \Delta \vdash_{\emptyset} M : \tau' \wedge \mathcal{F}s : \tau' \multimap \tau$$

Proof By induction on $\mathcal{F}s$.

If $\mathcal{F}s = \mathcal{I}d$ then the result is trivial.

If $\mathcal{F}s = \mathcal{F}s' \circ \mathcal{F}$ for some frame stack $\mathcal{F}s'$ and frame $\mathcal{F}$ then as in the proof of Lemma 4.18, $flv(\mathcal{F}[M]) = \mathrm{dom}(\Delta)$, thus by Lemma 4.18, $\emptyset; \Delta \vdash_{\emptyset} \mathcal{F}[M] : \tau''$ for some τ''. By inspection and Lemma 4.17, $\emptyset; a : \tau' \vdash_{\emptyset} \mathcal{F}[a] : \tau''$. By induction $\mathcal{F}s'$ is closed of type $\tau'' \multimap \tau$, thus $\mathcal{F}s$ is closed of type $\tau' \multimap \tau$. ∎

Corollary 4.20 *For all frame stacks $\mathcal{F}s$ and types τ, τ'.* *

$$\emptyset; a : \tau \vdash_{\emptyset} \mathcal{F}s[a] : \tau' \Rightarrow \mathcal{F}s : \tau \multimap \tau'.$$

Lemma 4.21 *For all frame stacks $\mathcal{F}s$, terms M, type environments Δ, Γ, set of type variables $\vec{\alpha}$ and types τ',* *

$$\Gamma; \Delta \vdash_{\vec{\alpha}} \mathcal{F}s[M] : \tau' \Rightarrow \exists \Delta_1, \Delta_2, \tau. \Delta = \Delta_1, \Delta_2 \wedge \Gamma; \Delta_1 \vdash_{\vec{\alpha}} M : \tau \wedge \Gamma; \Delta_2, a : \tau \vdash_{\vec{\alpha}} \mathcal{F}s[a] : \tau'.$$

Where a is a fresh linear variable and $\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset$.

Proof By induction on $\mathcal{F}s$.

If $\mathcal{F}s = \mathcal{I}d$ then $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau'$, thus letting $\tau = \tau'$, $\Delta_1 = \Delta$ and $\Delta_2 = \emptyset$, then by Lemma 3.7

$$\Gamma; \Delta_1 \vdash_{\vec{\alpha}} M : \tau \wedge \Gamma; \Delta_2, a : \tau \vdash_{\vec{\alpha}} \mathcal{I}d[a] : \tau'.$$

If $\mathcal{F}s = \mathcal{F}s' \circ \mathcal{F}$ then $\Gamma, \Delta \vdash_{\vec{\alpha}} \mathcal{F}s'[\mathcal{F}[M]] : \tau'$. By induction there exists $\Delta_1, \Delta_2 = \Delta$ and a type τ'' such that $\Gamma; \Delta_1 \vdash_{\vec{\alpha}} \mathcal{F}[M] : \tau''$, $\Gamma; \Delta_2, a : \tau'' \vdash_{\vec{\alpha}} \mathcal{F}s'[a] : \tau'$ and $\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset$, where a is a fresh linear variable.

We have the following cases for $\mathcal{F}$.

$\mathcal{F} = -\sigma$**:** Here $\Gamma; \Delta_1 \vdash_{\vec{\alpha}} M\sigma : \tau''$, thus by the typing rules $\Gamma; \Delta_1 \vdash_{\vec{\alpha}} M : \tau$ and $ftv(\sigma) \subseteq \vec{\alpha}$. By Lemma 3.7 $\Gamma; \emptyset, a : \tau \vdash_{\vec{\alpha}} a\sigma : \tau''$ and as a is fresh, by the substitution lemma $\Gamma; \Delta_2, a : \tau \vdash_{\vec{\alpha}} \mathcal{F}s[a] : \tau'$.

$\mathcal{F} = -M'$**:** Here $\Gamma; \Delta_1 \vdash_{\vec{\alpha}} MM' : \tau''$, thus by the typing rules $\Gamma; \Delta_3 \vdash_{\vec{\alpha}} M : \tau$ and $\Gamma; \Delta_4 \vdash_{\vec{\alpha}} M' : \sigma$ for some $\Delta_3, \Delta_4, \sigma, \tau$ such that $\tau = \sigma \multimap \tau''$, $\Delta_3, \Delta_4 = \Delta_1$ and $\mathrm{dom}(\Delta_3) \cap \mathrm{dom}(\Delta_4) = \emptyset$. By Lemma 3.7 and the typing rules $\Gamma; \Delta_4, b : \tau \vdash_{\vec{\alpha}} bM' : \tau''$ for some fresh b, and by the substitution lemma $\Gamma; \Delta_2, \Delta_4, b : \tau \vdash_{\vec{\alpha}} \mathcal{F}s'[bM'] : \tau'$. Thus $\Gamma; \Delta_2, \Delta_4, b : \tau \vdash_{\vec{\alpha}} \mathcal{F}s[b]$. It is an easy check to see that $\mathrm{dom}(\Delta_2, \Delta_4, b : \tau) \cap \mathrm{dom}(\Delta_3) = \emptyset$.

The other cases are similar to the two above. ∎

Lemma 4.22 *For all types $\tau, \tau' \in Typ$, frame stacks $\mathcal{F}s$, evaluation frames $\mathcal{F}$ and terms M such that $\emptyset; a : \tau \vdash_\emptyset (\mathcal{F}s \circ \mathcal{F})[M] : \tau'$.* *

If $\mathcal{F} = -\sigma$ and $M = \Lambda\alpha.N$ then $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[N[\sigma/\alpha]] : \tau'$.

If $\mathcal{F} = -N_2$ and $M = \lambda a : \sigma.N_1$ then $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[N_1[N_2/a]] : \tau'$.

If $\mathcal{F} = \mathbf{let}\ !y = -\ \mathbf{in}\ N_2$ and $M = !(x = N_1 : \sigma)$ then

$$\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[N_2[(\mathbf{let}\ !x = !(x = N_1 : \sigma)\ \mathbf{in}\ N_1)/y]] : \tau'.$$

If $\mathcal{F} = \mathbf{let}\ * = -\ \mathbf{in}\ N_2$ and $M = *$ then $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[N_2] : \tau'$.

If $\mathcal{F} = \mathbf{let}\ a_1 \otimes a_2 = -\ \mathbf{in}\ N_3$ and $M = N_1 \otimes N_2$ then $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[N_3[N_1, N_2/a_1, a_2]] : \tau'$.

Proof If $\mathcal{F} = -\sigma$ and $M = \Lambda\alpha.N$ then by Lemma 4.21 either $\emptyset; \emptyset \vdash_\emptyset (\Lambda\alpha.N)\sigma : \tau''$ and $\emptyset; a : \tau, b : \tau'' \vdash_\emptyset \mathcal{F}s[b] : \tau'$ or $\emptyset; a : \tau \vdash_\emptyset (\Lambda\alpha.N)\sigma : \tau''$ and $\emptyset; b : \tau'' \vdash_\emptyset \mathcal{F}s[b] : \tau'$.

In the first case $\emptyset; \emptyset \vdash_\emptyset \Lambda\alpha.N : \forall\alpha.\sigma'$, $ftv(\sigma) = \emptyset$ and $\tau'' = \sigma'[\sigma/\alpha]$, thus $\emptyset; \emptyset \vdash_{\{\alpha\}} N : \sigma'$. By the substitution lemma $\emptyset; \emptyset \vdash_\emptyset N[\sigma/\alpha] : \sigma'[\sigma/\alpha] = \tau''$ and again by the substitution lemma $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[N[\sigma/\alpha]] : \tau'$.

In the latter case $\emptyset; a : \tau \vdash_\emptyset \Lambda\alpha.N : \forall\alpha.\sigma'$, $\sigma'[\sigma/\alpha] = \tau''$, $ftv(\sigma) = \emptyset$ and by Lemma 3.7 $ftv(\tau) = \emptyset$. Thus $\emptyset; a : \tau \vdash_{\{\alpha\}} N : \sigma'$, by the substitution lemma $\emptyset; a : \tau[\sigma/\alpha] \vdash_\emptyset N[\sigma/\alpha] : \sigma'[\sigma/\alpha]$ whence (remembering $ftv(\tau) = \emptyset$) $\emptyset; a : \tau \vdash_\emptyset N[\sigma/\alpha] : \tau''$ and again by the substitution lemma $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[N[\sigma/\alpha]] : \tau'$.

If $\mathcal{F} = -N_2$ and $M = \lambda c : \sigma.N_1$ then by Lemma 4.21, either $\emptyset; \emptyset \vdash_\emptyset (\lambda c : \sigma.N_1)N_2 : \tau''$ and $\emptyset; a : \tau, b : \tau'' \vdash_\emptyset \mathcal{F}s[b] : \tau'$ or $\emptyset; a : \tau \vdash_\emptyset (\lambda c : \sigma.N_1)N_2 : \tau''$ or $\emptyset; b : \tau'' \vdash_\emptyset \mathcal{F}s[b] : \tau'$.

In the first case $\emptyset; \emptyset \vdash_\emptyset \lambda c : \sigma.N_1 : \sigma \multimap \tau''$, $\emptyset; \emptyset \vdash_\emptyset N_2 : \sigma$ and $\emptyset; a : \tau, b : \tau'' \vdash_\emptyset \mathcal{F}s[b] : \tau'$. Thus $\emptyset; c : \sigma \vdash_\emptyset N_1 : \tau''$ and by the substitution lemma $\emptyset; \emptyset \vdash_\emptyset N_1[N_2/c] : \tau''$ and again by the substitution lemma $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[N_1[N_2/c]] : \tau'$.

In the second case either $\emptyset; a : \tau \vdash_\emptyset \lambda c : \sigma.N_1 : \sigma \multimap \tau''$, $\emptyset; \emptyset \vdash_\emptyset N_2 : \sigma$ and $\emptyset; b : \tau'' \vdash_\emptyset \mathcal{F}s[b] : \tau'$ or $\emptyset; \emptyset \vdash_\emptyset \lambda c : \sigma.N_1 : \sigma \multimap \tau''$, $\emptyset; a : \tau \vdash_\emptyset N_2 : \sigma$ and $\emptyset; b : \tau'' \vdash_\emptyset \mathcal{F}s[b] : \tau'$. In the former case $\emptyset; a : \tau, c : \sigma \vdash_\emptyset N_1 : \tau''$ and $\emptyset; \emptyset \vdash_\emptyset N_2 : \sigma$. By the substitution lemma $\emptyset; a : \tau \vdash_\emptyset N_1[N_2/c] : \tau''$ and

again by the substitution lemma $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[N_1[N_2/c]] : \tau'$. In the later case $\emptyset; c : \sigma \vdash_\emptyset N_1 : \tau''$ and $\emptyset; a : \tau \vdash_\emptyset N_2 : \sigma$. Thus by the substitution lemma $\emptyset; a : \tau \vdash_\emptyset N_1[N_2/c] : \tau''$ and again by the substitution lemma $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[N_1[N_2/c]] : \tau'$.

The other cases are similar. ∎

Lemma 4.23 *For all closed frame stacks $\mathcal{F}s$ and terms N, M such that $M \in Term(\tau)$, $\emptyset; a : \tau \vdash_\emptyset N : \tau'$ and $(\mathcal{F}s[N])[M/a] \in Term(!\tau'')$ for some closed type τ'',* †

$$(\mathcal{F}s[N])[M/a] \Downarrow_n \Rightarrow \exists V.M \Downarrow_s V \wedge (\mathcal{F}s[N])[V/a] \Downarrow_n .$$

Proof I will prove

$$\begin{aligned}\forall \tau, \tau', \tau'' \in Typ. \forall M \in Term(\tau). \forall \mathcal{F}s. \forall N. \forall a.\\ \emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[N] :!\tau'' \wedge \mathcal{F}s[M/a] : \tau' \multimap !\tau'' \wedge\\ N[M/a] \in Term(\tau') \wedge \langle \mathcal{F}s[M/a], N[M/a]\rangle \searrow\\ \Rightarrow (\exists V \in Val(\tau). M \Downarrow_s V \wedge \mathcal{F}s[V/a], N[V/a]\rangle \searrow \wedge\\ M \notin Val(\tau) \Rightarrow h(\langle \mathcal{F}s[V/a], N[V/a]\rangle \searrow) < h(\langle \mathcal{F}s[M/a], N[M/a]\rangle \searrow))\end{aligned}$$

where $h(- \searrow)$ is the hight of the derivation of $- \searrow$. This is shown by complete induction on the height of the derivation of $\langle \mathcal{F}s[M/a], N[M/a]\rangle \searrow$. Finally by Lemma 4.14 and 4.12 we get the result.

Assume the height is 1. A derivation of height 1 must be a derivation using only the base rule. Here $\mathcal{F}s[M/a] = \mathcal{I}d$ and $N[M/a] = V$, for some $V \in Val(!\tau')$. As $\emptyset; a : \tau \vdash_\emptyset N :!\tau''$, N must be a thunk or the variable a. The first case cannot be, as thunks have no free linear variables. In the second case, M must be a value, but then the theorem is trivial.

Assume the height is $n > 1$. If M is a value then the theorem is trivial, thus we can assume M is not a value. We then have the following cases for the last rule applied in the derivation.

app$_1$: Here either $N = a$ or not. In the later case $N = N_1N_2$ for some N_1, N_2. Thus $\langle(\mathcal{F}s \circ -N_2)[M/a], N_1[M/a]\rangle \searrow$. By Lemma 4.11 $(\mathcal{F}s \circ -N_2)[M/a]$ is closed of some type $\sigma \multimap !\tau''$ and $N_1[M/a]$ is closed of type σ for some closed type σ. As $(\mathcal{F}s \circ -N_2)[N_1] = \mathcal{F}s[N_1N_2] = \mathcal{F}s[N]$, $\emptyset; a : \tau \vdash_\emptyset (\mathcal{F}s \circ -N_2)[N_1] :!\tau''$. By induction

$$\begin{aligned}\exists V.M \Downarrow_s V \wedge \langle(\mathcal{F}s \circ -N_2)[V/a], N_1[V/a]\rangle \searrow \wedge\\ h(\langle(\mathcal{F}s \circ -N_2)[M/a], N_1[M/a]\rangle \searrow) > h(\langle(\mathcal{F}s \circ -N_2)[V/a], N_1[V/a]\rangle \searrow).\end{aligned}$$

Hence $\exists V.M \Downarrow_s V \wedge \langle \mathcal{F}s[V/a], (N_1N_2)[V/a]\rangle \searrow$ and

$$h(\langle \mathcal{F}s[M/a], (N_1N_2)[M/a]\rangle \searrow) > h(\langle \mathcal{F}s[V/a], (N_1N_2)[V/a]\rangle \searrow).$$

In the former case we must have $M = M_1M_2$ and by Lemma 4.20 $\mathcal{F}s$ is closed, thus $\langle \mathcal{F}s, M_1M_2\rangle \searrow$. If $M_1 = V_1 = \lambda b : \sigma.M_1'$ and $M_2 = V_2$ are both values then $\langle \mathcal{F}s, (\lambda b : \sigma.M_1')V_2\rangle \searrow$, thus $\langle \mathcal{F}s, c[(M_1'[V_2/b])/c]\rangle \searrow$ where c is a fresh linear variable of type τ. By the substitution theorem $\emptyset; c : \tau \vdash_\emptyset \mathcal{F}s[c] : \tau''$. By induction $\langle \mathcal{F}s, V\rangle \searrow$, where $M_1'[V_2/b] \Downarrow_s V$. As $V_2 \Downarrow_s V_2$ and $V_1 \Downarrow_s \lambda b : \sigma.M_1'$, we get $V_1V_2 \Downarrow_s V$. Now

$$h(\langle \mathcal{F}s, M_1M_2\rangle \searrow) = h(\langle \mathcal{F}s, (\lambda b : \sigma.M_1')M_2\rangle \searrow) = h(\langle \mathcal{F}s, M_1'[V_2/b]\rangle \searrow) + 2,$$

thus if $M_1'[V_2/b]$ is a value then we clearly have the height condition. If it is not a value then

$$h(\langle \mathcal{F}s, M_1M_2\rangle \searrow) = h(\langle \mathcal{F}s, M_1'[V_2/b]\rangle \searrow) + 2 > h(\langle \mathcal{F}s, V\rangle \searrow) + 2 > h(\langle \mathcal{F}s, V\rangle \searrow),$$

thus the condition is satisfied.

If M_1 is not a value then $\langle \mathcal{F}s, (M_1V_2)\rangle \searrow$, thus $\langle \mathcal{F}s \circ -V_2, b[M_1/b]\rangle \searrow$ and by induction $\langle \mathcal{F}s \circ -V_2, V_1\rangle \searrow$ and $M_1 \Downarrow_s V_1$. $\langle \mathcal{F}s, V_1V_2\rangle \searrow$. By the height conditions of the induction we see the induction hypothesis can be applied as before, thus $\langle \mathcal{F}s, V\rangle \searrow$ and $M_1 \Downarrow_s V_1 \wedge V_1V_2 \Downarrow_s V$. Combining the strict evaluations gives the result.

If $M_1 = V_1$ is a value and M_2 is not a value then $\langle \mathcal{F}s, M_1M_2\rangle = \langle \mathcal{F}s, (V_1b)[M_2/b]\rangle$ for some fresh linear variable b. By the reasoning above we get $M_2 \Downarrow_s V_2$ and $\langle \mathcal{F}s, V_1V_2\rangle \searrow$ and the induction hypothesis is applicable as above. Thus $V_1V_2 \Downarrow_s V$ and $\langle \mathcal{F}s, V\rangle \searrow$, whence $M_1M_2 \Downarrow_s V$ and a check as above ensures the height condition.

If neither M_1 nor M_2 are values, then $\langle \mathcal{F}s, M_1M_2\rangle \searrow$, thus $\langle \mathcal{F}s \circ -M_2, b[M_1/b]\rangle \searrow$. By induction $M_1 \Downarrow_s V_1$ and $\langle \mathcal{F}s \circ -M_2, V_1\rangle \searrow$ and the induction hypothesis is applicable as before on $\langle \mathcal{F}s, V_1M_2\rangle \searrow$. Thus $V_1M_2 \Downarrow_s V$ $\langle \mathcal{F}s, V\rangle \searrow$ As above we get $M_1M_2 \Downarrow_n V$ and combining the strict evaluations gives the result.

spec$_1$: If $N \neq a$ then $N = N'\tau$ for some N', thus $\langle (\mathcal{F}s \circ -\tau)[M/a], N'[M/a]\rangle \searrow$. By induction

$$\exists V.M \Downarrow_s V \wedge \langle (\mathcal{F}s \circ -\tau)[V/a], N'[V/a]\rangle \searrow \wedge$$
$$h(\langle \mathcal{F}s \circ -\tau)[V/a], N'[V/a]\rangle \searrow) < h(\langle \mathcal{F}s \circ -\tau)[M/a], N'[M/a]\rangle \searrow)$$

Hence

$$\exists V.M \Downarrow_s V \wedge \langle \mathcal{F}s[V/a], (N'\tau)[V/a]\rangle \searrow \wedge$$
$$h\langle \mathcal{F}s[V/a], (N'\tau)[V/a]\rangle \searrow) < h\langle \mathcal{F}s[M/a], (N'\tau)[M/a]\rangle \searrow).$$

If $N = a$ then $M = M_1\sigma$ and $\langle \mathcal{F}s, M_1\sigma\rangle \searrow$ for some closed term M_1 and type σ. If $M_1 = V_1 = \Lambda\alpha.M_1'$ is a value then $\langle \mathcal{F}s, b[(M_1'[\sigma/\alpha])/b]\rangle \searrow$, where b is a fresh linear variable. By induction $M_1'[\sigma/\alpha] \Downarrow_s V$ and $\langle \mathcal{F}s, V\rangle \searrow$. Combining the strict evaluations gives the result.

If M_1 is not a value then $\langle \mathcal{F}s \circ -\sigma, b[M_1/b]\rangle \searrow$ and by induction $\exists V.M_1 \Downarrow_s V \wedge \langle \mathcal{F}s \circ -\sigma, V_1\rangle \searrow$, thus $\langle \mathcal{F}s, V_1\sigma\rangle \searrow$ and the induction hypothesis is applicable as before. Combining the strict evaluations gives the result.

rec$_1$: If $N \neq a$ then $N = \mathbf{let}\ !y = N_1\ \mathbf{in}\ N_2$ for some N_1, N_2, thus

$$\langle (\mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ N_2)[M/a], N_1[M/a]\rangle \searrow .$$

By induction

$$\exists V.M \Downarrow_s V \wedge \langle (\mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ N_2)[V/a], N_1[V/a]\rangle \searrow \wedge$$
$$h(\langle (\mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ N_2)[V/a], N_1[V/a]\rangle \searrow) <$$
$$h(\langle (\mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ N_2)[M/a], N_1[M/a]\rangle \searrow).$$

Hence $\exists V.M \Downarrow_s V \wedge \langle \mathcal{F}s[V/a], (\mathbf{let}\ !y = N_1\ \mathbf{in}\ N_2)[V/a]\rangle \searrow$, and

$$h(\langle \mathcal{F}s[V/a], (\mathbf{let}\ !y = N_1\ \mathbf{in}\ N_2)[V/a]\rangle \searrow) <$$
$$h(\langle \mathcal{F}s[M/a], (\mathbf{let}\ !y = N_1\ \mathbf{in}\ N_2)[M/a]\rangle \searrow).$$

If $N = a$ then $M = \mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2$ for some closed terms M_1 and M_2 and $\langle \mathcal{F}s, \mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2\rangle \searrow$. Thus $\langle \mathcal{F}s \circ\ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, M_1\rangle \searrow$. If $M_1 = V_2 = !(x = M_1' : \tau)$ is a value then $\langle \mathcal{F}s, b[M_2[\mathbf{fix}\ y : \tau.M_1'/y]/b]\rangle \searrow$, thus by induction $M_2[\mathbf{fix}\ y : \tau.M_1'/y] \Downarrow_s V$ and $\langle \mathcal{F}s, V\rangle \searrow$. Combining the strict evaluations and computing the height gives the result.

If M_1 is not a value then $\langle \mathcal{F}s \circ\ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, b[M_1/b]\rangle \searrow$. By induction $M_1 \Downarrow_s V_1$, $\langle \mathcal{F}s \circ\ \mathbf{let}\ !y = -\ \mathbf{in}\ M_2, V_1\rangle \searrow$ and the induction hypothesis is applicable on $\langle \mathcal{F}s, \mathbf{let}\ !y = V_1\ \mathbf{in}\ M_2\rangle \searrow$. Thus $\mathbf{let}\ !y = V_1\ \mathbf{in}\ M_2 \Downarrow_s V$ and $\langle \mathcal{F}s, V\rangle \searrow$. Combining the strict evaluations and computing the height gives the result.

unit$_1$: If $N \neq a$ then $N = \mathbf{let} * = N_1 \ \mathbf{in} \ N_2$ for some N_1, N_2, thus

$$\langle (\mathcal{F}s \circ \mathbf{let} * = - \ \mathbf{in} \ N_2)[M/a], N_1[M/a] \rangle \searrow .$$

By induction

$$\begin{gathered}\exists V.M \Downarrow_s V \wedge \langle (\mathcal{F}s \circ \mathbf{let} * = - \ \mathbf{in} \ N_2)[V/a], N_1[V/a] \rangle \searrow \wedge \\ h(\langle (\mathcal{F}s \circ \mathbf{let} * = - \ \mathbf{in} \ N_2)[V/a], N_1[V/a] \rangle \searrow) < \\ h(\langle (\mathcal{F}s \circ \mathbf{let} * = - \ \mathbf{in} \ N_2)[M/a], N_1[M/a] \rangle \searrow).\end{gathered}$$

Hence $\exists V.M \Downarrow_s V \wedge \langle \mathcal{F}s[V/a], (\mathbf{let} * = N_1 \ \mathbf{in} \ N_2)[V/a] \rangle \searrow$, and

$$\begin{gathered}h(\langle \mathcal{F}s[V/a], (\mathbf{let} * = N_1 \ \mathbf{in} \ N_2)[V/a] \rangle \searrow) < \\ h(\langle \mathcal{F}s[M/a], (\mathbf{let} * = N_1 \ \mathbf{in} \ N_2)[M/a] \rangle \searrow).\end{gathered}$$

If $N = a$ then $M = \mathbf{let} * = M_1 \ \mathbf{in} \ M_2$ for some closed terms M_1 and M_2 and $\langle \mathcal{F}s, \mathbf{let} * = M_1 \ \mathbf{in} \ M_2 \rangle \searrow$. Thus $\langle \mathcal{F}s \circ \ \mathbf{let} * = - \ \mathbf{in} \ M_2, M_1 \rangle \searrow$. If $M_1 = V_2$ is a value then $\langle \mathcal{F}s, b[M_2/b] \rangle \searrow$, thus by induction $M_2 \Downarrow_s V$ and $\langle \mathcal{F}s, V \rangle \searrow$. Combining the strict evaluations and computing the height gives the result.

If M_1 is not a value then $\langle \mathcal{F}s \circ \ \mathbf{let} * = - \ \mathbf{in} \ M_2, b[M_1/b] \rangle \searrow$. By induction $M_1 \Downarrow_s V_1$ and $\langle \mathcal{F}s \circ \ \mathbf{let} * = - \ \mathbf{in} \ M_2, V_1 \rangle \searrow$ and the induction hypothesis is applicable on $\langle \mathcal{F}s, \mathbf{let} * = V_1 \ \mathbf{in} \ M_2 \rangle \searrow$. Thus $\mathbf{let} * = V_1 \ \mathbf{in} \ M_2 \Downarrow_s V$ and $\langle \mathcal{F}s, V \rangle \searrow$. Combining the strict evaluations and computing the height gives the result.

tn$_1$: If $N \neq a$ then $N = \mathbf{let} \ a_1 \otimes a_2 = N_1 \ \mathbf{in} \ N_2$ for some N_1, N_2, thus

$$\langle (\mathcal{F}s \circ \mathbf{let} \ a_1 \otimes a_2 = - \ \mathbf{in} \ N_2)[M/a], N_1[M/a] \rangle \searrow .$$

By induction

$$\begin{gathered}\exists V.M \Downarrow_s V \wedge \langle (\mathcal{F}s \circ \mathbf{let} \ a_1 \otimes a_2 = - \ \mathbf{in} \ N_2)[V/a], N_1[V/a] \rangle \searrow \wedge \\ h(\langle (\mathcal{F}s \circ \mathbf{let} \ a_1 \otimes a_2 = - \ \mathbf{in} \ N_2)[V/a], N_1[V/a] \rangle \searrow) < \\ h(\langle (\mathcal{F}s \circ \mathbf{let} \ a_1 \otimes a_2 = - \ \mathbf{in} \ N_2)[M/a], N_1[M/a] \rangle \searrow).\end{gathered}$$

Hence $\exists V.M \Downarrow_s V \wedge \langle \mathcal{F}s[V/a], (\mathbf{let} \ a_1 \otimes a_2 = N_1 \ \mathbf{in} \ N_2)[V/a] \rangle \searrow$, and

$$\begin{gathered}h(\langle \mathcal{F}s[V/a], (\mathbf{let} \ a_1 \otimes a_2 = N_1 \ \mathbf{in} \ N_2)[V/a] \rangle \searrow) < \\ h(\langle \mathcal{F}s[M/a], (\mathbf{let} \ a_1 \otimes a_2 = N_1 \ \mathbf{in} \ N_2)[M/a] \rangle \searrow).\end{gathered}$$

If $N = a$ then $M = \mathbf{let} \ a_1 \otimes a_2 = M_1 \ \mathbf{in} \ M_2$ for some closed terms M_1 and M_2 and $\langle \mathcal{F}s, \mathbf{let} \ a_1 \otimes a_2 = M_1 \ \mathbf{in} \ M_2 \rangle \searrow$. Thus $\langle \mathcal{F}s \circ \ \mathbf{let} \ a_1 \otimes a_2 = - \ \mathbf{in} \ M_2, M_1 \rangle \searrow$. If $M_1 = V_1 = (M_1' \otimes M_2')$ is a value then $\langle \mathcal{F}s, b[(M_2[M_1', M_2'/a_1, a_2])/b] \rangle \searrow$, thus by induction $M_2[M_1', M_2'/a_1, a_2] \Downarrow_s V$ and $\langle \mathcal{F}s, V \rangle \searrow$. Combining the strict evaluations and computing the height gives the result.

If M_1 is not a value then $\langle \mathcal{F}s \circ \ \mathbf{let} \ a_1 \otimes a_2 = - \ \mathbf{in} \ M_2, b[M_1/b] \rangle \searrow$. By induction $M_1 \Downarrow_s V_1$ and $\langle \mathcal{F}s \circ \ \mathbf{let} \ a_1 \otimes a_2 = - \ \mathbf{in} \ M_2, V_1 \rangle \searrow$ and the induction hypothesis is applicable on $\langle \mathcal{F}s, \mathbf{let} \ a_1 \otimes a_2 = V_1 \ \mathbf{in} \ M_2 \rangle \searrow$. Thus $\mathbf{let} \ a_1 \otimes a_2 = V_1 \ \mathbf{in} \ M_2 \Downarrow_s V$ and $\langle \mathcal{F}s, V \rangle \searrow$. Combining the strict evaluations and computing the height gives the result.

spec$_2$: In all the $?_2$ cases $N = a$ is vacuously true, thus it is safe to assume $N \neq a$. In this case $\mathcal{F}s[M/a] = (\mathcal{F}s' \circ -\sigma)[M/a]$ for some $\mathcal{F}s', \sigma$ and $N = \Lambda\alpha.N'$ for some N'. Thus $\langle \mathcal{F}s'[M/a], (N'[\sigma/\alpha])[M/a] \rangle \searrow$. By Lemma 4.22, $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s'[N'[\sigma/\alpha]] :!\tau''$. By induction

$$\exists V.M \Downarrow_s V \wedge \langle \mathcal{F}s'[V/a], (N'[\sigma/\alpha])[V/a] \rangle \searrow .$$

Hence $\exists V.M \Downarrow_s V \wedge \langle (\mathcal{F}s' \circ -\sigma)[V/a], (\Lambda\alpha.N')[V/a] \rangle \searrow$. From the induction the height condition is clearly satisfied.

rest: As the height is larger than one we cannot have the base rule as the last rule used in the derivation. Thus there is only $?_2$ rules left. These are all handled as the spec_2 case using the specially tailored Lemma 4.22. ∎

The following definition defines an equivalence relation on terms that coincide with contextual equivalence. This is not a big surprise in view of Lemma 4.12, but when generalized to open terms this becomes non-trivial as the next section shows.

Definition (ciu equivalence on *closed* terms) Let M, M' be closed terms of type $\tau \in Typ$. † Then $M \leq_{\text{ciu}} M' : \tau$ if and only if, for all types $\tau' \in Typ$ and frame stacks $\mathcal{F}s : \tau \multimap !\tau'$,

$$\langle \mathcal{F}s, M\rangle \searrow \Rightarrow \langle \mathcal{F}s, M'\rangle \searrow .$$

Further more we define $M =_{\text{ciu}} M' : \tau \Leftrightarrow M \leq_{\text{ciu}} M' : \tau \wedge M' \leq_{\text{ciu}} M : \tau$.

Lemma 4.24 *Let $M, M' \in Term(\tau)$, $M \leq_{\text{ciu}} M' : \tau$ and N be a term such that $N[M/m] \in$* * *$Term(\tau')$. Then $N[M/m] \leq_{\text{ciu}} N[M'/m] : \tau'$.*

Proof I will prove for all closed terms $N[M/m] : \tau'$ and closed frame stacks $\mathcal{F}s[M/m] : \tau' \multimap !\tau''$.

$$\langle \mathcal{F}s[M/m], N[M/m]\rangle \searrow \Rightarrow \langle \mathcal{F}s[M'/m], N[M'/m]\rangle \searrow$$

by induction on $\langle \mathcal{F}s[M/m], N[M/m]\rangle \searrow$.

Assume $N \neq m$.

base: Here $\mathcal{F}s[M/m] = \mathcal{I}d = \mathcal{F}s[M'/m]$ and $N[M/m]$ is a value. Thus $N[M'/m]$ is also a value.

app$_1$: Here $N[M/m] = (N_1N_2)[M/m]$ for some N_1, N_2. By $\searrow$, $\langle(\mathcal{F}s \circ N_2)[M/m], N_1[M/m]\rangle \searrow$. By Lemma 4.11 $(\mathcal{F}s \circ N_2)[M/m]$ is closed of type $\sigma \multimap !\tau''$ for some closed type σ and $N_1[M/m]$ is closed of type σ. By induction $\langle(\mathcal{F}s \circ N_2)[M'/m], N_1[M'/m]\rangle \searrow$ and by $\searrow$, $\langle \mathcal{F}s[M'/m], N_1N_2[M'/m]\rangle \searrow$.

app$_2$: Here $\mathcal{F}s[M/m] = (\mathcal{F}s' \circ -N_2)[M/m]$ and $N[M/m] = (\lambda a : \tau''.N_1)[M/m]$ for some $\mathcal{F}s', N_2, N_1$. By $\searrow$, $\langle \mathcal{F}s'[M/m], (N_1[N_2/a])[M/m]\rangle \searrow$. By Lemma 4.11 $\mathcal{F}s'[M/m]$ is closed of some type σ and $(N_1[N_2/a])[M/m]$ is closed of type σ. By induction $\langle \mathcal{F}s'[M'/m], (N_1[N_2/a])[M'/m]\rangle \searrow$ and by $\searrow$, $\langle \mathcal{F}s[M'/m], (\lambda a : \tau''.N_1)[M'/m]\rangle \searrow$.

The other cases are proved by the same reasoning.

If $N = m$ then as M is closed $M[M/m] = M \neq m$ and $\langle \mathcal{F}s[M/m], M[M/m]\rangle \searrow$, thus as above $\langle \mathcal{F}s[M'/m], M[M'/m]\rangle \searrow$ and by $\leq_{\text{ciu}}$, $\langle \mathcal{F}s[M'/m], M'\rangle \searrow$. ∎

Theorem 4.25 (ciu theorem for *closed* terms) *For all closed terms $M, M' : \tau$, types $\tau' \in$* † *Typ and closed frame stacks $\mathcal{F}s : \tau \multimap !\tau'$.*

$$M =_{\text{ciu}} M' : \tau \Leftrightarrow (\langle \mathcal{F}s, M\rangle \searrow \Leftrightarrow \langle \mathcal{F}s, M'\rangle \searrow) \Leftrightarrow M =_{\text{gnd}} M' : \tau.$$

Proof Given m, N, τ' such that $m : \tau; \emptyset \vdash_\emptyset N :!\tau'$ or $\emptyset; m : \tau \vdash_\emptyset N :!\tau'$. Assume $N[M/m] \Downarrow_n$, then $\langle \mathcal{I}d, N[M/m]\rangle \searrow$. By Lemma 4.24 $\langle \mathcal{I}d, N[M'/m]\rangle \searrow$, thus $N[M'/m] \Downarrow_n$ and $M' =_{\text{gnd}} M : \tau$.

Given a frame stack $\mathcal{F}s : \tau \multimap !\tau'$ assume $\langle \mathcal{F}s, M\rangle \searrow$. By Lemma 4.1, $\emptyset; a : \tau \vdash_\emptyset \mathcal{F}s[a] :!\tau'$. By Lemma 4.14 $\mathcal{F}s[M] \Downarrow_n$. By contextual equivalence of M and M', $(\mathcal{F}s[a])[M/a] = \mathcal{F}s[M] \Downarrow_n \Leftrightarrow$ $\mathcal{F}s[M'] \Downarrow_n$, thus $\mathcal{F}s[M'] \Downarrow_n$ and by Lemma 4.14, $\langle \mathcal{F}s, M\rangle \searrow$.

Hence $=_{\text{gnd}}$ and $=_{\text{ciu}}$ coincide. ∎

Lemma 4.26 *On the conditions of Theorem 4.15*

$$N[M/a] \Downarrow_n \Leftarrow \exists V.M \Downarrow_s V \wedge N[V/a] \Downarrow_n .$$

Proof Given $M \Downarrow_s V \Rightarrow N[M/a] =_{\text{ciu}} N[V/a] :!\tau'$, assume $\exists V.M \Downarrow_s V$. Then $\mathcal{F}s[N[M/a]] \Downarrow_n \Leftrightarrow \mathcal{F}s[N[V/a]] \Downarrow_n$ for all closed frame stacks $\mathcal{F}s :!\tau' \multimap !\tau''$, thus in particular $N[M/a] \Downarrow_n \Leftrightarrow N[V/a] \Downarrow_n$. Assume $N[V/a] \Downarrow_n$ then $N[M/a] \Downarrow_n$.

Hence $M \Downarrow_s V \Rightarrow N[M/a] =_{\text{ciu}} N[V/a] :!\tau'$ is the only thing left. By lemma 4.24 it is enough to prove $M \Downarrow_s V \Rightarrow M =_{\text{ciu}} V : \tau$. This is proved by induction on $M \Downarrow_s V$. The only interesting case is when $M = M_1 M_2$. By induction $M_1 =_{\text{ciu}} \lambda a : \tau.M' : \tau'' \multimap \tau$, $M_2 =_{\text{ciu}} V_2 : \tau''$ and $M'[V_2/a] =_{\text{ciu}} V : \tau$. Thus given a frame stack $\mathcal{F}s :!\tau' \multimap !\tau'''$

$$\begin{aligned}\langle \mathcal{F}s, M_1 M_2\rangle \searrow &\Leftrightarrow \langle \mathcal{F}s \circ -M_2, M_1\rangle \searrow \\ &\Leftrightarrow \langle \mathcal{F}s \circ -M_2, \lambda a : \tau.M'\rangle \searrow \\ &\Leftrightarrow \langle \mathcal{F}s, M'[M_2/a]\rangle \searrow\end{aligned}$$

By lemma 4.24, $\langle \mathcal{F}s, M'[V_2/a]\rangle \searrow$. Hence $M_1 M_2 =_{\text{ciu}} V : \tau$.

Proof of the Strictness theorem (4.15) Letting $\mathcal{F}s = \mathcal{I}d$ in Lemma 4.23 we get $\Rightarrow$, and Lemma 4.26 gives $\Leftarrow$.

Remark In the strictness theorem it is very important that we only consider termination of terms of type $!\tau$ for some $\tau \in Typ$, since for other types the base case of Lemma 4.23 breaks, as the next example shows.

Lemma 4.27 *Let* $\Omega = \textbf{let } !x =!(x = x : \forall\alpha.\alpha) \textbf{ in } x$*, then for any* $\tau \in Typ$*,* $\emptyset;\emptyset \vdash_\emptyset \Omega\tau : \tau$ *and* $\Omega\tau$ *diverges under call-by-name evaluation.*

Proof This is a derivation of $\emptyset;\emptyset \vdash_\emptyset \Omega\tau : \tau$.

$$\text{Let } d \text{ be } \frac{}{x : \forall\alpha.\alpha;\emptyset \vdash_\emptyset x : \forall\alpha.\alpha}\ ftv(x : \forall\alpha.\alpha) = ftv(\forall\alpha.\alpha) = \emptyset \subseteq \emptyset$$

$$\frac{\dfrac{\dfrac{d}{\emptyset;\emptyset \vdash_\emptyset !(x = x : \forall\alpha.\alpha) : \forall\alpha.\alpha} \quad d}{\emptyset;\emptyset \vdash_\emptyset \textbf{let } !x =!(x = x : \forall\alpha.\alpha) \textbf{ in } x : \forall\alpha.\alpha} \quad \dfrac{}{ftv(\tau) \subseteq \emptyset}}{\emptyset;\emptyset \vdash_\emptyset (\textbf{let } !x =!(x = x : \forall\alpha.\alpha) \textbf{ in } x)\tau : \tau}$$

Assume $\Omega\tau \Downarrow_n V$, for some value V, then there must be a least such derivation

$$\frac{!(x = x : \forall\alpha.\alpha) \Downarrow_n !(x = x : \forall\alpha.\alpha) \qquad x[(\textbf{let } !x =!(x = x : \forall\alpha.\alpha) \textbf{ in } x)/x] \Downarrow_n V}{\textbf{let } !x =!(x = x : \forall\alpha.\alpha) \textbf{ in } x \Downarrow_n V}$$

but this cannot be as $x[(\textbf{let } !x =!(x = x : \forall\alpha.\alpha) \textbf{ in } x)/x] = \textbf{let } !x =!(x = x : \forall\alpha.\alpha) \textbf{ in } x$ then has an even smaller derivation.

Example Let $E = \lambda a : \tau.\lambda f : \tau \multimap \tau'.fa$. Now $E(\Omega\tau) \Downarrow_n$ but $E(\Omega\tau) \not\Downarrow_s$.

The next theorem characterizes the fixed point construction in Lily. It does so by proving an induction principle that allows us to reason about fixed points. The outline of this theorem follows a similar theorem in [Pit00], but the proof is different.

Theorem 4.28 (Unwinding Theorem for Lily) *Given a closed type $\tau \in Typ$ and a term M such that $x : \tau; \emptyset \vdash_\emptyset M : \tau$ we define:*

$$\mathbf{fix}^0(M) = \Omega\tau$$
$$\mathbf{fix}^n(M) = M[\mathbf{fix}^{n-1}(M)/x]$$

Then for all closed types τ' and terms N such that $m : \tau; \emptyset \vdash_\emptyset N :!\tau'$ or $\emptyset; m : \tau \vdash_\emptyset N :!\tau'$, $N[\mathbf{fix}\ x : \tau.M/m] \Downarrow_n \Leftrightarrow \exists n.N[\mathbf{fix}^n(M)/m] \Downarrow_n$.

I will prove the Unwinding Theorem by a small series of lemmas.

Lemma 4.29 *For all closed terms $M : \tau$, $\Omega\tau \leq_{\text{ciu}} M : \tau$.* *

Proof It is enough to show $\langle \mathcal{F}s, \Omega\tau\rangle \not\searrow$ for all $\mathcal{F}s : \tau \multimap !\tau'$. Assuming $\langle \mathcal{F}s, \Omega\tau\rangle \searrow$ then by Lemma 4.14 $\mathcal{F}s[\Omega\tau] \Downarrow_n$ and by Lemma 4.6 $\Omega\tau \Downarrow_n V' \wedge \mathcal{F}s[V'] \Downarrow_n V$ for some V', but $\Omega\tau \not\Downarrow_n$. $\lightning$ ∎

Lemma 4.30 *For all closed types $\tau \in Typ$ and terms M such that $x : \tau; \emptyset \vdash_\emptyset M : \tau$, $\mathbf{fix}^n(M) \in Term(\tau)$.* *

Proof This is shown by induction on n. If $n = 0$ then $\mathbf{fix}^n(M) = \Omega\tau$ and we are done by Lemma 4.27. If $n > 0$ then $\mathbf{fix}^n(M) = M[\mathbf{fix}^{n-1}(M)/x]$ and $x : \tau; \emptyset \vdash_\emptyset M : \tau$. By induction $\mathbf{fix}^{n-1}(M) \in Term(\tau)$ and by the substitution lemma we are done. ∎

Lemma 4.31 *For all n and all terms M such that $x : \tau; \emptyset \vdash_\emptyset M : \tau$, $\mathbf{fix}^n(M) \leq_{\text{ciu}} \mathbf{fix}^{n+1}(M)$.* *

Proof This is shown by induction on n. If $n = 0$ then $\mathbf{fix}^n(M) = \Omega\tau$, thus we have the result by Lemma 4.30. If $n > 0$ then $\mathbf{fix}^n(M) = M[\mathbf{fix}^{n-1}(M)]$ and by induction $\mathbf{fix}^{n-1}(M) \leq_{\text{ciu}} \mathbf{fix}^n(M)$, thus by Lemma 4.24 we get the result.

Now I will prove the Unwinding Theorem (Theorem 4.28).

Proof Let us begin with $\Rightarrow$. Here I will prove for all closed frame stacks $\mathcal{F}s[\mathbf{fix}\ x : \tau.M/m] : \tau' \multimap !\tau''$ and terms N such that $\emptyset; m : \tau \vdash_\emptyset N : \tau'$ or $m : \tau; \emptyset \vdash_\emptyset N : \tau'$ †

$$\langle \mathcal{F}s[\mathbf{fix}\ x : \tau.M/m], N[\mathbf{fix}\ x : \tau.M/m]\rangle \searrow \Rightarrow \exists n \geq 0.\langle \mathcal{F}s[\mathbf{fix}^n(M)/m], N[\mathbf{fix}^n(M)/m]\rangle \searrow$$

by induction on $\searrow$. Letting $\mathcal{F}s = \mathcal{I}d$ gives the result by lemma 4.12.

Assume $N \neq m$.

base: Here $\mathcal{F}s = \mathcal{I}d$ and $N[\mathbf{fix}\ x : \tau.M/m] = V$ for some value V. As $N \neq m$, $N[\mathbf{fix}^n(M)/m]$ is also a value. As the computation is done, it is obvious that any n will do.

rec$_1$: Here N must be $\mathbf{let}\ !y = N_1\ \mathbf{in}\ N_2$ for some terms N_1, N_2. Assume

$$\langle \mathcal{F}s[\mathbf{fix}\ x : \tau.M/m], (\mathbf{let}\ !y = N_1\ \mathbf{in}\ N_2)[\mathbf{fix}\ x : \tau.M/m]\rangle \searrow .$$

Then

$$\langle (\mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ N_2)[\mathbf{fix}\ x : \tau.M/m], N_1[\mathbf{fix}\ x : \tau.M/m]\rangle \searrow,$$

and by induction

$$\exists n \geq 0.\langle (\mathcal{F}s \circ \mathbf{let}\ !y = -\ \mathbf{in}\ N_2)[\mathbf{fix}^n(M)/m], N_1[\mathbf{fix}^n(M)/m]\rangle \searrow .$$

Hence

$$\exists n \geq 0.\langle \mathcal{F}s[\mathbf{fix}^n(M)/m], (\mathbf{let}\ !y = N_1\ \mathbf{in}\ N_2)[\mathbf{fix}^n(M)/m]\rangle \searrow .$$

rec$_2$: Here N must be a thunk $!(z = N' : \tau)$ for some N' and $\mathcal{F}s$ must be $\mathcal{F}s' \circ \mathbf{let}\ !y = -\ \mathbf{in}\ N_2$ for some $N_2, \mathcal{F}s'$. Assume

$$\langle (\mathcal{F}s' \circ \mathbf{let}\ !y = -\ \mathbf{in}\ N_2)[\mathbf{fix}\ x : \tau.M/m], !(z = N' : \tau)[\mathbf{fix}\ x : \tau.M/m]\rangle \searrow .$$

Then

$$\langle \mathcal{F}s'[\mathbf{fix}\ x : \tau.M/m], (N_2[(\mathbf{fix}\ z : \tau.N')/y])[\mathbf{fix}\ x : \tau.M/m]\rangle \searrow$$

and by induction

$$\langle \mathcal{F}s'[\mathbf{fix}^n(M)/m], (N_2[(\mathbf{fix}\ z : \tau.N')/y])[\mathbf{fix}^n(M)/m]\rangle \searrow .$$

Hence

$$\langle (\mathcal{F}s' \circ \mathbf{let}\ !y = -\ \mathbf{in}\ N_2)[\mathbf{fix}^n(M)/m], !(z = N' : \tau)[\mathbf{fix}^n(M)/m]\rangle \searrow .$$

The other cases are proved in the same way.

If $N = m$ and $M = x$ then I must show

$$\langle \mathcal{F}s[\mathbf{fix}\ x : \tau.M/m], \mathbf{fix}\ x : \tau.x\rangle \searrow\Rightarrow \ldots$$

Assuming $\langle \mathcal{F}s[\mathbf{fix}\ x : \tau.M/m], \mathbf{fix}\ x : \tau.x\rangle \searrow$, then by $\searrow$

$$\langle \mathcal{F}s[\mathbf{fix}\ x : \tau.M/m], \Omega\tau\rangle \searrow,$$

but from the proof of Lemma 4.30 this cannot be. Thus the theorem holds vacuously in this case.

If $N = m$ and $M \neq x$ then $\langle \mathcal{F}s[\mathbf{fix}\ x : \tau.M/m], (\mathbf{let}\ !x =!(x = M : \tau)\ \mathbf{in}\ M)\rangle \searrow$, thus by alpha equivalence $\langle \mathcal{F}s[\mathbf{fix}\ x : \tau.M/m], (\mathbf{let}\ !m =!(x = M : \tau)\ \mathbf{in}\ M[m/x])\rangle \searrow$. By rule inversion of $\searrow$, $\langle \mathcal{F}s[\mathbf{fix}\ x : \tau.M/m], (M[m/x])[\mathbf{fix}\ x : \tau.M/m]\rangle \searrow$ and $M[m/x] \neq m$, thus we are in the case above. Hence $\langle \mathcal{F}s[\mathbf{fix}^n(M)/m], (M[m/x])[\mathbf{fix}^n(M)/m]\rangle \searrow$ for some n. Now

$$(M[m/x])[\mathbf{fix}^n(M)/m] = M[\mathbf{fix}^n(M)/x] = \mathbf{fix}^{n+1}(M),$$

thus

$$\langle \mathcal{F}s[\mathbf{fix}^n(M)/m], \mathbf{fix}^{n+1}(M)\rangle \searrow .$$

and as $\mathbf{fix}^{n+1}(M)$ is closed

$$\langle \mathcal{I}d, \mathcal{F}s[\mathbf{fix}^{n+1}(M)][\mathbf{fix}^n(M)/m]\rangle \searrow .$$

As $\mathbf{fix}^n(M) \leq_{\text{ciu}} \mathbf{fix}^{n+1}(M)$,

$$\langle \mathcal{I}d, \mathcal{F}s[\mathbf{fix}^{n+1}(M)][\mathbf{fix}^{n+1}(M)/m]\rangle \searrow$$

and

$$\langle \mathcal{F}s[\mathbf{fix}^{n+1}(M)/m], \mathbf{fix}^{n+1}(M)\rangle \searrow .$$

Hence

$$\exists n.\langle \mathcal{F}s[\mathbf{fix}^n(M)/m], N[\mathbf{fix}^n(M)/m]\rangle \searrow .$$

Now $\Leftarrow$. By logic $\forall n.\,(P(n) \Rightarrow Q) \equiv (\exists n.P(n)) \Rightarrow Q$, thus by the ciu theorem for closed terms, it is equivalent to prove

$$\langle \mathcal{F}s, N[\mathbf{fix}^n(M)/m]\rangle \searrow\Rightarrow \langle \mathcal{F}s, N[\mathbf{fix}\ x : \tau.M/m]\rangle \searrow$$

for all n, closed frame stacks $\mathcal{F}s : \tau' \multimap !\tau''$ and terms N such that $m : \tau; \emptyset \vdash_\emptyset N : \tau'$ or $\emptyset; m : \tau \vdash_\emptyset N : \tau'$.

I will proceed by induction on n.

If $n = 0$ then given $\mathcal{F}s$ and N, assume $\langle \mathcal{F}s, N[\mathbf{fix}^n(M)/m]\rangle \searrow$. Then $\langle \mathcal{F}s, N[\Omega\tau/m]\rangle \searrow$. By Lemma 4.29 and Lemma 4.24, $N[\Omega\tau/m] \leq_{\text{ciu}} N[\mathbf{fix}\ x : \tau.M/m]$, thus $\langle \mathcal{F}s, N[\mathbf{fix}\ x : \tau.M/m]\rangle \searrow$.

Assuming $n > 0$. First I will show $\langle \mathcal{F}s, \mathbf{fix}^n(M)\rangle \searrow\Rightarrow \langle \mathcal{F}s, \mathbf{fix}\ x : \tau.M\rangle \searrow$ for all frame stacks $\mathcal{F}s : \tau \multimap !\tau'$. Given $\mathcal{F}s$, as $\mathbf{fix}^n(M) = M[\mathbf{fix}^{n-1}(M)/x]$, by induction

$$\langle \mathcal{F}s, \mathbf{fix}^n(M)\rangle \searrow\Rightarrow \langle \mathcal{F}s, M[\mathbf{fix}^{n-1}(M)/x]\rangle \searrow\Rightarrow \langle \mathcal{F}s, M[\mathbf{fix}\ x : \tau.M/x]\rangle \searrow\Rightarrow \langle \mathcal{F}s, \mathbf{fix}\ x : \tau.M\rangle \searrow$$

Thus $\mathbf{fix}^n(M) \leq_{\text{ciu}} \mathbf{fix}\ x : \tau.M : \tau$ and the result follows from Lemma 4.24. ∎

Definition The Y combinator is given by *

$$Y = \mathbf{fix}\ x : \forall\alpha.!(!\alpha \multimap \alpha) \multimap \alpha.\Lambda\alpha.\lambda f :!(!\alpha \multimap \alpha).\mathbf{let}\ !f' = f\ \mathbf{in}\ f'!(x\alpha(!f'))$$

5 A logical relation

In this section I will introduce four binary relations and prove them to be equal. As one of those is contextual equivalence, we get three other ways, to prove two terms contextual equivalent. The outline of this section follows [Pit00], although as we have a different concept of test functions and constructs not examined in [Pit00] (thunks, unit and tensor product), the details are very different.

I will start by introducing an operation on binary relations from [BPR00]. As in [Pit00] and [Pit05] this operation forms a Galois connection on relations. Again the proof details are quite different compared to [Pit00] and [Pit05], as they use frame stacks instead of test functions.

Definition For all closed types $\tau \in Typ$, let $Test(\tau)$ denote the set $\{\lambda a : \tau.M | \emptyset; \emptyset \vdash_\emptyset \lambda a : \tau.M : \tau \multimap !\tau'\}$ for some $\tau' \in Typ$. For all closed types $\tau, \tau' \in Typ$, let $Rel(\tau, \tau') =_{\text{def}} \{r | r \subseteq Term(\tau) \times Term(\tau')\}$, and let $Rel^*(\tau, \tau') =_{\text{def}} \{s | s \subseteq Test(\tau) \times Test(\tau')\}$.

Given $r \in Rel(\tau, \tau')$, let $r^\top \in Rel^*(\tau, \tau')$

$$r^\top =_{\text{def}} \{(V, V') | \forall (M, M') \in r.VM \Downarrow_n \Leftrightarrow V'M' \Downarrow_n\}$$

Given $s \in Rel^*(\tau, \tau')$, let $s^\top \in Rel(\tau, \tau')$

$$s^\top =_{\text{def}} \{(M, M') | \forall (V, V') \in s.VM \Downarrow_n \Leftrightarrow V'M' \Downarrow_n\}$$

Lemma 5.1 *Given relations $r \in Rel(\tau, \tau')$ and $s \in Rel^*(\tau, \tau')$, the operation $(-)^\top$ forms a* Galois *connection.*

Proof We must show $r \subseteq s^\top \Leftrightarrow s \subseteq r^\top$. Assume $r \subseteq s^\top$. Given $(V, V') \in s$ and $(M, M') \in r$ we have $(M, M') \in s^\top$, thus $VM \Downarrow_n \Leftrightarrow V'M' \Downarrow_n$. Hence $(V, V') \in r^\top$.

Assume $s \subseteq r^\top$ and $(M, M') \in r$. Given $(V, V') \in s$, we have $(V, V') \in r^\top$, thus $VM \Downarrow_n \Leftrightarrow V'M' \Downarrow_n$. Hence $(M, M') \in s^\top$. ∎

As we have shown that the operation $(-)^\top$ gives a Galois connection, the consequences of this results in the next lemma, which is just like in [Pit05].

Lemma 5.2 *The operator $(-)^{\top\top}$ is* monotone $(-_1 \subseteq -_2 \Rightarrow (-_1)^{\top\top} \subseteq (-_2)^{\top\top})$, inflationary $((-) \subseteq (-)^{\top\top})$ *and* idempotent $\left((-)^{\top\top} = (-)^{\top\top\top\top}\right)$. *The operator $(-)^\top$ is* order reversing $\left(-_1 \subseteq -_2 \Rightarrow (-_1)^\top \supseteq (-_2)^\top\right)$.

Proof As $(-)^\top \subseteq (-)^\top$ we get from the Galois connection $- \subseteq (-)^{\top\top}$. Assume $-_1 \subseteq -_2$, then $-_1 \subseteq -_2 \subseteq (-_2)^{\top\top}$, thus by the Galois connection $(-_2)^\top \subseteq (-_1)^\top$. As $(-)^\top$ is order reversing $(-)^{\top\top}$ is monotone. We have $(-)^{\top\top} \subseteq (-)^{\top\top}$, thus by the Galois connection we have $(-)^\top \subseteq (-)^{\top\top\top}$. Hence $(-)^{\top\top} \supseteq (-)^{\top\top\top\top}$ and we have $(-)^{\top\top} = (-)^{\top\top\top\top}$. ∎

Lemma 5.3 $(-)^\top = (-)^{\top\top\top}$

Proof As $(-)^\top$ form a Galois connection,

$$(-)^\top \subseteq (-)^{\top\top\top} \Leftrightarrow (-)^{\top\top} \subseteq (-)^{\top\top}$$
$$(-)^\top \supseteq (-)^{\top\top\top} \Leftrightarrow (-) \subseteq (-)^{\top\top\top\top} \stackrel{\star}{=} (-)^{\top\top}$$

$\star$ is by idempotentcy, thus as $(-)^\top$ is inflationary the right hand side is true. ∎

5.1 The Δ relation

Next I will introduce the Δ function as given in [BPR00]. The images of this function is a relation, which I call the Δ relation.

Definition Given $r_1 \in Rel(\tau_1, \tau_1')$ and $r_2 \in Rel(\tau_2, \tau_2')$, let $r_1 \multimap r_2 \in Rel(\tau_1 \multimap \tau_2, \tau_1' \multimap \tau_2')$ be given by

$$r_1 \multimap r_2 =_{\text{def}} \{(M, M') | \forall (M_1, M_1') \in r_1 . (MM_1, M'M_1') \in r_2\}.$$

Given a family $(R(r) \in Rel(\tau[\sigma/\alpha], \tau'[\sigma'/\alpha']) | \sigma, \sigma' \in Typ, r \in Rel(\sigma, \sigma'))$, we define $\forall r.R(r) \in Rel(\forall\alpha.\tau, \forall\alpha'.\tau')$ to be

$$\forall r.R(r) =_{\text{def}} \{(M, M') | \forall \sigma, \sigma' \in Typ, r \in Rel(\sigma, \sigma') . (M\sigma, M'\sigma') \in R(r)\}.$$

Given $r \in Rel(\tau, \tau')$, let $!r \in Rel(!\tau, !\tau')$ be given by

$$!r =_{\text{def}} \{(!(x = M : \tau), !(x' = M' : \tau)) | (\mathbf{fix}\ x : \tau.M, \mathbf{fix}\ x' : \tau'.M') \in r\}.$$

Given $r_1 \in Rel(\tau_1, \tau_1')$ and $r_2 \in Rel(\tau_2, \tau_2')$, let $r_1 \otimes r_2 \in Rel(\tau_1 \otimes \tau_2, \tau_1' \otimes \tau_2')$ be given by

$$r_1 \otimes r_2 =_{\text{def}} \{(M_1 \otimes M_2), (M_1' \otimes M_2') | (M_1, M_1') \in r_1 \wedge (M_2, M_2') \in r_2\}$$

Definition For all Lily types τ with free type variables $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$, let

$$\Delta_\tau : Rel(\tau_1, \tau_1'), \ldots, Rel(\tau_n, \tau_n') \to Rel(\tau[\vec{\tau}/\vec{\alpha}], \tau[\vec{\tau'}/\vec{\alpha}])$$

be a function defined inductively on τ, with

$$\Delta_{\alpha_i}(\vec{r}/\vec{\alpha}) =_{\text{def}} r_i \tag{5.1}$$
$$\Delta_{\tau_1 \multimap \tau_2}(\vec{r}/\vec{\alpha}) =_{\text{def}} \Delta_{\tau_1}(\vec{r}/\vec{\alpha}) \multimap \Delta_{\tau_2}(\vec{r}/\vec{\alpha}) \tag{5.2}$$
$$\Delta_{\forall\alpha.\tau}(\vec{r}/\vec{\alpha}) =_{\text{def}} \forall r.\Delta_\tau(r^{\top\top}/\alpha, \vec{r}/\vec{\alpha}) \tag{5.3}$$
$$\Delta_{!\tau}(\vec{r}/\vec{\alpha}) =_{\text{def}} (!\Delta_\tau(\vec{r}/\vec{\alpha}))^{\top\top} \tag{5.4}$$
$$\Delta_I(\vec{r}/\vec{\alpha}) =_{\text{def}} \{(*, *)\}^{\top\top} \tag{5.5}$$
$$\Delta_{\tau_1 \otimes \tau_2}(\vec{r}/\vec{\alpha}) =_{\text{def}} (\Delta_{\tau_1}(\vec{r}/\vec{\alpha}) \otimes \Delta_{\tau_2}(\vec{r}/\vec{\alpha}))^{\top\top}. \tag{5.6}$$

For closed Lily types $\tau \in Typ$ we write Δ_τ instead of $\Delta_\tau(\emptyset/\emptyset) \in Rel(\tau, \tau)$.

The purpose of this section is to prove that the Δ relation coincides with contextual equivalence. To do this I must extend the definition of contextual equivalence and Δ to open terms and types. In the same process I extend the definition of ciu equivalence to open terms and types as in [Pit05].

The outline of this section is like the outline in [Pit00] and the lemmas have much of the same wording, but the details are quite different. The main difference in the wording of the lemmas, reflects the use of test functions instead of frame stacks. As most of the work is done in the lemmas, the wording and details of the main theorems are very much the same as in [Pit00] and [Pit05].

5.2 The $=_{\mathbf{obs}}$ relation

To prove that the Δ relation coincides with contextual equivalence I define the concepts adequacy, compatibility and substitutivity. These concepts are properties of relations and there happens to be a largest relation with these properties. This relation is called observational equivalence and it is equivalent to contextual equivalence. Hence we get a description of contextual equivalence from its properties. This approach is taken from [Pit00] and [Pit05]. The definitions of substitutivity and compatibility are straight forward adoptions to Lily, and the definition of adequacy is the only thing that requires some thoughts. This is so since our contextual equivalence is different from what it is in [Pit00] and [Pit05] (our contexts must result in terms of exponential type).

Definition Given a binary relation on arbitrary terms $\mathcal{E}$ we define *

$$\Gamma;\Delta \vdash_{\vec{\alpha}} M\ \mathcal{E}\ M' : \tau \overset{\text{def}}{\Leftrightarrow} \Gamma;\Delta \vdash_{\vec{\alpha}} M : \tau \wedge \Gamma;\Delta \vdash_{\vec{\alpha}} M' : \tau \wedge (M, M') \in \mathcal{E}\ .$$

We say a binary relation $\mathcal{E}$ has the property given by the rule

$$\frac{\Gamma_1;\Delta_1 \vdash_{\vec{\alpha}_1} M_1\ \mathcal{E}\ N_1 : \tau_1 \quad \cdots \quad \Gamma_n;\Delta_n \vdash_{\vec{\alpha}_n} M_n\ \mathcal{E}\ N_n : \tau_n}{\Gamma';\Delta' \vdash_{\vec{\alpha}'} M'\ \mathcal{E}\ N' : \tau'}$$

if and only if $\left(\bigwedge_{i\in\{1,\ldots,n\}} \Gamma_i;\Delta_i \vdash_{\vec{\alpha}_i} M_i\ \mathcal{E}\ N_i : \tau\right) \Rightarrow \Gamma';\Delta' \vdash_{\vec{\alpha}'} M'\ \mathcal{E}\ N' : \tau'$. We say a binary relation $\mathcal{E}$ has a property given by several rules if and only if it has the property given by each of the rules.

Definition (Adequacy) We say a relation $\mathcal{E}$ is *adequate* if for all types $\tau \in Typ$ and terms $M, M' \in Term(!\tau)$.

$$\emptyset;\emptyset \vdash_{\emptyset} M\ \mathcal{E}\ M' :!\tau \Rightarrow (M \Downarrow_n \Leftrightarrow M' \Downarrow_n)$$

Definition (Compatibility) We say a relation $\mathcal{E}$ is *compatible* if

$$\text{Int:}\frac{ftv(\Gamma,\tau) \subseteq \vec{\alpha} \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x:\tau;\emptyset \vdash_{\vec{\alpha}} x\ \mathcal{E}\ x : \tau} \qquad \text{Lin:}\frac{ftv(\Gamma,\tau) \subseteq \vec{\alpha}}{\Gamma; a:\tau \vdash_{\vec{\alpha}} a\ \mathcal{E}\ a : \tau}$$

$$\text{FnAb:}\frac{\Gamma;\Delta, a:\tau \vdash_{\vec{\alpha}} M\ \mathcal{E}\ M' : \tau' \quad a \notin \mathrm{dom}(\Delta)}{\Gamma;\Delta \vdash_{\vec{\alpha}} \lambda a:\tau.M\ \mathcal{E}\ \lambda a:\tau.M' : \tau \multimap \tau'}$$

$$\text{FnApp:}\frac{\Gamma;\Delta_1 \vdash_{\vec{\alpha}} M_1\ \mathcal{E}\ M_1' : \tau \multimap \tau' \quad \Gamma;\Delta_2 \vdash_{\vec{\alpha}} M_2\ \mathcal{E}\ M_2' : \tau \quad \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset}{\Gamma;\Delta_1,\Delta_2 \vdash_{\vec{\alpha}} M_1 M_2\ \mathcal{E}\ M_1' M_2' : \tau'}$$

$$\text{TyAb:}\frac{\Gamma;\Delta \vdash_{\vec{\alpha},\alpha} M\ \mathcal{E}\ M' : \tau \quad \alpha \notin \vec{\alpha} \cup ftv(\Gamma,\Delta)}{\Gamma;\Delta \vdash_{\vec{\alpha}} \Lambda\alpha.M\ \mathcal{E}\ \Lambda\alpha.M' : \forall\alpha.\tau}$$

$$\text{TyApp:}\frac{\Gamma;\Delta \vdash_{\vec{\alpha}} M\ \mathcal{E}\ M' : \forall\alpha.\tau \quad ftv(\tau') \subseteq \vec{\alpha}}{\Gamma;\Delta \vdash_{\vec{\alpha}} M\tau'\ \mathcal{E}\ M'\tau' : \tau[\tau'/\alpha]} \qquad \text{Thunk:}\frac{\Gamma, x:\tau;\emptyset \vdash_{\vec{\alpha}} M\ \mathcal{E}\ M' : \tau \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma;\emptyset \vdash_{\vec{\alpha}} !(x = M : \tau)\ \mathcal{E}\ !(x = M' : \tau) :!\tau}$$

$$\text{Let:}\frac{\Gamma;\Delta_1 \vdash_{\vec{\alpha}} M_1\ \mathcal{E}\ M_1' :!\tau \quad \Gamma, y:\tau;\Delta_2 \vdash_{\vec{\alpha}} M_2\ \mathcal{E}\ M_2' : \tau' \quad y \notin \mathrm{dom}(\Gamma) \quad \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset}{\Gamma;\Delta_1,\Delta_2 \vdash_{\vec{\alpha}} \mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2\ \mathcal{E}\ \mathbf{let}\ !y = M_1'\ \mathbf{in}\ M_2' : \tau'.}$$

$$\text{UnitI:}\frac{}{\Gamma;\emptyset \vdash_{\vec{\alpha}} *\ \mathcal{E}\ * : I}$$

$$\text{UnitE:}\frac{\Gamma;\Delta_1 \vdash_{\vec{\alpha}} M_1\ \mathcal{E}\ M_1' : I \quad \Gamma;\Delta_2 \vdash_{\vec{\alpha}} M_2\ \mathcal{E}\ M_2' : \tau \quad \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset}{\Gamma;\Delta_1,\Delta_2 \vdash_{\vec{\alpha}} \mathbf{let}\ * = M_1\ \mathbf{in}\ M_2\ \mathcal{E}\ \mathbf{let}\ * = M_1'\ \mathbf{in}\ M_2' : \tau}$$

$$\text{TensorI:}\frac{\Gamma;\Delta_1 \vdash_{\vec{\alpha}} M_1\ \mathcal{E}\ M_1' : \tau_1 \quad \Gamma;\Delta_2 \vdash_{\vec{\alpha}} M_2\ \mathcal{E}\ M_2' : \tau_2 \quad \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset}{\Gamma;\Delta_1,\Delta_2 \vdash_{\vec{\alpha}} M_1 \otimes M_1'\ \mathcal{E}\ M_2 \otimes M_2' : \tau_1 \otimes \tau_2}$$

$$\text{TensorE:}\frac{\begin{array}{c}\Gamma;\Delta_1 \vdash_{\vec{\alpha}} M_1\ \mathcal{E}\ M_1' : \tau_1 \otimes \tau_2 \quad \Gamma;\Delta_2, a_1:\tau_1, a_2:\tau_2 \vdash_{\vec{\alpha}} M_2\ \mathcal{E}\ M_2' : \sigma \\ \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset \quad a_1, a_2 \notin \mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2) \quad a_1 \neq a_2\end{array}}{\Gamma;\Delta_1,\Delta_2 \vdash_{\vec{\alpha}} \mathbf{let}\ a_1 \otimes a_2 = M_1\ \mathbf{in}\ M_2\ \mathcal{E}\ \mathbf{let}\ a_1 \otimes a_2 = M_1'\ \mathbf{in}\ M_2' : \sigma}$$

Definition (Substitutivity) We say a relation $\mathcal{E}$ is *substitutive* if †

$$\frac{\Gamma;\Delta \vdash_{\vec{\alpha},\alpha} M\ \mathcal{E}\ M' : \tau \qquad \alpha \notin \vec{\alpha} \qquad ftv(\tau') \subseteq \vec{\alpha}}{\Gamma[\tau'/\alpha];\Delta[\tau'/\alpha] \vdash_{\vec{\alpha}} M[\tau'/\alpha]\ \mathcal{E}\ M'[\tau'/\alpha] : \tau[\tau'/\alpha]}$$

$$\frac{\Gamma, x:\tau;\Delta \vdash_{\vec{\alpha}} M\ \mathcal{E}\ M' : \tau' \qquad \Gamma;\emptyset \vdash_{\vec{\alpha}} N\ \mathcal{E}\ N' : \tau \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma;\Delta \vdash_{\vec{\alpha}} M[N/x]\ \mathcal{E}\ M'[N'/x] : \tau'}$$

$$\frac{\Gamma;\Delta_1, a:\tau \vdash_{\vec{\alpha}} M\ \mathcal{E}\ M' : \tau' \qquad \Gamma;\Delta_2 \vdash_{\vec{\alpha}} N\ \mathcal{E}\ N' : \tau \qquad a \notin \mathrm{dom}(\Delta_1) \qquad \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset}{\Gamma;\Delta_1,\Delta_2 \vdash_{\vec{\alpha}} M[N/a]\ \mathcal{E}\ M'[N'/a] : \tau'.}$$

Remark By comparing the rules of compatibility with the type system, we see that to prove a binary relation, defined on well typed terms, compatible, we get the type criteria for free. Likewise, the substitution lemma gives the type criteria of substitutivity.

Definition We define *observational congruence* $=_{\mathrm{obs}}$ to be the union of all *adequate* and *compatible* relations. †

The next two lemmas and the next theorem proves that $=_{\mathrm{obs}}$ is itself adequate and compatible, and that it coincides with contextual equivalence. The proofs of the lemmas are taken from [Pit05] and the definition of contextual equivalence on open terms, is as in [Pit00].

Lemma 5.4

(i) The identity relation Id is adequate and compatible

(ii) The set of adequate relations is closed under arbitrary union.

(iii) Every compatible relation contains Id.

(iv) The set of compatible relations is closed under composition and reciprocation. The set of adequate relations is closed under composition and reciprocation.

(v) If the union of a non-empty family of compatible relations is transitive then it is compatible.

Proof

(i) By inspection we get the result.

(ii) Let $(M, M') \in \bigcup R_i$ then there must be an i such that $(M, M') \in R_i$ thus $M \Downarrow_n \Leftrightarrow M' \Downarrow_n$.

(iii) This is shown by induction on derivation of the typing relation. Eg. if $\Gamma;\Delta \vdash_{\vec{\alpha}} \lambda a : \tau.M : \tau \multimap \tau'$, then by induction we get $(M, M) \in R$, thus $(\lambda a : \tau.M, \lambda a : \tau.M) \in R$ as R is compatible.

(iv) This is shown by inspection. Eg. assume $(M_1, M_1'') \in (R_1 \circ R_2)$ and $(M_2, M_2'') \in (R_1 \circ R_2)$, then there exists M_1', M_2' such that $(M_1, M_1') \in R_1$, $(M_2, M_2') \in R_1$, $(M_1', M_1'') \in R_2$ and $(M_2', M_2'') \in R_2$, thus $(M_1M_2, M_1'M_2') \in R_1$ and $(M_1'M_2', M_1''M_2'') \in R_2$ as R_i is compatible. Hence $(M_1M_2, M_1''M_2'') \in (R_1 \circ R_2)$.

(v) This is also shown by inspection. Eg. let $(M_1, M_1') \in \bigcup_{i\in I} R_i$ where M_1, M_1' has ! type and $(M_2, M_2') \in \bigcup_{i\in I} R_i$ then there must be an i and an j such that $(M_1, M_1') \in R_i$ and $(M_2, M_2') \in R_j$. By (iii) we have $(M_2, M_2) \in R_i$ and $(M_1', M_1') \in R_j$ thus

$$(\mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2, \mathbf{let}\ !y = M_1'\ \mathbf{in}\ M_2) \in R_i$$
$$(\mathbf{let}\ !y = M_1'\ \mathbf{in}\ M_2, \mathbf{let}\ !y = M_1'\ \mathbf{in}\ M_2') \in R_j$$

and by transitivity $(\mathbf{let}\ !y = M_1\ \mathbf{in}\ M_2, \mathbf{let}\ !y = M_1'\ \mathbf{in}\ M_2') \in (R_i \circ R_j) \in \bigcup_{i\in I} R_i$. ∎

Lemma 5.5 *The $=_{\text{obs}}$ is a compatible and adequate equivalence relation.*

Proof By Lemma 5.4(ii) $=_{\text{obs}}$ is adequate. By Lemma 5.4(i) $=_{\text{obs}}$ is reflexive and non-empty, and by Lemma 5.4(iv) $=_{\text{obs}}$ is symmetric and transitive. Hence by Lemma 5.4(v) $=_{\text{obs}}$ is compatible. ∎

Definition (Ground contextual equivalence) Given terms M, M' such that $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$ and $\Gamma; \Delta \vdash_{\vec{\alpha}} M' : \tau$ we define $\Gamma; \Delta \vdash_{\vec{\alpha}} M =_{\text{gnd}} M' : \tau$ if and only if for all types $\tau' \in Typ$ and all contexts $\mathcal{C}$ such that $\emptyset; \emptyset \vdash_{\emptyset} \mathcal{C}[M] :!\tau'$ and $\emptyset; \emptyset \vdash_{\emptyset} \mathcal{C}[M'] :!\tau'$, we have $\mathcal{C}[M] \Downarrow_n \Leftrightarrow \mathcal{C}[M'] \Downarrow_n$.

Remark When M, M' are closed terms the definition of contextual equivalence degenerates to the definition for closed terms. This is easily seen as there are no free variables or types in M or M' to capture and $\mathcal{C}[M]$ must have type $!\tau$ for some $\tau \in Typ$ in the empty environment.

Theorem 5.6 *Contextual equivalence coincide with $=_{\text{obs}}$. Hence for all terms M, M' such that $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$ and $\Gamma; \Delta \vdash_{\vec{\alpha}} M' : \tau$, we have $\Gamma; \Delta \vdash_{\vec{\alpha}} M =_{\text{gnd}} M' : \tau \Leftrightarrow \Gamma; \Delta \vdash_{\vec{\alpha}} M =_{\text{obs}} M' : \tau$.*

Proof We must show $=_{\text{gnd}}$ is adequate and compatible.

Adequacy: Given a type $\tau \in Typ$ and terms M, M' such that $\emptyset; \emptyset \vdash_{\emptyset} M =_{\text{gnd}} M' :!\tau$. Then $M :!\tau$, $M' :!\tau$ and for the context $\mathcal{C} = -$, $\mathcal{C}[M] = M :!\tau$ and $\mathcal{C}[M'] :!\tau$, thus $M \Downarrow_n \Leftrightarrow M' \Downarrow_n$.

Compatibility: We have to show that $=_{\text{gnd}}$ respects each of the rules of compatibility.

Int: Here we must show $\Gamma, x : \tau; \emptyset \vdash_{\vec{\alpha}} x =_{\text{gnd}} x : \tau$ for all types τ, type environments Γ and intuitionistic variables x such that $ftv(\Gamma, \tau) \subseteq \vec{\alpha}$ and $x \notin \text{dom}(\Gamma)$.
Given a type τ' and a context $\mathcal{C}$ such that $\mathcal{C}[x] :!\tau'$ then by determinism and $\mathcal{C}[x] = \mathcal{C}[x]$,

$$\mathcal{C}[x] \Downarrow_n \Leftrightarrow \mathcal{C}[x] \Downarrow_n .$$

Lin: Similar to the Int case.

FnAb: Assume $\Gamma; \Delta, a : \tau \vdash_{\vec{\alpha}} M =_{\text{gnd}} M' : \tau'$, then given a type τ'' and a context $\mathcal{C}$ such that $\mathcal{C}[\lambda a : \tau.M] :!\tau''$, $\mathcal{C}[\lambda a : \tau.-]$ is a context $\mathcal{C}'$ such that $\mathcal{C}'[M] :!\tau''$. Assume $\mathcal{C}[\lambda a : \tau.M] \Downarrow_n$ then $\mathcal{C}'[M] \Downarrow_n$ and as $\Gamma; \Delta, a : \tau \vdash_{\vec{\alpha}} M =_{\text{gnd}} M' : \tau'$, $\mathcal{C}'[M'] \Downarrow_n$. Thus $\mathcal{C}[\lambda a : \tau.M'] \Downarrow_n$.

FnApp: Assume $\Gamma; \Delta \vdash_{\vec{\alpha}} M_1 M_2 =_{\text{gnd}} M_1' M_2' : \tau$, then given a type τ'' and a context $\mathcal{C}$ such that $\mathcal{C}[M_1 M_2] :!\tau''$. Let $\mathcal{C}'$ be the context $\mathcal{C}[-M_2]$ then $\mathcal{C}[M_1] :!\tau''$. Assume $\mathcal{C}[M_1 M_2] \Downarrow_n$, then $\mathcal{C}'[M_1] \Downarrow_n$. Now $\mathcal{C}'[M_1'] :!\tau''$ and $\mathcal{C}'[M_1'] \Downarrow_n$, thus $\mathcal{C}[M_1' M_2] \Downarrow_n$. Doing the same with the argument M_2 we get $\mathcal{C}[M_1' M_2'] \Downarrow_n$. Thus $\Gamma; \Delta \vdash_{\vec{\alpha}} M_1 M_2 =_{\text{gnd}} M_1' M_2' :!\tau''$.
The rest of the cases are handled similar to the cases above.

By the above $=_{\text{gnd}} \subseteq =_{\text{obs}}$.

To show $=_{\text{obs}} \subseteq =_{\text{gnd}}$, assume $\mathcal{E}$ is a compatible and adequate relation and $\Gamma; \Delta \vdash_{\vec{\alpha}} M \; \mathcal{E} \; M' : \tau$. Given a context $\mathcal{C}$ such that $\emptyset; \emptyset \vdash_{\emptyset} \mathcal{C}[M] :!\tau'$ for some $\tau' \in Typ$ and $\mathcal{C}[M] \Downarrow_n$, then by compatibility $\mathcal{C}[M] \; \mathcal{E} \; \mathcal{C}[M'] :!\tau'$ and by adequacy $\mathcal{C}[M'] \Downarrow_n$. ∎

The next series of lemmas are all parts of the proof that the Δ relation is compatible. Each of the lemmas handle a particular rule, and the lemmas also proves how $\top\top$ closed relations are preserved by the actions on them, as defined in the very beginning of this section.

These lemmas are in wording very similar to lemmas in [Pit00], but the proofs are quite different.

Lemma 5.7 *Given $\tau_1, \tau_1', \tau_2, \tau_2' \in Typ$, suppose $r_1 \in Rel(\tau_1, \tau_1')$, $r_2 \in Rel(\tau_2, \tau_2')$ and $r_2 = r_2^{\top\top}$ then,*

(i) For all values $\lambda a : \tau_1.M : \tau_1 \multimap \tau_2$ *and* $\lambda a : \tau_1'.M' : \tau_1' \multimap \tau_2'$

$$\big(\forall (A, A') \in r_1.(M[A/a], M'[A'/a]) \in r_2\big) \Rightarrow (\lambda a : \tau_1.M, \lambda a : \tau_1'.M') \in r_1 \multimap r_2$$

(ii) For all $(V, V') \in r_2^{\top}$ *and* $(A, A') \in r_1$

$$(\lambda f : \tau_1 \multimap \tau_2.V(fA), \lambda f : \tau_1' \multimap \tau_2'.V'(fA')) \in (r_1 \multimap r_2)^{\top}$$

This is true even if $r_2 \neq r_2^{\top\top}$.

(iii) $r_1 \multimap r_2 = (r_1 \multimap r_2)^{\top\top}$.

Proof

(i) Assume $\forall (A, A') \in r_1.(M[A/a], M'[A'/a]) \in r_2$, as $r_2 = r_2^{\top\top}$,

$$\forall (V, V') \in r_2^{\top}.V(M[A/a]) \Downarrow_n \Leftrightarrow V'(M'[A'/a]) \Downarrow_n .$$

Now $V(M[A/a]) = (VM)[A/a]$ as V is closed. As $\lambda a : \tau.M : \tau_1 \multimap \tau_2$, $\emptyset; a : \tau_1 \vdash_{\emptyset} M : \tau_2$, thus $\emptyset; a : \tau_1 \vdash_{\emptyset} VM :!\tau'$ for some τ'. As $\tau_1 \in Typ$, $\emptyset; b : \tau_1 \vdash_{\emptyset} Vb :!\tau'$. By the substitution theorem $M[A/a]$ is closed and when $A \Downarrow_s V_a$, $M[V_a/a]$ is closed. Thus using the strictness theorem, we get for some V_a, V_b

$$\begin{aligned} V(M[A/a]) \Downarrow_n &\Leftrightarrow A \Downarrow_s V_a \wedge (VM)[V_a/a] \Downarrow_n \\ &\Leftrightarrow A \Downarrow_s V_a \wedge (Vb)[(M[V_a/a])/b] \Downarrow_n \\ &\Leftrightarrow A \Downarrow_s V_a \wedge M[V_a/a] \Downarrow_s V_b \wedge VV_b \Downarrow_n \\ &\Leftrightarrow (\lambda a : \tau_1.M)A \Downarrow_s V_b \wedge (Vb)[V_b/b] \Downarrow_n \\ &\Leftrightarrow (Vb)[(\lambda a : \tau_1.M)A/b] \Downarrow_n \\ &\Leftrightarrow V((\lambda a : \tau_1.M)A) \Downarrow_n . \end{aligned}$$

Likewise

$$V'M'[A'/a] \Downarrow_n \Leftrightarrow V'((\lambda a : \tau_1.M')A') \Downarrow_n .$$

Thus $\forall (A, A') \in r_1.((\lambda a : \tau_1.M)A, (\lambda a : \tau_1'.M')A') \in r_2$. Hence $(\lambda a : \tau_1.M, \lambda a : \tau_1'.M') \in r_1 \multimap r_2$.

(ii) Given $(A, A') \in r_1$, $(V, V') \in r_2^{\top}$ and $(F, F') \in r_1 \multimap r_2$, then $V(FA) \Downarrow_n \Leftrightarrow V'(F'A') \Downarrow_n$ and

$$\begin{aligned} V(FA) \Downarrow_n &\Leftrightarrow \langle \mathcal{I}d, V(FA)\rangle \searrow \Leftrightarrow \langle \mathcal{I}d \circ -F, \lambda f : \tau_1 \multimap \tau_2.V(fA)\rangle \searrow \\ &\Leftrightarrow \langle \mathcal{I}d, (\lambda f : \tau_1 \multimap \tau_2.V(fA))F\rangle \searrow \Leftrightarrow (\lambda f : \tau_1 \multimap \tau_2.V(fA))F \Downarrow_n . \end{aligned}$$

where f is fresh. Likewise $V'(F'A') \Downarrow_n \Leftrightarrow (\lambda f : \tau_1 \multimap \tau_2.V'(fA'))F' \Downarrow_n$. As $(F, F') \in r_1 \multimap r_2$, $(A, A') \in r_1$ and $(V, V') \in r_2^{\top}$, all are arbitrary, we have for all $(A, A') \in r_1$ and $(V, V') \in r_1^{\top}$

$$(\lambda f : \tau_1 \multimap \tau_2.V(fA), \lambda f : \tau_1 \multimap \tau_2.V'(fA')) \in (r_1 \multimap r_2)^{\top}.$$

And we did not use $r_2 = r_2^{\top\top}$.

(iii) Assume $(F, F') \in (r_1 \multimap r_2)^{\top\top}$. Given $(A, A') \in r_1$, $(V, V') \in r_2^{\top}$,

$$(\lambda f : \tau_1 \multimap \tau_2.V(fA), \lambda f : \tau_1 \multimap \tau_2.V'(fA')) \in (r_1 \multimap r_2)^{\top}.$$

As above $(\lambda f : \tau_1 \multimap \tau_2.V(fA))F \Downarrow_n \Leftrightarrow V(FA) \Downarrow_n$. Likewise for $V'(F'A')$, and as (V, V') was arbitrary, $(FA, F'A') \in (r_2^{\top})^{\top} = r_2$. Hence as (A, A') was arbitrary, $(F, F') \in r_1 \multimap r_2$. ∎

Lemma 5.8 *Given types τ_1, τ_1' with at most one free type variable α. If for all $\tau_2, \tau_2' \in Typ$, R is a function $R : Rel(\tau_2, \tau_2') \to Rel(\tau_1[\tau_2/\alpha], \tau_1'[\tau_2'/\alpha])$ then we have the following.*

(i) Given values $\Lambda\alpha.M : \forall\alpha.\tau_1$ and $\Lambda\alpha.M' : \forall\alpha.\tau_1'$, such that

$$\forall r \in Rel(\tau_2, \tau_2').(M[\tau_2/\alpha], M'[\tau_2'/\alpha]) \in R(r).$$

If for all r, $R(r) = (R(r))^{\top\top}$ then $(\Lambda\alpha.M, \Lambda\alpha.M') \in \forall r.R(r)$.

(ii) If $r \in Rel(\tau_2, \tau_2')$ and $(V, V') \in (R(r))^{\top}$ then

$$(\lambda a : (\forall\alpha.\tau_1).V(a\tau_2), \lambda a : (\forall\alpha.\tau_1').V'(a\tau_2')) \in (\forall r.R(r))^{\top}.$$

(iii) If for all r, $R(r) = (R(r))^{\top\top}$ then $\forall r.R(r) = (\forall r.R(r))^{\top\top}$.

Proof

(i) Given $\tau_2, \tau_2' \in Typ$, $R : Rel(\tau_2, \tau_2') \to Rel(\tau_1[\tau_2/\alpha], \tau_1'[\tau_2'/\alpha])$ and $r \in Rel(\tau_2, \tau_2')$, assume $R(r) = (R(r))^{\top\top}$. Then as $(M[\tau_2/\alpha], M'[\tau_2'/\alpha]) \in R(r)$,

$$\forall (V, V') \in (R(r))^{\top}.V(M[\tau_2/\alpha]) \Downarrow_n \Leftrightarrow V'(M'[\tau_2'/\alpha]) \Downarrow_n .$$

By the type system and the substitution lemma $M[\tau_2/\alpha] : \tau_1[\tau_2/\alpha]$, thus $\emptyset; b : \tau_1[\tau_2/\alpha] \vdash_\emptyset Vb :!\sigma$ for some $\sigma \in Typ$. Hence

$$\begin{aligned} V(M[\tau_2/\alpha]) \Downarrow_n &\Leftrightarrow M[\tau_2/\alpha] \Downarrow_s V_\alpha \wedge VV_\alpha \text{ for some } V_\alpha \\ &\Leftrightarrow \Lambda\alpha.M \Downarrow_s \Lambda\alpha.M \wedge M[\tau_2/\alpha] \Downarrow_s V_\alpha \wedge VV_\alpha \text{ for some } V_\alpha \\ &\Leftrightarrow (\Lambda\alpha.M)\tau_2 \Downarrow_s V_\alpha \wedge VV_\alpha \Downarrow_n \text{ for some } V_\alpha \\ &\Leftrightarrow V((\Lambda\alpha.M)\tau_2) \Downarrow_n . \end{aligned}$$

Thus

$$\forall (V, V') \in (R(r))^{\top}.V((\Lambda\alpha.M)\tau_2) \Downarrow_n \Leftrightarrow V'((\Lambda\alpha.M')\tau_2') \Downarrow_n .$$

As $R(r) = (R(r))^{\top\top}$, $((\Lambda\alpha.M)\tau_2, (\Lambda\alpha.M')\tau_2') \in R(r)$. Hence as $\tau_2, \tau_2' \in Typ$ and $r \in Rel(\tau_2, \tau_2')$ were arbitrary,

$$(\Lambda\alpha.M, \Lambda\alpha.M') \in \forall r.R(r).$$

(ii) Assume $r \in Rel(\tau_2, \tau_2')$ and $(V, V') \in (R(r))^{\top}$. Given $(F, F') \in \forall r.R(r)$ we must show

$$(\lambda a : (\forall\alpha.\tau_1).V(a\tau_2))F \Downarrow_n \Leftrightarrow (\lambda a : (\forall\alpha.\tau_1').V'(a\tau_2'))F' \Downarrow_n .$$

Now

$$\begin{aligned} (\lambda a : (\forall\alpha.\tau_1).V(a\tau_2))F \Downarrow_n &\Leftrightarrow \langle \mathcal{I}d, (\lambda a : (\forall\alpha.\tau_1).V(a\tau_2))F\rangle \searrow \\ &\Leftrightarrow \langle \mathcal{I}d \circ -F, \lambda a : (\forall\alpha.\tau_1).V(a\tau_2)\rangle \searrow \\ &\Leftrightarrow \langle \mathcal{I}d, V(F\tau_2)\rangle \searrow \\ &\Leftrightarrow V(F\tau_2) \Downarrow_n . \end{aligned}$$

Likewise $(\lambda a : (\forall\alpha.\tau_1').V'(a\tau_2'))F \Downarrow_n \Leftrightarrow V'(F'\tau_2') \Downarrow_n$, thus we need to show

$$V(F\tau_2) \Downarrow_n \Leftrightarrow V'(F'\tau_2') \Downarrow_n .$$

As $(F\tau_2, F'\tau_2') \in R(r)$ and $(V, V') \in (R(r))^{\top}$ we clearly have this.

(iii) Assume $(F,F) \in (\forall r.R(r))^{\top\top}$. Given $\tau_2, \tau_2' \in Typ$, $r \in Rel(\tau_2, \tau_2')$ and $(V,V') \in (R(r))^{\top}$,

$$(\lambda a : (\forall \alpha.\tau_1).V(a\tau_2))F \Downarrow_n \Leftrightarrow (\lambda a : (\forall \alpha.\tau_1').V'(a\tau_2'))F' \Downarrow_n .$$

As above

$$(\lambda a : (\forall \alpha.\tau_1).V(a\tau_2))F \Downarrow_n \Leftrightarrow V(F\tau_2) \Downarrow_n \text{ and } (\lambda a : (\forall \alpha.\tau_1).V(a\tau_2))F \Downarrow_n \Leftrightarrow V(F\tau_2) \Downarrow_n .$$

Thus $V(F\tau_2) \Downarrow_n \Leftrightarrow V'(F'\tau_2') \Downarrow_n$. As (V,V') was arbitrary $(F\tau_2, F'\tau_2') \in (R(r))^{\top\top}$. Thus $(F\tau_2, F\tau_2') \in R(r)$. As τ_2, τ_2', r was arbitrary $(F,F') \in \forall r.R(r)$.

∎

Lemma 5.9 *For all types $\tau_1, \tau_1' \in Typ$, suppose $r \in Rel(\tau, \tau')$ and $r = r^{\top\top}$. If terms M, M', such that $x : \tau; \emptyset \vdash_\emptyset M : \tau$ and $x' : \tau'; \emptyset \vdash_\emptyset M' : \tau'$, satisfy*

$$\forall (A, A') \in r.(M[A/x], M'[A'/x']) \in r, \tag{5.7}$$

then $(\mathbf{fix}\ x : \tau.M, \mathbf{fix}\ x' : \tau'.M') \in r$.

Proof First a little induction proof on n, proving

$$(\mathbf{fix}^n(M), \mathbf{fix}^n(M')) \in r \text{ for all } n \tag{5.8}$$

Given $(V,V') \in r^{\top}$ and M, M' as assumed in the lemma

$$V\mathbf{fix}^0(M) \Downarrow_n \Leftrightarrow \Omega\tau \Downarrow_s V_\tau \wedge VV_\tau \Downarrow_n \Leftrightarrow \Omega\tau' \Downarrow_s V_{\tau'} \wedge V'V_{\tau'} \Downarrow_n \Leftrightarrow V'\mathbf{fix}^0(M') \Downarrow_n .$$

Thus $(\mathbf{fix}^0(M), \mathbf{fix}^0(M')) \in r^{\top\top} = r$.
If $n > 0$ then as $\mathbf{fix}^n(M) = M[\mathbf{fix}^{n-1}(M)/x]$, we have by induction on n and (5.7) that

$$(\mathbf{fix}^n(M), \mathbf{fix}^n(M')) \in r.$$

Given $(V,V') \in r^{\top}$, by the unwinding theorem

$$V(\mathbf{fix}\ x : \tau.M) \Downarrow_n \Leftrightarrow \exists n.V\mathbf{fix}^n(M) \Downarrow_n \overset{\star}{\Leftrightarrow} \exists n.V'\mathbf{fix}^n(M') \Downarrow_n \Leftrightarrow V'(\mathbf{fix}\ x' : \tau'.M') \Downarrow_n .$$

$\star$ Using existential elimination, (5.8) and existential introduction.
Hence $(\mathbf{fix}\ x : \tau.M, \mathbf{fix}\ x' : \tau'.M') \in r^{\top\top} = r$. ∎

Lemma 5.10 *For all types τ with free type variables in $\vec{\alpha}$, types $\tau_1, \ldots, \tau_n, \tau_1', \ldots, \tau_n' \in Typ$ and $\top\top$ closed term relations $r_i \in Rel(\tau_i, \tau_i')$, such that $|\vec{\alpha}| = n$,* †

$$\Delta_\tau(\vec{r}/\vec{\alpha}) = (\Delta_\tau(\vec{r}/\vec{\alpha}))^{\top\top} \wedge \Delta_\tau(\vec{r}/\vec{\alpha}) \in Rel(\tau[\vec{\tau}/\vec{\alpha}], \tau[\vec{\tau'}/\vec{\alpha}]).$$

Proof The proof goes by induction on the type τ.

$\tau = \alpha_i$ By assumption we have the result.

$\tau = \sigma_1 \multimap \sigma_2$ for some types σ_1, σ_2. We have $\Delta_\tau(\vec{r}/\vec{\alpha}) = \Delta_{\sigma_1 \multimap \sigma_2}(\vec{r}/\vec{\alpha}) = \Delta_{\sigma_1}(\vec{r}/\vec{\alpha}) \multimap \Delta_{\sigma_2}(\vec{r}/\vec{\alpha})$, thus by induction $\Delta_{\sigma_2}(\vec{r}/\vec{\alpha}) = (\Delta_{\sigma_2}(\vec{r}/\vec{\alpha}))^{\top\top}$, $\Delta_{\sigma_2}(\vec{r}/\vec{\alpha}) \in Rel(\sigma_2[\vec{\tau}/\vec{\alpha}], \sigma_2[\vec{\tau'}/\vec{\alpha}])$ and $\Delta_{\sigma_1}(\vec{r}/\vec{\alpha}) \in Rel(\sigma_1[\vec{\tau}/\vec{\alpha}], \sigma_1[\vec{\tau'}/\vec{\alpha}])$. By Lemma 5.7 (iii) we get the first part and by the definition of $\Delta_{\sigma_1}(\vec{r}/\vec{\alpha}) \multimap \Delta_{\sigma_2}(\vec{r}/\vec{\alpha})$, we get the second part.

$\tau = \forall\alpha.\tau'$ for some type τ'. We have $\Delta_\tau(\vec{r}/\vec{\alpha}) = \Delta_{\forall\alpha.\tau'}(\vec{r}/\vec{\alpha}) = \forall r.\Delta_{\tau'}(\vec{r}/\vec{\alpha}, r^{\top\top}/\alpha)$. Given types $\sigma, \sigma' \in Typ$ and a relation r then as $(-)^{\top\top}$ is idempotent $r^{\top\top} = (r^{\top\top})^{\top\top}$ and by induction

$$(\Delta_{\tau'}(\vec{r}/\vec{\alpha}, r^{\top\top}/\alpha) = \Delta_{\tau'}(\vec{r}/\vec{\alpha}, r^{\top\top}/\alpha))^{\top\top}$$
$$\Delta_{\tau'}(\vec{r}/\vec{\alpha}, r^{\top\top}/\alpha) \in Rel(\tau'[\vec{\tau}/\vec{\alpha}, \sigma/\alpha], \tau'[\vec{\tau'}/\vec{\alpha}, \sigma'/\alpha]).$$

As σ, σ', r was arbitrary it works for all σ, σ' and r. Thus by definition $\forall r.\Delta_{\tau'}(\vec{r}/\vec{\alpha}, r^{\top\top}/\alpha) \in Rel(\forall\alpha.\tau'[\vec{\tau}/\vec{\alpha}], \forall\alpha.\tau'[\vec{\tau'}/\vec{\alpha}])$. Given $\sigma, \sigma' \in Typ$ let

$$R : Rel(\sigma, \sigma') \to Rel((\tau'[\vec{\tau}/\vec{\alpha}])[\sigma/\alpha], (\tau'[\vec{\tau'}/\vec{\alpha}])[\sigma'/\alpha])$$

be the function $r \to \Delta_{\tau'}(\vec{r}/\vec{\alpha}, r^{\top\top}/\alpha)$. Then as $\tau'[\vec{\tau}/\vec{\alpha}]$ and $\tau'[\vec{\tau'}/\vec{\alpha}]$ has at most one free type variable, and as for all r, $R(r) = (R(r))^{\top\top}$, by Lemma 5.8 (iii) we get the result.

$\tau = !\tau'$ for some type τ'. We have $\Delta_\tau(\vec{r}/\vec{\alpha}) = (!(\Delta_{\tau'}(\vec{r}/\vec{\alpha}))^{\top\top}$, thus as $(-)^{\top\top}$ is idempotent, we get the first part and by induction

$$\Delta_{\tau'}(\vec{r}/\vec{\alpha}) \in Rel(\tau'[\vec{\tau}/\vec{\alpha}], \tau'[\vec{\tau'}/\vec{\alpha}]).$$

By definition

$$!\Delta_{\tau'}(\vec{r}/\vec{\alpha}) \in Rel((!\tau')[\vec{\tau}/\vec{\alpha}], (!\tau')[\vec{\tau'}/\vec{\alpha}]),$$

and as for all types σ, σ', $(-)^{\top\top} : Rel(\sigma, \sigma') \to Rel(\sigma, \sigma')$,

$$(!\Delta_{\tau'}(\vec{r}/\vec{\alpha}))^{\top\top} \in Rel((!\tau')[\vec{\tau}/\vec{\alpha}], (!\tau')[\vec{\tau'}/\vec{\alpha}]).$$

$\tau = I$**.** By idempotentcy we get the first part and as $* : I$, $(*, *) \in Rel(I, I)$, thus

$$\Delta_\tau(\vec{r}/\vec{\alpha}) = \Delta_I(\vec{r}/\vec{\alpha}) = \{(*, *)\}^{\top\top} \in Rel(I, I) = Rel(I[\vec{\tau}/\vec{\alpha}], I[\vec{\tau'}/\vec{\alpha}]) = Rel(\tau[\vec{\tau}/\vec{\alpha}], \tau[\vec{\tau'}/\vec{\alpha}]).$$

$\tau = \sigma \otimes \sigma'$**.** This case is handled by the same reasoning as the $!\tau$ case. ∎

Lemma 5.11 *Given relations $r_1 \in Rel(\tau_1, \tau_1'), r_2 \in Rel(\tau_2, \tau_2')$, such that $r_1 = r_1^{\top\top}$ and $r_2 = r_2^{\top\top}$, closed terms $(M_1, M_1') \in (!r_1)^{\top\top}$ and terms M_2, M_2', such that $y : \tau_1; \emptyset \vdash_\emptyset M_2 : \tau_2$, $y' : \tau_1'; \emptyset \vdash_\emptyset M_2' : \tau_2'$ and*

$$\forall (A, A') \in r_1.(M_2[A/y], M_2'[A'/y']) \in r_2.$$

Then

$$(\textbf{let } !y = M_1 \textbf{ in } M_2, \textbf{let } !y' = M_1' \textbf{ in } M_2') \in r_2.$$

Proof As r_2 is $\top\top$ closed, we only have to show

$$\forall (V, V') \in r_2^\top.V(\textbf{let } !y = M_1 \textbf{ in } M_2) \Downarrow_n \Leftrightarrow V'(\textbf{let } !y = M_1' \textbf{ in } M_2') \Downarrow_n .$$

Given $(V, V') \in r_2^\top$, then

$$\begin{aligned}
&V(\textbf{let } !y = M_1 \textbf{ in } M_2) \Downarrow_n \\
\overset{\star}{\Leftrightarrow} &M_1 \Downarrow_s V_1 \wedge V(\textbf{let } !y = V_1 \textbf{ in } M_2) \Downarrow_n \text{ for some } V_1 \\
\overset{\star}{\Leftrightarrow} &M_1 \Downarrow_s V_1 \wedge \textbf{let } !y = V_1 \textbf{ in } M_2 \Downarrow_s V_2 \wedge VV_2 \Downarrow_n \text{ for some } V_1, V_2 \\
\overset{\dagger}{\Leftrightarrow} &M_1 \Downarrow_s !(x = M : \tau) \wedge \textbf{let } !y = !(x = M : \tau) \textbf{ in } M_2 \Downarrow_s V_2 \wedge VV_2 \Downarrow_n \text{ for some } V_2, M \\
\Leftrightarrow &M_1 \Downarrow_s !(x = M : \tau) \wedge M_2[(\textbf{fix } x : \tau.M)/y] \Downarrow_s V_2 \wedge VV_2 \Downarrow_n \text{ for some } V_2, M \\
\overset{\star}{\Leftrightarrow} &M_1 \Downarrow_s !(x = M : \tau) \wedge V(M_2[(\textbf{fix } x : \tau.M)/y]) \Downarrow_n \text{ for some } M.
\end{aligned}$$

$\star$ It is an easy check to see that the type conditions of the strictness lemma are fulfilled.

† As $(M_1, M_1') \in (!r_1)^{\top\top}$, $M_1 : !\tau_1$, thus by type preservation $V_1 : !\tau_1$ and therefore V_1 must be a thunk.

Likewise

$$V'(\mathbf{let}\ !y' = M_1'\ \mathbf{in}\ M_2') \Downarrow_n \Leftrightarrow M_1' \Downarrow_s !(x' = M' : \tau') \wedge V'(M_2'[(\mathbf{fix}\ x' : \tau.M')/y']) \Downarrow_n \text{ for some } M'.$$

If $M_1 \Downarrow_s !(x = M : \tau)$ and $M_1' \Downarrow_s !(x' = M' : \tau')$ then $(!(x = M : \tau), !(x' = M' : \tau')) \in (!r_1)^{\top\top}$, as given $(V, V') \in (!r_1)^{\top}$,

$$VM_1 \Downarrow_n \overset{\star}{\Leftrightarrow} V(!(x = M : \tau)) \Downarrow_n .$$

$\star$ By the strictness theorem and determinism of Lily.

Moreover as $!r_1$ only relate values, $(\lambda a : !\tau.a, \lambda a : !\tau.a) \in (!r_1)^{\top}$, thus $M_1 \Downarrow_s !(x = M : \tau) \Leftrightarrow M_1' \Downarrow_s !(x' = M' : \tau')$.

Thus if $M_1 \not\Downarrow_s$ then we have the main result.

Given $(V_1, V_1') \in r_1^{\top}$, then

$$(\lambda a : !\tau.V(\mathbf{let}\ !y = a\ \mathbf{in}\ y), \lambda a : !\tau.V'(\mathbf{let}\ !y' = a\ \mathbf{in}\ y')) \in (!r_1)^{\top},$$

since $!r_1$ only contain values $(!(x = M' : \tau), !(x' = M' : \tau')) \mid (\mathbf{fix}\ x : \tau.M, \mathbf{fix}\ x' : \tau'.M') \in r_1$, and

$$\begin{aligned}
&(\lambda a : !\tau.V(\mathbf{let}\ !y = a\ \mathbf{in}\ y))!(x = M : \tau) \Downarrow_n \\
\Leftrightarrow &V(\mathbf{let}\ !y = !(x = M : \tau)\ \mathbf{in}\ y) \Downarrow_n \\
\Leftrightarrow &\mathbf{let}\ !y = !(x = M : \tau)\ \mathbf{in}\ y \Downarrow_s V_y \wedge VV_y \Downarrow_n \text{ for some } V_y \\
\Leftrightarrow &\mathbf{fix}\ x : \tau.M \Downarrow_s V_y \wedge VV_y \Downarrow_n \text{ for some } V_y \\
\Leftrightarrow &V(\mathbf{fix}\ x : \tau.M) \Downarrow_n .
\end{aligned}$$

Assume $M_1 \Downarrow_s !(x = M : \tau)$ and $(M_1, M_1') \in (!r_1)^{\top\top}$. Then $M_1' \Downarrow_s !(x' = M' : \tau')$ for some $x, x', M, M', \tau, \tau'$. Given $(V, V') \in r_1^{\top}$ then as

$$(\lambda a : !\tau.V(\mathbf{let}\ !y = a\ \mathbf{in}\ y), \lambda a : !\tau.V'(\mathbf{let}\ !y' = a\ \mathbf{in}\ y')) \in (!r_1)^{\top},$$

$V(\mathbf{fix}\ x : \tau.M) \Downarrow_n \Leftrightarrow V'(\mathbf{fix}\ x' : \tau' = M') \Downarrow_n$. Thus $(\mathbf{fix}\ x : \tau.M, \mathbf{fix}\ x' : \tau'.M') \in r_1^{\top\top} = r_1$. With the very first calculation, this gives the result. ∎

Lemma 5.12 *Given relations r_1, r_2, r_3 such that $r_3 = r_3^{\top\top}$, $r_1 \in Rel(\tau_1, \tau_1')$, $r_2 \in Rel(\tau_2, \tau_2')$,* *
$r_3 \in Rel(\tau_3, \tau_3')$, closed terms $(M_1, M_1') \in (r_1 \otimes r_2)^{\top\top}$, and terms (M_3, M_3'), such that $\emptyset; a_1 : \tau_1, a_2 : \tau_2 \vdash_\emptyset M_3 : \tau_3$, $\emptyset; a_1' : \tau_1', a_2' : \tau_2' \vdash_\emptyset M_3' : \tau_3'$ and

$$\forall (A, A') \in r_1, (B, B') \in r_2.(M_3[A/a_1, B/a_2], M_3'[A'/a_1', B'/a_2']) \in r_3,$$

Then

$$(\mathbf{let}\ a_1 \otimes a_1 = M_1\ \mathbf{in}\ M_3, \mathbf{let}\ a_1' \otimes a_2' = M_1'\ \mathbf{in}\ M_3') \in r_3$$

Proof As $r_3 = r_3^{\top\top}$ we only have to show

$$\forall (V, V') \in r_3^{\top}.V(\mathbf{let}\ a_1 \otimes a_2 = M_1\ \mathbf{in}\ M_3) \Downarrow_n \Leftrightarrow V'(\mathbf{let}\ a_2' \otimes a_2' = M_1'\ \mathbf{in}\ M_3') \Downarrow_n .$$

Given $(V,V') \in r_3^\top, (N_1,N_1') \in r_1, (N_2,N_2') \in r_2$ and (M_3,M_3') as in the lemma,

$$\begin{aligned}
&(\lambda a:\tau_1\otimes\tau_2.V(\mathbf{let}\ a_1\otimes a_2 = a\ \mathbf{in}\ M_3))N_1\otimes N_2 \Downarrow_n\\
\Leftrightarrow &V(\mathbf{let}\ a_1\otimes a_2 = N_1\otimes N_2\ \mathbf{in}\ M_3)\Downarrow_n\\
\Leftrightarrow &VV_M\Downarrow_n \wedge \mathbf{let}\ a_1\otimes a_2 = N_1\otimes N_2\ \mathbf{in}\ M_3\Downarrow_s V_M \text{ for some } V_M\\
\Leftrightarrow &VV_M\Downarrow_n \wedge M_3[N_1,N_2/a_1,a_2]\Downarrow_s V_M \text{ for some } V_M\\
\Leftrightarrow &V(M_3[N_1,N_2/a_1,a_2])\Downarrow_n\\
\Leftrightarrow &V'(M_3'[N_1',N_2'/a_1',a_2'])\Downarrow_n\\
\Leftrightarrow &V'V_M'\Downarrow_n \wedge M_3'[N_1',N_2'/a_1',a_2']\Downarrow_s V_M' \text{ for some } V_M'\\
\Leftrightarrow &V'V_M'\Downarrow_n \wedge \mathbf{let}\ a_1'\otimes a_2' = N_1'\otimes N_2'\ \mathbf{in}\ M_3'\Downarrow_s V_M' \text{ for some } V_M'\\
\Leftrightarrow &V'(\mathbf{let}\ a_1'\otimes a_2' = N_1'\otimes N_2'\ \mathbf{in}\ M_3')\Downarrow_n\\
\Leftrightarrow &(\lambda a:\tau_1\otimes\tau_2.V'(\mathbf{let}\ a_1'\otimes a_2' = a\ \mathbf{in}\ M_3'))N_1'\otimes N_2'\Downarrow_n .
\end{aligned}$$

Thus for all $(N_1,N_1') \in r_1, (N_2,N_2') \in r_2, (V,V') \in r_3^\top$ and $(M_3,M_3') \in r_3$ as in the lemma

$$\begin{aligned}
&(\lambda a:\tau_1\otimes\tau_2.V(\mathbf{let}\ a_1\otimes a_2 = a\ \mathbf{in}\ M_3))N_1\otimes N_2\Downarrow_n\Leftrightarrow\\
&\qquad (\lambda a:\tau_1\otimes\tau_2.V'(\mathbf{let}\ a_1'\otimes a_2' = a\ \mathbf{in}\ M_3'))N_1'\otimes N_2'\Downarrow_n .
\end{aligned}$$

Given $(M_1,M_1) \in r_1\otimes r_2$ then $M_1 = N_1\otimes N_2, M_1' = N_1'\otimes N_2'$ and $(N_1,N_1') \in r_1, (N_2,N_2') \in r_2$, thus given $(V,V') \in r_3^\top$ and $(M_3,M_3') \in r_3$ as in the lemma,

$$(\lambda a:\tau_1\otimes\tau_2.V(\mathbf{let}\ a_1\otimes a_2 = a\ \mathbf{in}\ M_3), \lambda a:\tau_1\otimes\tau_2.V'(\mathbf{let}\ a_1'\otimes a_2' = a\ \mathbf{in}\ M_3')) \in (r_1\otimes r_2)^\top.$$

whence by Lemma 5.3,

$$(\lambda a:\tau_1\otimes\tau_2.V(\mathbf{let}\ a_1\otimes a_2 = a\ \mathbf{in}\ M_3), \lambda a:\tau_1\otimes\tau_2.V'(\mathbf{let}\ a_1'\otimes a_2' = a\ \mathbf{in}\ M_3')) \in (r_1\otimes r_2)^{\top\top\top}.$$

thus given $(M_1,M_1') \in (r_1\otimes r_2)^{\top\top}$

$$\begin{aligned}
&V(\mathbf{let}\ a_1\otimes a_2 = M_1\ \mathbf{in}\ M_3)\Downarrow_n\\
\Leftrightarrow &(\lambda a:\tau_1\otimes\tau_2.V(\mathbf{let}\ a_1\otimes a_2 = a\ \mathbf{in}\ M_3))M_1\Downarrow_n\\
\Leftrightarrow &(\lambda a:\tau_1\otimes\tau_2.V'(\mathbf{let}\ a_1'\otimes a_2' = a\ \mathbf{in}\ M_3'))M_1'\Downarrow_n\\
\Leftrightarrow &V'(\mathbf{let}\ a_1'\otimes a_2' = M_1'\ \mathbf{in}\ M_3')\Downarrow_n .
\end{aligned}$$

∎

Lemma 5.13 *Given relations r_1, r_2 such that $r_1 = \{(*,*)\} \in Rel(I,I)$, $r_2 = r_2^{\top\top} \in Rel(\tau_2,\tau_2')$, * closed terms $(M_1,M_1') \in r_1^{\top\top}$ and $(M_2,M_2') \in r_2$. Then*

$$(\mathbf{let}\ * = M_1\ \mathbf{in}\ M_2, \mathbf{let}\ * = M_1'\ \mathbf{in}\ M_2') \in r_2.$$

Proof As $r_2 = r_2^{\top\top}$ we only have to show

$$\forall (V,V') \in r_2^\top. V(\mathbf{let}\ * = M_1\ \mathbf{in}\ M_2)\Downarrow_n \Leftrightarrow V'(\mathbf{let}\ * = M_1'\ \mathbf{in}\ M_2')\Downarrow_n$$

Given $(V,V') \in r_2^\top$, then

$$\begin{aligned}
&V(\mathbf{let}\ * = M_1\ \mathbf{in}\ M_2)\Downarrow_n\\
\Leftrightarrow &M_1\Downarrow_s V_1 \wedge V(\mathbf{let}\ * = V_1\ \mathbf{in}\ M_2)\Downarrow_n \text{ for some } V_1\\
\Leftrightarrow &M_1\Downarrow_s V_1 \wedge \mathbf{let}\ * = V_1\ \mathbf{in}\ M_2\Downarrow_s V_2 \wedge VV_2\Downarrow_n \text{ for some } V_1, V_2\\
\Leftrightarrow &M_1\Downarrow_s * \wedge \mathbf{let}\ * = *\ \mathbf{in}\ M_2\Downarrow_s V_2 \wedge VV_2\Downarrow_n \text{ for some } V_2\\
\Leftrightarrow &M_1\Downarrow_s * \wedge M_2\Downarrow_s V_2 \wedge VV_2\Downarrow_n \text{ for some } V_2\\
\Leftrightarrow &M_1\Downarrow_s * \wedge VM_2\Downarrow_n .
\end{aligned}$$

Likewise $V'(\mathbf{let}\ * = M_1'\ \mathbf{in}\ M_2')\Downarrow_n \Leftrightarrow M_1'\Downarrow_s * \wedge V'M_2'\Downarrow_n$. As r_1 only relates values, $(* \text{ to } *)$ $(\lambda a:I.\mathbf{let}\ * = a\ \mathbf{in}\ !*, \lambda a:I.\mathbf{let}\ * = a\ \mathbf{in}\ !*) \in r_1^\top$, thus $M_1\Downarrow_s \Leftrightarrow M_1'\Downarrow_s$. Hence if $M_1\not\Downarrow_s$ then we have the result. Assume $M_1\Downarrow_s *$, then $M_1'\Downarrow_s *$ and we have the result. ∎

Lemma 5.14 *Given types τ, τ'_i and type variables α_i, α'_i such that $ftv(\tau) \subseteq \{\vec{\alpha}, \vec{\alpha_i}'\}$ then*

$$\Delta_{\tau[\vec{\tau}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha}) = \Delta_{\tau}(\vec{r}/\vec{\alpha}, \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha})/\vec{\alpha}')$$

Proof By induction on τ, we get this case study which gives the result.

$\tau = \alpha$**:** If $\alpha \notin \vec{\alpha}'$ then the result is trivial. If not we get it from equation (5.1), as

$$\Delta_{\alpha[\vec{\tau}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha}) = \Delta_{\tau'_i}(\vec{r}/\vec{\alpha}) = \Delta_{\alpha}(\vec{r}/\vec{\alpha}', \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha})/\vec{\alpha}')$$

$\tau = \tau_1 \multimap \tau_2$**:**

$$\begin{aligned}\Delta_{(\tau_1 \multimap \tau_2)[\vec{\tau}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha}) &= \Delta_{\tau_1[\vec{\tau}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha}) \multimap \Delta_{\tau_2[\vec{\tau}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha}) \\ &= \Delta_{\tau_1}(\vec{r}/\vec{\alpha}', \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha})/\vec{\alpha}') \multimap \Delta_{\tau_2}(\vec{r}/\vec{\alpha}', \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha})/\vec{\alpha}') \\ &= \Delta_{\tau_1 \multimap \tau_2}(\vec{r}/\vec{\alpha}', \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha})/\vec{\alpha}')\end{aligned}$$

$\tau = \forall \alpha.\tau_1$**:**

$$\begin{aligned}\Delta_{\forall \alpha.\tau_2[\vec{\tau}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha}) &= \forall r.\Delta_{\tau_2[\vec{\tau}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha}, r^{\top\top}/\alpha) \\ &= \forall r.\Delta_{\tau_2}(\vec{r}/\vec{\alpha}, r^{\top\top}/\alpha, \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha}, r^{\top\top}/\alpha)/\vec{\alpha}') \\ &= \Delta_{\forall \alpha.\tau_2}(\vec{r}/\vec{\alpha}, \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha})/\vec{\alpha}')\end{aligned}$$

$\tau = !\tau_2$**:**

$$\begin{aligned}\Delta_{!\tau_2[\vec{\tau}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha}) &= (!\Delta_{\tau_2[\vec{\tau}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha}))^{\top\top} \\ &= (!\Delta_{\tau_2}(\vec{r}/\vec{\alpha}), \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha})/\vec{\alpha}')^{\top\top} \\ &= \Delta_{!\tau_2}(\vec{r}/\vec{\alpha}), \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha})/\vec{\alpha}')\end{aligned}$$

$\tau = \tau_1 \otimes \tau_2$**:**

$$\begin{aligned}\Delta_{(\tau_1 \otimes \tau_2)[\vec{\alpha}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha}) &= (\Delta_{\tau_1[\vec{\alpha}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha}) \otimes \Delta_{\tau_2[\vec{\alpha}'/\vec{\alpha}']}(\vec{r}/\vec{\alpha})) \\ &= (\Delta_{\tau_1}(\vec{r}/\vec{\alpha}, \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha})/\vec{\alpha}') \otimes \Delta_{\tau_2}(\vec{r}/\vec{\alpha}, \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha})/\vec{\alpha}')) \\ &= \Delta_{(\tau_1 \otimes \tau_2)}(\vec{r}/\vec{\alpha}, \Delta_{\vec{\tau}'}(\vec{r}/\vec{\alpha})/\vec{\alpha}')\end{aligned}$$

$\tau = I$**:** Trivial as I is a closed type.

∎

The next definition is taken from [Pit00] and slightly adapted to handle linear variables.

Definition (Logical relation on open terms) Suppose $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$ and $\Gamma; \Delta \vdash_{\vec{\alpha}} M' : \tau$ with free type variables $\alpha_1, \ldots, \alpha_l$, free intuitionistic variables $x_1 : \tau_1, \ldots, x_m : \tau_m$ and free linear variables $a_1 : \tau'_1, \ldots, a_n : \tau'_n$, we write

$$\Gamma; \Delta \vdash M \Delta M' : \tau \tag{5.9}$$

if and only if, given any σ_i, σ'_i, $r_i \in Rel(\sigma_i, \sigma'_i)$ where each r_i is $\top\top$ closed, $(N_j, N'_j) \in \Delta_{\tau_j}(\vec{r}/\vec{\alpha})$ and $(M_k, M'_k) \in \Delta_{\tau'_k}(\vec{r}/\vec{\alpha})$, we have

$$\left(M[\vec{\sigma}/\vec{\alpha}, \vec{N}/\vec{x}, \vec{M}/\vec{a}], M'[\vec{\sigma'}/\vec{\alpha}, \vec{N'}/\vec{x}, \vec{M'}/\vec{a}]\right) \in \Delta_{\tau}(\vec{r}/\vec{\alpha}) \tag{5.10}$$

Remark Please note how the terms substituted in are *closed* terms. This makes the Δ relation very close to that of $=_{\text{ciu}}$ on open terms (see page 35).

Lemma 5.15 *The Δ relation for open terms* (5.9) *is compatible.* †

Proof We must show that $\Gamma; \Delta \vdash_{\vec{\alpha}} M \Delta M' : \tau$ is closed under the axioms of compatibility.

Variables: We substitute terms in relation into the variables, thus the axioms are satisfied by definition.

FnApp: Assume $\Gamma; \Delta_1 \vdash_{\vec{\alpha}} M_1 \Delta M_1' : \tau \multimap \tau'$ and $\Gamma; \Delta_2 \vdash_{\vec{\alpha}} M_2 \Delta M_2' : \tau$. Given terms, types and relations as in the definition of Δ, then we only have to show

$$(N_1 N_2, N_1' N_2') \in \Delta_{\sigma'}(\vec{r}/\vec{\alpha})$$

where the N's and σ' are the results of performing the substitutions as in the definition.

Performing the same substitutions on M_1, M_1', M_2, M_2' and using the definition of $\Delta_{\sigma \multimap \sigma'}$ we get the result.

FnAb: Performing substitutions as above and noting that $\Delta_{\tau'}(\vec{r}/\vec{\alpha})$ is $\top\top$ closed, we get the result from Lemma 5.7(i).

TyAb: Given $\Gamma; \Delta \vdash_{\vec{\alpha},\alpha} M \Delta M' : \tau$ and $\alpha \notin ftv(\Gamma, \Delta) \cup \vec{\alpha}$, I must show

$$\Gamma; \Delta \vdash_{\vec{\alpha}} \Lambda\alpha.M \Delta \Lambda\alpha.M' : \forall\alpha.\tau$$

Given $\vec{N}, \vec{M}, \vec{\sigma}, \vec{N}', \vec{M}', \vec{\sigma}', \vec{r}$ where $r_i = r_i^{\top\top}$ for all i, as in the definition of Δ on open terms, I must show

$$\Big(\Lambda\alpha.M[\vec{\sigma}, \vec{N}, \vec{M}/\vec{\alpha}, \vec{x}, \vec{a}], \Lambda\alpha.M'[\vec{\sigma}', \vec{N}', \vec{M}'/\vec{\alpha}, \vec{x}, \vec{a}]\Big) \in \Delta_{\forall\alpha.\tau}(\vec{r}/\vec{\alpha}) = \forall r.\Delta_\tau(\vec{r}/\vec{\alpha}, r^{\top\top}/\alpha)$$

Given types $\sigma, \sigma' \in Typ$ and a relation $r \in Rel(\sigma, \sigma')$, let $R : Rel(\sigma, \sigma') \to Rel(\tau[\vec{\sigma}/\vec{\alpha}, \sigma/\alpha], \tau[\vec{\sigma}'/\vec{\alpha}, \sigma'/\alpha)$ be the function $\lambda r.\Delta_\tau(\vec{r}/\vec{\alpha}, r^{\top\top}/\alpha)$. This function is well-defined by idempotentcy and $\Gamma; \Delta \vdash_{\vec{\alpha},\alpha} M \Delta M' : \tau$. By idempotentcy Lemma 5.10 then applies and as σ, σ', r all were arbitrary, we got all the requirements of Lemma 5.8(i), which gives the desired result.

TyApp: Performing substitutions as above and specializing with τ', τ' and $\Delta_{\tau'}()$ we get the result by Lemma 5.14.

Thunks: Performing substitutions as above we end up with terms satisfying the conditions of Lemma 5.9 where r is $\Delta_\sigma()$ and σ is the type of the resulting terms. By Lemma 5.10, Lemma 5.9, the definition of $!\Delta_\sigma$ and the fact that $(-)^{\top\top}$ is monotone, we get the result.

Let: Performing substitutions as above we end up with closed terms $(N_1, N_1') \in \Delta_{!\sigma}$ and terms N_2, N_2', satisfying the requirements of Lemma 5.11 where $r_1 = \Delta_\sigma$ and $r_2 = \Delta_{\sigma'}$. By definition of $\Delta_{!\sigma}$ and Lemma 5.10 we get the result by Lemma 5.11.

UnitI: Performing substitutions as above we end up with $(*, *) \in \Delta_I$ which we get by monotonicity of $(-)^{\top\top}$.

UnitE: Performing substitutions as above we end up with terms satisfying the requirements of Lemma 5.13 where $r_1 = (*, *)$ and $r_2 = \Delta_\sigma$, thus by the lemma we get the result.

TensorI: Performing substitutions as above we get the result by definition of $\Delta_\sigma \otimes \Delta_{\sigma'}$ and monotonicity of $(-)^{\top\top}$.

TensorE: Performing substitutions as above we end up with terms $(N_1, N_1') \in \Delta_{\sigma \otimes \sigma'}$ and terms N_3, N_3' such that the requirements of Lemma 5.12 where $r_1 = \Delta_\sigma, r_2 = \Delta_{\sigma'}, r_3 = \Delta_{\sigma''}$ is satisfied. By Lemma 5.10, we get the result from Lemma 5.12.

∎

Corollary 5.16 *As a compatible relation is reflexive,*

$$\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau \Rightarrow \Gamma; \Delta \vdash_{\vec{\alpha}} M \Delta M : \tau$$

Definition Given a set of type variables $\vec{\alpha}$, an intuitionistic type environment Γ, a linear type environment Δ and a type τ such that $ftv(\tau, \Delta, \Gamma) \subseteq \vec{\alpha}$, let the set of $(\Gamma, \Delta, \vec{\alpha}, \tau)$–closing substitutions be the following set of functions defined on terms t such that $\Gamma; \Delta \vdash_{\vec{\alpha}} t : \tau$, *

$$\{\lambda t.t[\vec{\tau}, \vec{N'}, \vec{N}/\vec{\alpha}, \vec{x}, \vec{a}] \mid \vec{\tau} \in Typ, \vec{N}, \vec{N'} \subseteq Term(\tau[\vec{\tau}/\vec{\alpha}]), \mathrm{dom}(\Gamma) = \vec{x}, \mathrm{dom}(\Delta) = \vec{a}\}.$$

I will omit the type τ when it is clear from the context. Given a closing substitution σ I will overload σ to types by letting $\sigma(\tau)$ be $\tau[\vec{\tau}/\vec{\alpha}]$.

Remark It is easy to see that the image of a closing substitution is a set of closed terms.

Definition: (ciu equivalence for open terms) Suppose $\Gamma; \Delta \vdash_{\vec{\alpha}} M : \tau$ and $\Gamma; \Delta \vdash_{\vec{\alpha}} M' : \tau$, then we write † †

$$\Gamma; \Delta \vdash_{\vec{\alpha}} M =_{\mathrm{ciu}} M' : \tau$$

if and only if, for all $(\Gamma, \Delta, \vec{\alpha})$–closing substitutions σ

$$\sigma M =_{\mathrm{ciu}} \sigma M' : \tau[\vec{\sigma}/\vec{\alpha}].$$

Remark The relation $=_{\mathrm{ciu}}$ for open terms is clearly reflexive by the fact that $=_{\mathrm{ciu}}$ for closed terms is reflexive.

Definition: (Kleene equivalence) Two closed terms M, M' are *kleene equivalent* ($M =_{\mathrm{kl}} M' : \tau$) if and only if

$$\forall V. M \Downarrow_n V \Leftrightarrow M' \Downarrow_n V.$$

Lemma 5.17 *Given $\tau \in Typ$ and $M, M' \in Term(\tau)$. If $M =_{\mathrm{obs}} M' : \tau$ or $M =_{\mathrm{kl}} M' : \tau$ or $M =_{\mathrm{ciu}} M' : \tau$ or $M =_{\mathrm{gnd}} M : \tau$, then for all types $\tau' \in Typ$ and all test functions $V : \tau \multimap !\tau'$ for some $\tau' \in Typ$,*

$$VM \Downarrow_n \Leftrightarrow VM' \Downarrow_n .$$

And any $\top\top$ closed term relation r, respects $=_{\mathrm{kl}}, =_{\mathrm{obs}}, =_{\mathrm{gnd}}$ and $=_{\mathrm{ciu}}$.

Proof Let $M =_{\mathrm{obs}} M' : \tau$. Given $V : \tau \multimap !\tau'$, as $=_{\mathrm{obs}}$ contains $\mathcal{I}d$ we get from compatibility $VM =_{\mathrm{obs}} VM' : !\tau'$, whence by adequacy $VM \Downarrow_n \Leftrightarrow VM' \Downarrow_n$.

If $M =_{\mathrm{ciu}} M' : \tau$ then $M : \tau$ and $Vm[M/m] : !\tau'$, thus by Lemma 4.24 $VM =_{\mathrm{ciu}} VM' : !\tau'$. Hence for all types $\tau'' \in Typ$ and frame stacks $\mathcal{F}s : !\tau' \to !\tau''$, $\langle \mathcal{F}s, VM \rangle \searrow \Leftrightarrow \langle \mathcal{F}s, VM' \rangle \searrow$. If we let $\mathcal{F}s$ be $\mathcal{I}d$ then by Lemma 4.14, $\mathcal{I}d[VM] \Downarrow_n \Leftrightarrow \mathcal{I}d[VM'] \Downarrow_n$, thus $VM \Downarrow_n \Leftrightarrow VM' \Downarrow_n$.

Let $M =_{\mathrm{kl}} M' : \tau$, then for all values W, $M \Downarrow_n W \Leftrightarrow M' \Downarrow_n W$, thus by Lemma 4.13 $M =_{\mathrm{ciu}} V =_{\mathrm{ciu}} M' : \tau$.

As $=_{\mathrm{gnd}}$ for closed terms is equivalent to $=_{\mathrm{ciu}}$ (Lemma 4.25), we have the last case.

Given a term relation r, such that $r = r^{\top\top}$, assume $(M_1, M_1') \in r = r^{\top\top}$ and $M_1 =_{\mathrm{obs}} M$. Then

$$\forall (V, V') \in r^{\top}. VM \Downarrow_n \Leftrightarrow VM_1 \Downarrow_n \Leftrightarrow V' M_1' \Downarrow_n,$$

thus $(M, M_1') \in r^{\top\top} = r$. Likewise, if $M' =_{\mathrm{obs}} M_1' : \tau_1$ then $(M, M') \in r$. Hence

$$M_1 =_{\mathrm{obs}} M : \tau \wedge (M_1, M_1') \in r \wedge M_1' =_{\mathrm{obs}} M' : \tau_1 \Rightarrow (M, M') \in r.$$

The other equivalences ($=_{\mathrm{kl}}, =_{\mathrm{ciu}}, =_{\mathrm{gnd}}$) are proved in the same way as $=_{\mathrm{obs}}$. ∎

Lemma 5.18 $=_{\text{ciu}}$ *is substitutive.* *

Proof I will start by showing

$$\Gamma;\Delta \vdash_{\vec{\alpha},\alpha} M =_{\text{ciu}} M' : \tau \wedge ftv(\tau') \subseteq \vec{\alpha} \Rightarrow \Gamma[\tau'/\alpha];\Delta[\tau'/\alpha] \vdash_{\vec{\alpha}} M[\tau'/\alpha] =_{\text{ciu}} M[\tau'/\alpha] : \tau[\tau'/\alpha].$$

Given a $(\Gamma[\tau'/\alpha], \Delta[\tau'/\alpha], \vec{\alpha}, \tau[\tau'/\alpha])$–closing substitution σ, then σ must be defined by $\lambda t.t[\vec{\tau}, \vec{N'}, \vec{N}/\vec{\alpha}, \vec{x}, \vec{a}]$ for some $\vec{N}, \vec{N'}, \vec{\tau}$ such that $\vec{N'}, \vec{N} \subseteq (\tau[\tau'/\alpha])[\vec{\tau}/\vec{\alpha}]$.

Let σ' be $\lambda t.t[\vec{\tau}, \tau'[\vec{\tau}/\vec{\alpha}], \vec{N'}, \vec{N}/\vec{\alpha}, \alpha, \vec{x}, \vec{a}]$, then σ' is a $(\Gamma, \Delta, \vec{\alpha}, \alpha, \tau)$–closing substitution as $\text{dom}(\Gamma) = \text{dom}(\Gamma[\tau'/\alpha]), \text{dom}(\Delta) = \text{dom}(\Delta[\tau'/\alpha]), \tau[\tau'/\alpha] \in Typ$ and $(\tau[\tau'/\alpha])[\vec{\tau}/\vec{\alpha}] = \tau[\vec{\tau}, \tau'[\vec{\tau}/\vec{\alpha}]/\vec{\alpha}, \alpha]$. By assumption $\Gamma;\Delta \vdash_{\vec{\alpha}} M =_{\text{ciu}} M' : \tau$, thus $\sigma'(M) =_{\text{ciu}} \sigma'(M') : \sigma'(\tau)$. Now $\sigma'(M) = \sigma(M[\tau'/\alpha])$ and $\sigma'(M') = \sigma(M[\tau'/\alpha])$, thus $\sigma(M[\tau'/\alpha]) =_{\text{ciu}} \sigma(M'[\tau'/\alpha]) : \sigma(\tau[\tau'/\alpha])$. As σ was arbitrary we get the result.

Using the same reasoning where σ is extended with $a \to \sigma(N)$ gives the linear result (N is the term being substituted in). Likewise we get the intuitionistic result by extending with $x \to \sigma(N)$.

∎

Lemma 5.19 *

$$\Gamma;\Delta \vdash_{\vec{\alpha}} M \Delta M' : \tau \Rightarrow \Gamma;\Delta \vdash_{\vec{\alpha}} M =_{\text{obs}} M' : \tau$$

Proof To show $\Delta \subseteq =_{\text{obs}}$, we only need to show adequacy of Δ, since by Lemma 5.15, we then have adequacy and compatibility.

Assume $(M, M') \in \Delta_{!\tau}$ for some $\tau \in Typ$. It is enough to show $(\lambda x :!\tau.x, \lambda x :!\tau.x) \in (\Delta_{!\tau})^{\top} = (!\Delta_{\tau})^{\top\top^{\top}} = (!\Delta_{\tau})^{\top}$, since $(\lambda x :!\tau.x)M \Downarrow_n \Leftrightarrow M \Downarrow_n$. Now in $!\Delta_{\tau}$ there are only values $V :!\tau$, and $(\lambda x :!\tau.x)V \Downarrow_n \Leftrightarrow V \Downarrow_n$. Hence for all $(V, V') \in !\Delta_{\tau}$,

$$(\lambda x :!\tau.x)V \Downarrow_n \Leftrightarrow (\lambda x :!\tau.x)V' \Downarrow_n .$$

∎

Lemma 5.20 *

$$\Gamma;\Delta \vdash_{\vec{\alpha}} M \Delta M' : \tau \Leftrightarrow \Gamma;\Delta \vdash_{\vec{\alpha}} M =_{\text{ciu}} M' : \tau$$

Proof First $\Delta \subseteq =_{\text{ciu}}$. Given a $(\Gamma, \Delta, \vec{\alpha})$–closing substitution $\sigma : \tau \to \tau[\vec{\tau}/\vec{\alpha}]$, we must show for all frame stacks $\mathcal{F}s : \tau[\vec{\tau}/\vec{\alpha}] \multimap !\tau'$, $\langle \mathcal{F}s, \sigma M\rangle \searrow \Leftrightarrow \langle \mathcal{F}s, \sigma M'\rangle \searrow$.

Assume $\Gamma;\Delta \vdash_{\vec{\alpha}} M \Delta M' : \tau$ then using the definition of Δ on open terms where $r_i = \Delta_{\tau_i}$, and the fact that Δ_{τ_i} is reflexive, we get $(\sigma M, \sigma M') \in \Delta_{\tau}(\vec{r}/\vec{\alpha})$. By Lemma 5.14

$$(\sigma M, \sigma M') \in \Delta_{\tau[\vec{\tau}/\vec{\alpha}]}.$$

Thus $\sigma M =_{\text{obs}} \sigma M' : \tau[\vec{\tau}/\vec{\alpha}]$ and therefore by compatibility $\mathcal{F}s[\sigma M] =_{\text{obs}} \mathcal{F}s[\sigma M'] :!\tau'$. By adequacy $\mathcal{F}s[\sigma M] \Downarrow_n \Leftrightarrow \mathcal{F}s[\sigma M'] \Downarrow_n$, and by Lemma 4.14, $\langle \mathcal{F}s, \sigma M\rangle \searrow \Leftrightarrow \langle \mathcal{F}s, \sigma M'\rangle \searrow$.

Now to $=_{\text{ciu}} \subseteq \Delta$. By Lemma 5.18, $=_{\text{ciu}}$ is substitutive, thus by Lemma 5.10 and Lemma 5.17

$$\Gamma;\Delta \vdash_{\vec{\alpha}} M_1 =_{\text{ciu}} M_1' : \tau \wedge \Gamma;\Delta \vdash_{\vec{\alpha}} M_1' \Delta M_2' : \tau \wedge \Gamma;\Delta \vdash_{\vec{\alpha}} M_2 =_{\text{ciu}} M_2' : \tau \Rightarrow \Gamma;\Delta \vdash_{\vec{\alpha}} M_1 \Delta M_2 : \tau.$$

As both Δ and $=_{\text{ciu}}$ are reflexive

$$\Gamma;\Delta \vdash_{\vec{\alpha}} M_1 =_{\text{ciu}} M_1' : \tau \Rightarrow \Gamma;\Delta \vdash_{\vec{\alpha}} M_1 \Delta M_1' : \tau.$$

∎

The next corollaries are all stated in [BPR00] and a single equation in Corollary 5.27 is proved there.

Corollary 5.21 *Kleene equivalence is contained in ground contextual equivalence. Thus*

$$((\lambda a:\tau.M)N, M[N/a]) \in \Delta_{\tau'}$$
$$((\Lambda\alpha.M)\sigma, M[\sigma/\alpha]) \in \Delta_{\tau[\sigma/\alpha]}$$
$$(\mathbf{let}\ !y = !(x = N:\tau)\ \mathbf{in}\ M, M[(\mathbf{fix}\ x:\tau.N)/y]) \in \Delta_{\tau'}$$
$$(\mathbf{let}\ * = *\ \mathbf{in}\ M, M) \in \Delta_\tau$$
$$(\mathbf{let}\ a_1 \otimes a_2 = M_1 \otimes M_2\ \mathbf{in}\ M_3, M_3[M_1, M_2/a_1, a_2]) \in \Delta_\tau$$
$$(\mathbf{fix}\ x:\tau.M, M) \in \Delta_\tau \qquad x \text{ not free in } M$$

and divergent terms of equal type are all contextual equivalent.

Proof From Lemma 5.17 we know $\Delta_{\tau'}$ respects kleene equivalence. As kleene equivalence and $\Delta_{\tau'}$ are both reflexive, $=_{\mathrm{kl}} \subseteq \Delta$ (like in the proof of Theorem 5.20). The equations are clearly kleene equivalent and divergent terms are kleene equivalent. ∎

Lemma 5.22 *

$$(\mathbf{let}\ !y = !(x = M_1:\tau)\ \mathbf{in}\ M_2, M_2[M_1/y]) \in \Delta_{\tau'} \qquad x \text{ not free in } M_1$$
$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{let}\ !y = !(x = M_1:\tau)\ \mathbf{in}\ M_2 \Delta M_2[M_1/y] : \tau' \qquad x \text{ not free in } M_1$$

Proof The first is given by the third and sixth equation above, compatibility and transitivity. Ciu equality then gives the second. ∎

Theorem 5.23 *All the equivalences given in Corollary 5.21 works with open terms and types, thus* *

$$\Gamma;\Delta \vdash_{\vec{\alpha}} (\lambda a:\tau.M)N \Delta M[N/a] : \tau'$$
$$\Gamma;\Delta \vdash_{\vec{\alpha}} (\Lambda\alpha.M)\sigma \Delta M[\sigma/\alpha] : \tau[\sigma/\alpha]$$
$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{let}\ !y = !(x = N:\tau)\ \mathbf{in}\ M \Delta M[(\mathbf{fix}\ x:\tau.N)/y] : \tau'$$
$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{let}\ * = *\ \mathbf{in}\ M \Delta M : \tau$$
$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{let}\ a_1 \otimes a_2 = M_1 \otimes M_2\ \mathbf{in}\ M_3 \Delta M_3[M_1, M_2/a_1, a_2] : \tau$$
$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{fix}\ x:\alpha.M \Delta M \qquad x \text{ not free in } M.$$

Proof Using the ciu correspondence with Δ (Lemma 5.20) we get the result by Corollary 5.21. Eg. Given a $(\Gamma, \Delta, \vec{\alpha})$–closing substitution $\sigma : \tau' \to \tau'[\vec{\tau}/\vec{\alpha}]$,

$$\sigma((\lambda a:\tau.M)N) = (\lambda a:\tau[\vec{\tau}/\vec{\alpha}].\sigma(M))\sigma(N) =_{\mathrm{ciu}} \sigma(M)[\sigma(N)/a] = \sigma(M[N/a]) : \tau'[\vec{\tau}/\vec{\alpha}].$$

Lemma 5.24 $=_{\mathrm{obs}}$ *is substitutive.* *

Proof Given $\Gamma;\Delta \vdash_{\vec{\alpha},\alpha} M =_{\mathrm{obs}} M' : \tau$ and a type τ' such that $ftv(\tau') \subseteq \vec{\alpha}$, I will show

$$\Gamma[\tau'/\alpha];\Delta[\tau'/\alpha] \vdash_{\vec{\alpha}} M[\tau'/\alpha] =_{\mathrm{obs}} M'[\tau'/\alpha] : \tau[\tau'/\alpha].$$

If $\Delta = \Delta', a:\sigma$, then by compatibility

$$\Gamma;\Delta' \vdash_{\vec{\alpha},\alpha} \lambda a:\sigma.M =_{\mathrm{obs}} \lambda a:\sigma.M' : \sigma \multimap \tau.$$

Continuing in this manner we get

$$\Gamma;\emptyset \vdash_{\vec{\alpha},\alpha} \lambda a_n : \sigma_n. \ldots . \lambda a_1 : \sigma_1.M =_{\text{obs}} \lambda a_n : \sigma_n. \ldots . \lambda a_1 : \sigma_1.M' : \sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau.$$

Let $\lambda a_n : \sigma_n. \ldots . \lambda a_1 : \sigma_1.M = N$ and $\lambda a_n : \sigma_n. \ldots . \lambda a_1 : \sigma_1.M' = N'$.
If $\Gamma = \Gamma', x : \sigma'$ then

$$\Gamma';\emptyset \vdash_{\vec{\alpha},\alpha} \lambda a :!\sigma'.\mathbf{let}\ !x = a\ \mathbf{in}\ N =_{\text{obs}} \lambda a :!\sigma'.\mathbf{let}\ !x = a\ \mathbf{in}\ N' : \sigma'_1 \multimap \sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau.$$

Continuing as before we get for some K, K'

$$\begin{gathered}\emptyset;\emptyset \vdash_{\vec{\alpha},\alpha} K =_{\text{obs}} K' : \sigma'_m \multimap \cdots \multimap \sigma'_1 \multimap \sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau \\ K_m = \lambda a_m :!\sigma'_m.\mathbf{let}\ !x_m = a_m\ \mathbf{in}\ \ldots \lambda a_1 :!\sigma'_1.\mathbf{let}\ !x_1 = a_1\ \mathbf{in}\ N \\ K'_m = \lambda a_m :!\sigma'_m.\mathbf{let}\ !x_m = a_m\ \mathbf{in}\ \ldots \lambda a_1 :!\sigma'_1.\mathbf{let}\ !x_1 = a_1\ \mathbf{in}\ N'\end{gathered}$$

By compatibility

$$\emptyset;\emptyset \vdash_{\vec{\alpha}} (\Lambda\alpha.K_m)\tau' =_{\text{obs}} (\Lambda\alpha.K'_m)\tau' : (\sigma'_m \multimap \cdots \multimap \sigma'_1 \multimap \sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau)[\tau'/\alpha],$$

and by Lemma 5.23 and Lemma 5.19

$$\begin{gathered}\emptyset;\emptyset \vdash_{\vec{\alpha}} (\Lambda\alpha.K_m)\tau' =_{\text{obs}} K_m[\tau'/\alpha] : (\sigma'_m \multimap \cdots \multimap \sigma'_1 \multimap \sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau)[\tau'/\alpha] \\ \emptyset;\emptyset \vdash_{\vec{\alpha}} (\Lambda\alpha.K'_m)\tau' =_{\text{obs}} K'_m[\tau'/\alpha] : (\sigma'_m \multimap \cdots \multimap \sigma'_1 \multimap \sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau)[\tau'/\alpha],\end{gathered}$$

thus by transitivity

$$\emptyset;\emptyset \vdash_{\vec{\alpha}} K_m[\tau'/\alpha] =_{\text{obs}} K'_m[\tau'/\alpha] : (\sigma'_m \multimap \cdots \multimap \sigma'_1 \multimap \sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau)[\tau'/\alpha].$$

By compatibility

$$\begin{aligned}x_m : \sigma'_m[\tau'/\alpha];\emptyset \vdash_{\vec{\alpha},\alpha} (K_m!x_m)[\tau'/\alpha] =_{\text{obs}} (K'_m!x_m)[\tau'/\alpha] :& \\ (\sigma'_{m-1} \multimap \cdots \multimap \sigma'_1 \multimap \sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau)[\tau'/\alpha]&\end{aligned}$$

and by Lemma 5.23 and Lemma 5.19

$$\begin{gathered}x : \sigma'_m[\tau'/\alpha];\emptyset \vdash_{\vec{\alpha}} K_m!x =_{\text{obs}} K_{m-1} : (\sigma'_{m-1} \multimap \cdots \multimap \sigma'_1 \multimap \sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau)[\tau'/\alpha] \\ x : \sigma'_m[\tau'/\alpha];\emptyset \vdash_{\vec{\alpha}} K'_m!x =_{\text{obs}} K'_{m-1} : (\sigma'_{m-1} \multimap \cdots \multimap \sigma'_1 \multimap \sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau)[\tau'/\alpha]\end{gathered}$$

thus by transitivity

$$\begin{aligned}x_m : \sigma'_m[\tau'/\alpha];\emptyset \vdash_{\vec{\alpha},\alpha} K_{m-1}[\tau'/\alpha] =_{\text{obs}} K'_{m-1}[\tau'/\alpha] :& \\ (\sigma'_{m-1} \multimap \cdots \multimap \sigma'_1 \multimap \sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau)[\tau'/\alpha].&\end{aligned}$$

Continuing in this manner we get

$$\Gamma[\tau'/\alpha];\emptyset \vdash_{\vec{\alpha}} N[\tau/\alpha] =_{\text{obs}} N'[\tau'/\alpha] : (\sigma_n \multimap \cdots \multimap \sigma_1 \multimap \tau)[\tau'/\alpha].$$

Doing the same with the linear variables we get

$$\Gamma[\tau'/\alpha];\Delta[\tau'/\alpha] \vdash_{\vec{\alpha}} M[\tau/\alpha] =_{\text{obs}} M'[\tau'/\alpha] : \tau[\tau'/\alpha].$$

Next I will prove that given $\Gamma, x : \tau;\Delta \vdash_{\vec{\alpha}} M =_{\text{obs}} M' : \tau'$ and a term $\Gamma;\Delta \vdash_{\vec{\alpha}} N : \tau$ then

$$\Gamma;\Delta \vdash_{\vec{\alpha}} M[N/x] =_{\text{obs}} M'[N/x] : \tau'.$$

By compatibility,

$$\Gamma;\Delta \vdash_{\vec{\alpha}} \lambda a :!\tau.\mathbf{let}\ !x = a\ \mathbf{in}\ M =_{\mathrm{obs}} \lambda a :!\tau.\mathbf{let}\ !x = a\ \mathbf{in}\ M' : \tau',$$

thus

$$\Gamma;\Delta \vdash_{\vec{\alpha}} (\lambda a :!\tau.\mathbf{let}\ !x = a\ \mathbf{in}\ M)!N =_{\mathrm{obs}} (\lambda a :!\tau.\mathbf{let}\ !x = a\ \mathbf{in}\ M')!N : \tau'.$$

By Lemma 5.23 and Lemma 5.19

$$\Gamma;\Delta \vdash_{\vec{\alpha}} (\lambda a :!\tau.\mathbf{let}\ !x = a\ \mathbf{in}\ M)!N =_{\mathrm{obs}} M[N/x] : \tau'$$
$$\Gamma;\Delta \vdash_{\vec{\alpha}} (\lambda a :!\tau.\mathbf{let}\ !x = a\ \mathbf{in}\ M')!N =_{\mathrm{obs}} M'[N/x] : \tau',$$

thus

$$\Gamma;\Delta \vdash_{\vec{\alpha}} M[N/x] =_{\mathrm{obs}} M'[N/x] : \tau'.$$

The same reasoning gives the linear result. ∎

Lemma 5.25 $\Gamma;\Delta \vdash_{\vec{\alpha}} M =_{\mathrm{obs}} M' : \tau \Rightarrow \Gamma;\Delta \vdash_{\vec{\alpha}} M =_{\mathrm{ciu}} M' : \tau.$ *

Proof Given $\Gamma;\Delta \vdash_{\vec{\alpha}} M =_{\mathrm{obs}} M' : \tau$ and a $(\Gamma,\Delta,\vec{\alpha})$–closing substitution $\sigma : \tau \to \tau[\vec{\tau}/\vec{\alpha}]$, then by Lemma 5.24, $\sigma M =_{\mathrm{obs}} \sigma M' : \tau[\vec{\tau}/\vec{\alpha}]$. Given a closed frame stack $\mathcal{F}s : \tau[\vec{\tau}/\vec{\alpha}] \multimap !\tau'$ for some type $\tau' \in Typ$, then by compatibility $\emptyset;\emptyset \vdash_{\emptyset} \mathcal{F}s[\sigma M] =_{\mathrm{obs}} \mathcal{F}s[\sigma M'] :!\tau'$ and by adequacy $\mathcal{F}s[\sigma M] \Downarrow_n \Leftrightarrow \mathcal{F}s[\sigma M'] \Downarrow_n$, thus by Lemma 4.14 $\sigma M, \sigma M'$ are ciu equivalent. Hence as the $(\Gamma,\Delta,\vec{\alpha})$–closing substitution was arbitrary, M, M' are ciu equivalent.

Theorem 5.26 *We have*

$$\Gamma;\Delta \vdash_{\vec{\alpha}} M =_{\mathrm{obs}} M' : \tau \Leftrightarrow \Gamma;\Delta \vdash_{\vec{\alpha}} M \Delta M' : \tau \Leftrightarrow \Gamma;\Delta \vdash_{\vec{\alpha}} M_1 =_{\mathrm{ciu}} M_1' : \tau.$$

Proof Combining Lemma 5.25, Lemma 5.20 and Lemma 5.19. ∎

Corollary 5.27 (Extensionality properties of $=_{\mathbf{gnd}}$)

$$\begin{aligned}
M =_{\mathrm{gnd}} M' : \tau \multimap \tau' &\Leftrightarrow \forall N \in Term(\tau).MN =_{\mathrm{gnd}} M'N : \tau' \\
M =_{\mathrm{gnd}} M' : \tau \multimap \tau' &\Leftrightarrow \forall V. \in Val(\tau).MV =_{\mathrm{gnd}} M'V : \tau' \\
M =_{\mathrm{gnd}} M' : \forall\alpha.\tau &\Leftrightarrow \forall \sigma \in Typ.M\sigma =_{\mathrm{gnd}} M'\sigma : \tau[\sigma/\alpha] \qquad (5.11)\\
M =_{\mathrm{gnd}} M' :!\tau &\Leftrightarrow (M\not\Downarrow_! \wedge M'\not\Downarrow_!) \vee \exists x, N, x', N'. \\
&\qquad M \Downarrow_s !(x = N : \tau) \wedge \\
&\qquad M' \Downarrow_s !(x' = N' : \tau) \\
&\qquad \mathbf{fix}\ x : \tau.N =_{\mathrm{gnd}} \mathbf{fix}\ x' : \tau.N' : \tau
\end{aligned}$$

Proof Starting from the top.

Assume $M =_{\mathrm{gnd}} M' : \tau \multimap \tau'$, then $(M, M') \in \Delta_{\tau\multimap\tau'}$. Thus for all $(N, N') \in \Delta_\tau$, $(MN, M'N') \in \Delta_{\tau'}$. As Δ is reflexive, $\forall N \in Term(\tau).(MN, M'N) \in \Delta_{\tau'}$.

Assume $\forall N \in Term(\tau).(MN, M'N) \in \Delta_{\tau'}$. Given $(N, N') \in \Delta_\tau$ then $(MN, M'N) \in \Delta_{\tau'}$, and as $\Delta_{\tau\multimap\tau'}$ is reflexive $(M', M') \in \Delta_{\tau\multimap\tau'}$, thus $(M'N, M'N') \in \Delta_{\tau'}$, whence by transitivity $(MN, M'N') \in \Delta_{\tau'}$. As (N, N') was arbitrary we have $(M, M') \in \Delta_{\tau\multimap\tau'}$, and $M =_{\mathrm{gnd}} M' : \tau \multimap \tau'$.

As $Val(\tau) \subseteq Term(\tau)$, we get the first $\Rightarrow$. Given $N \in Term(\tau)$ then as $\Delta_{\tau'} = \Delta_{\tau'}^{\top\top}$,

$$(MN, M'N) \in \Delta_{\tau'} \Leftrightarrow \big(\forall (W, W') \in \Delta_{\tau'}^{\top}.W(MN) \Downarrow_n \Leftrightarrow W'(M'N) \Downarrow_n\big)$$

thus given $(W, W') \in \Delta^{\top}_{\tau'}$ we must show $\star$,

$$N \Downarrow_s V \wedge W(MV) \Downarrow_n \Leftrightarrow W(MN) \Downarrow_n \stackrel{\star}{\Leftrightarrow} W'(M'N) \Downarrow_n \Leftrightarrow N \Downarrow_s V \wedge W'(M'V) \Downarrow_n \text{ for some } V$$

Note that by determinism the V's are equal. If $N \not\Downarrow_s$ then the result is obvious. Otherwise we get the result by the assumption $(MV, M'V) \in \Delta_{\tau'}$.

The $\Rightarrow$ in equation (5.11) follows from compatibility. Assume $\forall \sigma \in Typ. M\sigma =_{\text{gnd}} M'\sigma : \tau[\sigma/\alpha]$. Given types $\sigma', \sigma'' \in Typ$ and a relation $r \in Rel(\sigma', \sigma'')$ we must show

$$(M\sigma', M'\sigma'') \in \Delta_\tau(r^{\top\top}/\alpha)$$

Now $(M, M) \in \Delta_{\forall\alpha.\tau} = \forall r. \Delta_\tau(r^{\top\top}/\alpha)$, thus $(M\sigma', M\sigma'') \in \Delta_\tau(r^{\top\top}/\alpha)$. By Lemma 5.10

$$\Delta_\tau(r^{\top\top}/\alpha) = (\Delta_\tau(r^{\top\top}/\alpha))^{\top\top}.$$

Given $(V, V') \in (\Delta_\tau(r^{\top\top}/\alpha))^{\top}$, then by contextual equivalence

$$V'(M\sigma'') \Downarrow_n \Leftrightarrow V'(M'\sigma'') \Downarrow_n,$$

thus

$$V(M\sigma') \Downarrow_n \Leftrightarrow V'(M\sigma'') \Downarrow_n \Leftrightarrow V'(M'\sigma'') \Downarrow_n .$$

Hence

$$(M\sigma', M'\sigma'') \in (\Delta_\tau(r^{\top\top}/\alpha))^{\top\top} = \Delta_\tau(r^{\top\top}/\alpha).$$

Assume $M =_{\text{gnd}} M' :!\tau$. Either $M \Downarrow_s$ or not. In the later case then as $\Delta_{!\tau}$ is adequate $M' \not\Downarrow_s$.

In the former case then by adequacy $M' \Downarrow_s$ as well. Assume $M \Downarrow_s !(x = N : \tau), M' \Downarrow_s !(x' = N' : \tau)$ for some x, x', N, N'. As $(M, M') \in \Delta_{!\tau}$,

$$\forall (V, V') \in \Delta^{\top}_{!\tau}. VM \Downarrow_n \Leftrightarrow V'M' \Downarrow_n,$$

thus by the strictness theorem and determinism of Lily.

$$\forall (V, V') \in \Delta^{\top}_{!\tau}. V!(x = N : \tau) \Downarrow_n \wedge M \Downarrow_s !(x = N : \tau) \Leftrightarrow \\ V!(x = N' : \tau) \Downarrow_n \wedge M' \Downarrow_s !(x' = N' : \tau)$$

Hence $(!(x = N : \tau), !(x' = N' : \tau)) \in \Delta^{\top\top}_{!\tau} = \Delta_{!\tau}$. Let $E = \lambda a :!\tau. \mathbf{let}\ !y = a\ \mathbf{in}\ y :!\tau \multimap \tau$. By compatibility $E!(x = N : \tau) =_{\text{gnd}} E!(x' = N' : \tau) : \tau$, thus $\mathbf{fix}\ x : \tau. N =_{\text{gnd}} \mathbf{fix}\ x' : \tau. N' : \tau$ as $E!(x = N : \tau) =_{\text{gnd}} \mathbf{fix}\ x : \tau. N$.

Given $(V, V') \in \Delta^{\top}_{!\tau} = (!\Delta_\tau)^{\top\top\top}$, then by the strictness theorem

$$VM \Downarrow_n \Leftrightarrow M \Downarrow_s !(x = N : \tau) \wedge V!(x = N : \tau) \Downarrow_n \text{ for some } x, N$$
$$V'M' \Downarrow_n \Leftrightarrow M' \Downarrow_s !(x' = N' : \tau) \wedge V'!(x' = N' : \tau) \Downarrow_n \text{ for some } x', N'$$

If $M \not\Downarrow_s$ and $M' \not\Downarrow_s$ then $VM \Downarrow_n \Leftrightarrow V'M' \Downarrow_n$, thus $(M'M') \in \Delta^{\top\top}_{!\tau} = \Delta_{!\tau}$.

Assume there exists x, x', N, N', such that $\mathbf{fix}\ x : \tau. N =_{\text{gnd}} \mathbf{fix}\ x' : \tau. N' : \tau$, $M \Downarrow_s !(x = N : \tau)$ and $M' \Downarrow_s !(x' = N' : \tau)$, then

$$(!(x = N : \tau), !(x' = N' : \tau)) \in (!\Delta_\tau).$$

and by determinism

$$VM \Downarrow_n \Leftrightarrow M \Downarrow_s !(x = N : \tau) \wedge V!(x = N : \tau) \Downarrow_n$$
$$V'M' \Downarrow_n \Leftrightarrow M' \Downarrow_s !(x' = N' : \tau) \wedge V'!(x' = N' : \tau) \Downarrow_n$$

and as $(!\Delta_\tau)^{\top\top\top} = (!\Delta_\tau)^{\top}$

$$V!(x = N : \tau) \Downarrow_n \Leftrightarrow V'!(x' = N' : \tau) \Downarrow_n .$$

Hence $(M, M') \in \Delta^{\top\top}_{!\tau} = \Delta_{!\tau}$.

∎

Lemma 5.28 *We have* *

$$\mathbf{let}\ !x = M\ \mathbf{in}\ !(y = x : \tau) =_{\mathrm{gnd}} M :!\tau \quad \textit{where } x \neq y$$
$$\Gamma; \Delta \vdash_{\vec{\alpha}} \mathbf{let}\ !x = M\ \mathbf{in}\ !(y = x : \tau) =_{\mathrm{gnd}} M :!\tau \quad \textit{where } x \neq y$$

Proof If $M \not\Downarrow!$ then $\mathbf{let}\ !x = M\ \mathbf{in}\ !x \not\Downarrow_!$, thus by Corollary 5.27 we get the result.
Assume $M \Downarrow_s !(z = M' : \tau)$ then as

$$\mathbf{let}\ !x = M\ \mathbf{in}\ !(y = x : \tau) \Downarrow_s !(y = \mathbf{fix}\ z : \tau.M' : \tau)$$

by Corollary 5.27, we only have to show

$$\mathbf{fix}\ y : \tau.\mathbf{fix}\ z : \tau.M' =_{\mathrm{gnd}} \mathbf{fix}\ z : \tau.M' : \tau$$

By Corollary 5.21 and Lemma 5.26 we get the result.

Given a $(\Gamma, \Delta, \vec{\alpha})$–closing substitution $\sigma : \tau \to \tau[\vec{\tau}/\vec{\alpha}]$, then as $\sigma(\mathbf{let}\ !x = M\ \mathbf{in}\ !(y = x : \tau)) = \mathbf{let}\ !x = \sigma M\ \mathbf{in}\ !(y = x : \tau[\vec{\tau}/\vec{\alpha}])$, by the first equation $(\sigma M, \sigma(\mathbf{let}\ !x = M\ \mathbf{in}\ !(y = x : \tau))$ are contextual equivalent, thus we have the result. ∎

Lemma 5.29 (η equivalence) *We have* †

$$\Gamma; \Delta \vdash_{\vec{\alpha}} \lambda a : \tau.ga =_{\mathrm{gnd}} g : \tau \multimap \tau'$$
$$\Gamma; \Delta \vdash_{\vec{\alpha}} \Lambda\alpha.g\alpha =_{\mathrm{gnd}} g : \forall\alpha.\tau \qquad \alpha \textit{ not free in } g$$

Proof To begin I will start with the closed case.

Given a value $V \in Val(\tau)$, by the type system a is not free in g, thus $(\lambda a : \tau.ga)V =_{\mathrm{gnd}} gV$. Thus

$$\forall V \in Val(\tau).(\lambda a : \tau.ga)V =_{\mathrm{gnd}} gV : \tau'.$$

By Corollary 5.27 we get the result.

Given a $(\Gamma, \Delta, \vec{\alpha})$–closing substitution $\sigma : \tau \multimap \tau' \to (\tau \multimap \tau')[\vec{\tau}/\vec{\alpha}]$,

$$\sigma(\lambda a : \tau.ga) = \lambda a : \tau[\vec{\tau}/\vec{\alpha}].\sigma(g)a =_{\mathrm{gnd}} \sigma(g) : (\tau \multimap \tau')[\vec{\tau}/\vec{\alpha}].$$

Thus as the closing substitution was arbitrary we get the result for open terms.

The type abstraction equation is proved the same way. ∎

Corollary 5.30 *Any recursively defined thunk,* $!(x = M : \tau) \in Val(!\tau)$ *is contextual equivalent to a non-recursive thunk of a fix point term* $!(\mathbf{fix}\ x : \tau.M) :!\tau$.

Proof By Corollary 5.27 we only have to show

$$\mathbf{fix}\ y : \tau.\mathbf{fix}\ x : \tau.M =_{\mathrm{gnd}} \mathbf{fix}\ x : \tau.M : \tau \quad y \text{ not free in } M.$$

By Corollary 5.21 we get the result. ∎

5.3 The tensor product

Above I have introduced the syntactical structure of a tensor product and unit. To justify the names we must show that they behave as such. In [Bar96] the requirements are quantified as the following theorem.

Theorem 5.31

$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{let} * = * \mathbf{in}\ M =_{\mathrm{gnd}} M : \tau \tag{5.12}$$

$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{let} * = M \mathbf{in}\ * =_{\mathrm{gnd}} M : I \tag{5.13}$$

$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{let}\ a_1 \otimes a_2 = M_1 \otimes M_2 \mathbf{in}\ M_3 =_{\mathrm{gnd}} M_3[M_1, M_2/a_1, a_2] : \tau \tag{5.14}$$

$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{let}\ a_1 \otimes a_2 = M \mathbf{in}\ a_1 \otimes a_2 =_{\mathrm{gnd}} M : \tau_1 \otimes \tau_2 \tag{5.15}$$

For all linear contexts C

$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{let} * = M_1 \mathbf{in}\ C[M_2] =_{\mathrm{gnd}} C[\mathbf{let} * = M_1 \mathbf{in}\ M_2] : \tau.$$

A linear contexts is a context where the hole is not inside a thunk. This ensures that the strictness theorem applies.

For all linear contexts C which do not bind $a_1 : \tau_1, a_2 : \tau_2$.

$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{let}\ a_1 \otimes a_2 = M_1 \mathbf{in}\ C[M_2] =_{\mathrm{gnd}} C[\mathbf{let}\ a_1 \otimes a_2 = M_1 \mathbf{in}\ M_2] : \tau$$

For all linear contexts C which do not bind $x : \tau'$.

$$\Gamma;\Delta \vdash_{\vec{\alpha}} \mathbf{let}\ !x = M_1 \mathbf{in}\ C[M_2] =_{\mathrm{gnd}} C[\mathbf{let}\ !x = M_1 \mathbf{in}\ M_2] : \tau$$

Proof By the ciu theorem we only have to consider closed terms. In (5.12), the left side is kleene equivalent to the right side, thus they are contextual equivalent. Likewise with (5.13), (5.14) and (5.15).

As Δ is $\top\top$ closed the first equality is equivalent to

$$(\mathbf{let} * = M_1 \mathbf{in}\ C[M_2], C[\mathbf{let} * = M_1 \mathbf{in}\ M_2]) \in \Delta_\tau^{\top\top}$$

which is equivalent to

$$\forall (V, V') \in \Delta_\tau^{\top}. V(\mathbf{let} * = M_1 \mathbf{in}\ C[M_2]) \Downarrow_n \Leftrightarrow V'(C[\mathbf{let} * = M_1 \mathbf{in}\ M_2]) \Downarrow_n$$

Given $(V, V') \in \Delta_\tau^{\top}$

$$\begin{aligned} V(\mathbf{let} * = M_1 \mathbf{in}\ C[M_2]) \Downarrow_n &\Leftrightarrow M_1 \Downarrow_s * \wedge V(\mathbf{let} * = * \mathbf{in}\ C[M_2]) \Downarrow_n \\ &\Leftrightarrow M_1 \Downarrow_s * \wedge M_2 \Downarrow_s V_2 \wedge V(\mathbf{let} * = * \mathbf{in}\ C[V_2]) \Downarrow_n \text{ for some } V_2 \\ &\Leftrightarrow \mathbf{let} * = M_1 \mathbf{in}\ M_2 \Downarrow_s V_2 \wedge V(C[V_2]) \Downarrow_n \text{ for some } V_2 \\ &\Leftrightarrow V(C[\mathbf{let} * = M_1 \mathbf{in}\ M_2]) \Downarrow_n \end{aligned}$$

then as Δ_τ is reflexive we have the result.

Next we have to show

$$\forall (V, V') \in \Delta_\tau^{\top}. V(\mathbf{let}\ a_1 \otimes a_2 = M_1 \mathbf{in}\ C[M_2]) \Downarrow_n \Leftrightarrow V'(C[\mathbf{let}\ a_1 \otimes a_2 = M_1 \mathbf{in}\ M_2]) \Downarrow_n .$$

Given $(V, V') \in \Delta_\tau^{\top}$,

$$\begin{aligned} & V(\mathbf{let}\ a_1 \otimes a_2 = M_1 \mathbf{in}\ C[M_2]) \Downarrow_n \\ \Leftrightarrow & M_1 \Downarrow_s N_1 \otimes N_2 \wedge V(\mathbf{let}\ a_1 \otimes a_2 = N_1 \otimes N_2 \mathbf{in}\ C[M_2]) \Downarrow_n \text{ for some } N_1, N_2 \\ \Leftrightarrow & M_1 \Downarrow_s N_1 \otimes N_2 \wedge \mathbf{let}\ a_1 \otimes a_2 = N_1 \otimes N_2 \mathbf{in}\ C[M_2] \Downarrow_s V_2 \wedge VV_2 \Downarrow_n \text{ for some } N_1, N_2, V_2 \\ \Leftrightarrow & M_1 \Downarrow_s N_1 \otimes N_2 \wedge C[M_2][N_1, N_2/a_1, a_2] \Downarrow_s V_2 \wedge VV_2 \Downarrow_n \text{ for some } N_1, N_2, V_2 \\ \Leftrightarrow & M_1 \Downarrow_s N_1 \otimes N_2 \wedge C[M_2[N_1, N_2/a_1, a_2]] \Downarrow_s V_2 \wedge VV_2 \Downarrow_n \text{ for some } N_1, N_2, V_2 \\ \Leftrightarrow & M_1 \Downarrow_s N_1 \otimes N_2 \wedge V(C[M_2[N_1, N_2/a_1, a_2]]) \Downarrow_n \text{ for some } N_1, N_2 \\ \Leftrightarrow & M_1 \Downarrow_s N_1 \otimes N_2 \wedge M_2[N_1, N_2/a_1, a_2] \Downarrow_s V_3 \wedge VC[V_3] \Downarrow_n \text{ for some } N_1, N_2, V_3 \\ \Leftrightarrow & \mathbf{let}\ a_1 \otimes a_2 = M_1 \mathbf{in}\ M_2 \Downarrow_s V_3 \wedge VC[V_3] \Downarrow_n \text{ for some } V_3 \\ \Leftrightarrow & VC[\mathbf{let}\ a_1 \otimes a_2 = M_1 \mathbf{in}\ M_2] \Downarrow_n . \end{aligned}$$

then as Δ_τ is reflexive we have the result.

Next we have to show

$$\forall (V, V') \in \Delta_\tau^\top . V(\textbf{let } !x = M_1 \textbf{ in } C[M_2]) \Downarrow_n \Leftrightarrow V'(C[\textbf{let } !x = M_1 \textbf{ in } M_2]) \Downarrow_n$$

Given $(V, V') \in \Delta_\tau^\top$,

$$\begin{aligned}
&V(\textbf{let } !x = M_1 \textbf{ in } C[M_2]) \Downarrow_n \\
\Leftrightarrow &M_1 \Downarrow_s V_1 \wedge V(\textbf{let } !x = V_1 \textbf{ in } C[M_2]) \Downarrow_n \text{ for some } V_1 \\
\Leftrightarrow &M_1 \Downarrow_s !(y = M_1' : \tau') \wedge \textbf{let } !x = !(y = M_1' : \tau') \textbf{ in } C[M_2] \Downarrow_s V_2 \wedge VV_2 \Downarrow_n \text{ for some } M_1', V_2 \\
\Leftrightarrow &M_1 \Downarrow_s !(y = M_1' : \tau') \wedge (C[M_2])[\textbf{fix } y : \tau'.M_1'/x] \Downarrow_s V_2 \wedge VV_2 \Downarrow_n \text{ for some } M_1', V_2 \\
\Leftrightarrow &M_1 \Downarrow_s !(y = M_1' : \tau') \wedge C[M_2[\textbf{fix } y : \tau'.M_1'/x]] \Downarrow_s V_2 \wedge VV_2 \Downarrow_n \text{ for some } M_1', V_2 \\
\Leftrightarrow &M_1 \Downarrow_s !(y = M_1' : \tau') \wedge V(C[M_2[\textbf{fix } y : \tau'.M_1'/x]]) \Downarrow_n \text{ for some } M_1' \\
\Leftrightarrow &M_1 \Downarrow_s !(y = M_1' : \tau') \wedge M_2[\textbf{fix } y : \tau'.M_1'/x] \Downarrow_s V_3 \wedge V(C[V_3]) \Downarrow_n \text{ for some } M_1', V_3 \\
\Leftrightarrow &\textbf{let } !x = M_1 \textbf{ in } M_2 \Downarrow_s V_3 \wedge V(C[V_3]) \Downarrow_n \text{ for some } V_3 \\
\Leftrightarrow &V(C[\textbf{let } !x = M_1 \textbf{ in } M_2]) \Downarrow_n
\end{aligned}$$

then as Δ_τ is reflexive we have the result. ∎

5.4 The graph relation on linear functions

The next lemma is mentioned in [BPR00] as it is needed to show interesting properties of functions; like uniqueness modulo contextual equivalence, etc. An example is given in section 6.1.

The lemma is quite effective, as it proves that any function of type $\tau \multimap \tau'$ gives a $\top\top$ closed relation in $Rel(\tau, \tau')$.

Lemma 5.32 *For any linear function $G \in Term(\tau_1 \multimap \tau_2)$ and relation $r \in Rel(\tau_1, \tau_2)$, $r \triangleq \{(M, M') | GM =_{\text{gnd}} M' : \tau_2\}$, we have $r = r^{\top\top}$.*

Proof Given $(F, F') \in r^{\top\top}$, we must show $(GF, F') \in \Delta_{\tau_2} = \Delta_{\tau_2}^{\top\top}$. This is true if and only if

$$\forall (V, V') \in \Delta_{\tau_2}^\top . VGF \Downarrow_n \Leftrightarrow V'F' \Downarrow_n .$$

Given $(V, V') \in \Delta_{\tau_2}^\top$, then $(VG, V') \in r^\top$, as given $(M, M') \in r$, we have $(GM, M') \in \Delta_{\tau_2}$. Thus $VGM \Downarrow_n \Leftrightarrow V'M' \Downarrow_n$. Hence $VGF \Downarrow_n \Leftrightarrow V'F' \Downarrow_n$. ∎

5.5 Pilly

This section provides lemmas needed in the model of Pilly.

Lemma 5.33 †

$$\Gamma; \Delta \vdash_{\vec{\alpha}} (\lambda a : \tau.M)N =_{\text{gnd}} M[N/a] : \tau'$$
$$\Gamma; \Delta \vdash_{\vec{\alpha}} (\Lambda \alpha.M)\sigma =_{\text{gnd}} M[\sigma/\alpha] : \tau[\sigma/\alpha]$$
$$\Gamma; \Delta \vdash_{\vec{\alpha}} \textbf{let } !y = !(x = N : \tau) \textbf{ in } M =_{\text{gnd}} M[(\textbf{fix } x : \tau.N)/y] : \tau'$$
$$\Gamma; \Delta \vdash_{\vec{\alpha}} \textbf{let } * = * \textbf{ in } M =_{\text{gnd}} M : \tau$$
$$\Gamma; \Delta \vdash_{\vec{\alpha}} \textbf{let } a_1 \otimes a_2 = M_1 \otimes M_2 \textbf{ in } M_3 =_{\text{gnd}} M_3[M_1, M_2/a_1, a_2] : \tau$$
$$\Gamma; \Delta \vdash_{\vec{\alpha}} \textbf{fix } x : \alpha.M =_{\text{gnd}} M \qquad x \text{ not free in } M$$
$$Y\sigma(!M) =_{\text{gnd}} M!(Y\sigma(!M)) : \sigma$$
$$\Gamma; \Delta \vdash_{\vec{\alpha}} Y\sigma(!M) =_{\text{gnd}} M!(Y\sigma(!M)) : \sigma$$
$$\Gamma; \Delta \vdash_{\vec{\alpha}} \textbf{let } !y = !(x = M_1 : \tau) \textbf{ in } M_2 =_{\text{gnd}} M_2[M_1/y] : \tau' \qquad x \text{ not free in } M_1$$

Proof The first six we have from Lemma 5.23 and Lemma 5.26.

$$\begin{aligned} Y\sigma(!M) &=_{\text{gnd}} (\Lambda\alpha.\lambda f :!(!\alpha \multimap \alpha).\mathbf{let}\ !f' = f\ \mathbf{in}\ f'!(Y\alpha(!f')))\sigma(!M) \\ &=_{\text{gnd}} \mathbf{let}\ !f' =!M\ \mathbf{in}\ f'!(Y\sigma(!M')) \\ &=_{\text{gnd}} (\mathbf{fix}\ x :!\sigma \to \sigma.M)!(Y\sigma(!(\mathbf{fix}\ x :!\sigma \to \sigma.M))) \quad x \text{ not free in } M \end{aligned}$$

As x is not free in M, $M =_{\text{gnd}} \mathbf{fix}\ x :!\sigma \to \sigma.M$. Thus by compatibility and transitivity we get the equation. By the ciu theorem together with Lemma 5.22 we get the next two equations.

Lemma 5.34 *Given* $\Gamma;\Delta_1 \vdash_{\vec{\alpha}} M : \tau \multimap \tau'$ *and* $\Gamma;\Delta_1 \vdash_{\vec{\alpha}} M' : \tau \multimap \tau'$, *

$$\begin{aligned} &\Gamma;\Delta_1 \vdash_{\vec{\alpha}} M =_{\text{gnd}} M' : \tau \multimap \tau' \Leftrightarrow \\ &\quad (\forall N.\Gamma;\Delta_2 \vdash_{\vec{\alpha}} N : \tau \wedge \text{dom}(\Delta_2) \cap \text{dom}(\Delta_1) = \emptyset \Rightarrow \Gamma;\Delta_1,\Delta_2 \vdash_{\vec{\alpha}} MN =_{\text{gnd}} M'N : \tau') \end{aligned}$$

Proof First the $\Rightarrow$. Given $\Gamma;\Delta_1 \vdash_{\vec{\alpha}} M =_{\text{gnd}} M' : \tau \multimap \tau'$ and $\Gamma;\Delta_2 \vdash_{\vec{\alpha}} N : \tau$ such that $\text{dom}(\Delta_2) \cap \text{dom}(\Delta_1) = \emptyset$ then by compatibility $\Gamma;\Delta_1,\Delta_2 \vdash_{\vec{\alpha}} MN =_{\text{gnd}} M'N : \tau'$, thus for all N we have the result.

Now the $\Leftarrow$. We must show $\Gamma;\Delta_1 \vdash_{\vec{\alpha}} M =_{\text{gnd}} M' : \tau \multimap \tau'$. Given a $(\Gamma,\Delta_1,\vec{\alpha})$–closing substitution $\sigma : \tau \multimap \tau' \to (\tau \multimap \tau')[\vec{\tau}/\vec{\alpha}]$, I will show $\sigma M =_{\text{gnd}} \sigma M' : (\tau \multimap \tau')[\vec{\tau}/\vec{\alpha}]$. As $\Gamma;\Delta_1 \vdash_{\vec{\alpha}} M : \tau \multimap \tau'$, $ftv(\Gamma;\tau) \in \vec{\alpha}$, thus for some fresh linear variable a, $\Gamma;a : \tau \vdash_{\vec{\alpha}} a : \tau$. Hence by assumption $\Gamma;\Delta_1,a : \tau \vdash_{\vec{\alpha}} Ma =_{\text{gnd}} M'a : \tau'$ and by substitutivity

$$\emptyset;a : \tau[\vec{\tau}/\vec{\alpha}] \vdash_{\emptyset} \sigma(Ma) =_{\text{gnd}} \sigma(M'a) : \tau'[\vec{\tau}/\vec{\alpha}].$$

Now $\sigma(Ma) = \sigma(M)a$ and $\sigma(M'a) = \sigma(M')a$ thus

$$\emptyset;a : \tau[\vec{\tau}/\vec{\alpha}] \vdash_{\emptyset} \sigma(M)a =_{\text{gnd}} \sigma(M')a : \tau'[\vec{\tau}/\vec{\alpha}].$$

Given $N : \tau[\vec{\tau}/\vec{\alpha}]$ then by substitutivity $\sigma(M)N =_{\text{gnd}} \sigma(M')N : \tau'[\vec{\sigma}/\vec{\alpha}]$. Thus for all $N : \tau[\vec{\sigma}/\vec{\alpha}]$, $\sigma(M)N =_{\text{gnd}} \sigma(M')N : \tau'[\vec{\sigma}/\vec{\alpha}]$. Hence by extensionality of closed terms $\sigma(M) =_{\text{gnd}} \sigma(M')(\tau \multimap \tau')[\vec{\sigma}/\vec{\alpha}]$. As the closing substitution was arbitrary we get the result by the ciu-theorem. ∎

Lemma 5.35 *Given* $\Gamma;\Delta \vdash_{\vec{\alpha}} M : \forall\alpha.\tau$, $\Gamma;\Delta \vdash_{\vec{\alpha}} M' : \forall\alpha.\tau$, *

$$\Gamma;\Delta \vdash_{\vec{\alpha}} M =_{\text{gnd}} M' : \forall\alpha.\tau \Leftrightarrow \forall\tau' \in Typ.\Gamma;\Delta \vdash_{\vec{\alpha}} M\tau' =_{\text{gnd}} M'\tau' : \tau[\tau'/\alpha]$$

Proof First the $\Rightarrow$. Given $\tau' \in Typ$, then by compatibility $\Gamma;\Delta \vdash_{\vec{\alpha}} M\tau' =_{\text{gnd}} M'\tau' : \tau[\tau'/\alpha]$.

Now the $\Leftarrow$. Given a $(\Gamma,\Delta,\vec{\alpha})$–closing substitution $\sigma : \forall\alpha.\tau \to (\forall\alpha.\tau)[\vec{\tau}/\vec{\alpha}]$. By alpha equivalence α can be chosen fresh in τ. Given type $\tau'' \in Typ$, then $\Gamma;\Delta \vdash_{\vec{\alpha}} M\tau'' =_{\text{gnd}} M'\tau'' : \tau[\tau''/\alpha]$. By substitutivity,

$$\emptyset;\emptyset \vdash_{\emptyset} \sigma(M\tau'') =_{\text{gnd}} \sigma(M'\tau'') : (\tau[\tau''/\alpha])[\vec{\tau}/\vec{\alpha}].$$

As τ'' is closed and α is fresh,

$$\emptyset;\emptyset \vdash_{\emptyset} \sigma(M)\tau'' =_{\text{gnd}} \sigma(M')\tau'' : (\tau[\vec{\tau}/\vec{\alpha}])[\tau''/\alpha].$$

Thus as τ'' was arbitrary,

$$\forall\tau'' \in Typ.\sigma(M)\tau'' =_{\text{gnd}} \sigma(M')\tau'' : (\tau[\vec{\tau}/\vec{\alpha}])[\tau''/\alpha].$$

Hence by extensionality for closed terms,

$$\sigma(M) =_{\text{gnd}} \sigma(M') : \forall\alpha.\tau[\vec{\tau}/\vec{\alpha}],$$

and as $\alpha \notin \vec{\alpha}$,

$$\sigma(M) =_{\text{gnd}} \sigma(M') : (\forall\alpha.\tau)[\vec{\tau}/\vec{\alpha}].$$

As the closing substitution was arbitrary we get the result by the ciu-theorem. ∎

6 Program equivalence as a category

This section shows an example on how parametricity can be used to encode types in Lily. I encode a natural number object and proves with the use of the Δ relation that this behaves as it should. The definition of the category is given in [BPR00].

Definition Let the closed Lily types $\tau \in Typ$ be objects and let the morphisms be ground contextual equivalence classes of type $\tau \multimap \tau'$. Given two morphisms M, M' we define

$$M' \circ M \triangleq \lambda a : \tau . M'(Ma)$$
$$Id_\tau \triangleq \lambda a : \tau . a$$

Theorem 6.1 *The objects, morphisms, composition and identity, as defined above is a well defined category.*

Proof We start by ensuring that morphisms, composition and identity are well defined, then we ensure composition is associative and that identity exists for all objects.

welldefinedness of morphisms: A morphism $M \in Term(\tau \multimap \tau')$ has τ as *domain*, τ' as *co-domain* and as contextually equivalent terms have the same type, morphisms are well defined.

welldefinedness of composition: Given $M_1 =_{\text{gnd}} M_1' : \tau_1 \multimap \tau_2$ and $M_2 =_{\text{gnd}} M_2' : \tau_2 \multimap \tau_3$ we have

$$M_2 \circ M_1 = \lambda a : \tau_1 . M_2(M_1 a) =_{\text{gnd}} \lambda a : \tau_1 . M_2(M_1' a) =_{\text{gnd}} \lambda a : \tau_1 . M_2'(M_1' a) = M_2' \circ M_1'$$

welldefinedness of identity: $Id \in Term(\tau \multimap \tau)$ is a morphism thus Id is well defined as above.

Composition is associative: Given $f \in Term(\tau_1 \multimap \tau_2), g \in Term(\tau_2 \multimap \tau_3), h \in Term(\tau_3 \multimap \tau_4)$ we have

$$\begin{aligned} h \circ (g \circ f) &= \lambda a : \tau_1 . h((g \circ f)a) \\ &= \lambda a : \tau_1 . h((\lambda b : \tau_1 . g(fb))a) \\ &=_{\text{gnd}} \lambda a : \tau_1 . h(g(fa)) \\ &=_{\text{gnd}} \lambda a : \tau_1 . (\lambda b : \tau_2 . h(gb))(fa) \\ &= \lambda a : \tau_1 . (h \circ g)(fa) \\ &= (h \circ g) \circ f \end{aligned}$$

For every $g : \tau \multimap \tau'$ and $f : \tau' \multimap \tau$ we have

$$g \circ Id_\tau = \lambda a : \tau . g(Id_\tau a) = \lambda a : \tau . g((\lambda b : \tau . b)a) =_{\text{gnd}} \lambda a : \tau . ga =_{\text{gnd}} g : \tau \multimap \tau'$$
$$Id_\tau \circ f = \lambda a : \tau' . Id_\tau(fa) = \lambda a : \tau' . \lambda b : \tau . b(fa) =_{\text{gnd}} \lambda a : \tau' . fa =_{\text{gnd}} f : \tau' \multimap \tau$$

∎

6.1 The natural numbers $(\mathbb{N}, s, 0)$

This definition is a simple translation of the encoding given in [BMP05], and the proof of the next theorem is a composition of the ideas given in an example in [BPR00] and the flow of the similar proof in [BMP05].

Definition Lets define

$$\mathbb{N} \triangleq \forall\alpha.!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha \tag{6.1}$$

$$0 \triangleq \Lambda\alpha.\lambda f :!(\alpha \multimap \alpha).\lambda x : \alpha.\mathbf{let}\ !f' = f\ \mathbf{in}\ x \tag{6.2}$$

$$s \triangleq \lambda n : \mathbb{N}.\Lambda\alpha.\lambda f :!(\alpha \multimap \alpha).\lambda x : \alpha.\mathbf{let}\ !f' = f\ \mathbf{in}\ f'(n\alpha!f'x) \tag{6.3}$$

Theorem 6.2 $(\mathbb{N}, s, 0)$ *defines a natural numbers object in this category.*

Proof We have to show for any type $\tau \in Typ$, function $b : \mathbb{N} \multimap \tau$ and term $a : \tau$ there is a unique (up to ground contextually equivalence) map h such that

$$h(0) =_{\text{gnd}} a \tag{6.4a}$$

$$h(sx) =_{\text{gnd}} b(hx) \tag{6.4b}$$

Given $\tau \in Typ$, $b \in Term(\mathbb{N} \multimap \tau)$ and $a \in Term(\tau)$, let $f = \lambda n : \mathbb{N}.n\tau!ba$. I will now show that f behaves as h and that f is the unique morphism with that property.

$$\begin{aligned}
f0 &= (\lambda n : \mathbb{N}.n\tau!ba)0 \\
&=_{\text{gnd}} (\Lambda\alpha.\lambda f :!(\alpha \multimap \alpha).\lambda x : \alpha.\mathbf{let}\ !f' = f\ \mathbf{in}\ x)\tau!ba \\
&=_{\text{gnd}} (\lambda f :!(\tau \multimap \tau).\lambda x : \tau.\mathbf{let}\ !f' = f\ \mathbf{in}\ x)!ba \\
&=_{\text{gnd}} (\lambda x : \tau.\mathbf{let}\ !f' =!b\ \mathbf{in}\ x)a \\
&=_{\text{gnd}} \mathbf{let}\ !f' =!b\ \mathbf{in}\ a \\
&=_{\text{gnd}} a \\
f(sm) &=_{\text{gnd}} (sm)\tau!ba \\
&=_{\text{gnd}} ((\lambda n : \mathbb{N}.\Lambda\alpha.\lambda f :!(\alpha \multimap \alpha).\lambda x : \alpha.\mathbf{let}\ !f' = f\ \mathbf{in}\ f'(n\alpha!f'x))m)\tau!ba \\
&=_{\text{gnd}} (\Lambda\alpha.\lambda f :!(\alpha \multimap \alpha).\lambda x : \alpha.\mathbf{let}\ !f' = f\ \mathbf{in}\ f'(m\alpha!f'x))\tau!ba \\
&=_{\text{gnd}} (\lambda f :!(\tau \multimap \tau).\lambda x : \tau.\mathbf{let}\ !f' = f\ \mathbf{in}\ f'(m\tau!f'x))!ba \\
&=_{\text{gnd}} (\lambda x : \tau.\mathbf{let}\ !f' =!b\ \mathbf{in}\ f'(m\tau!f'x))a \\
&=_{\text{gnd}} \mathbf{let}\ !f' =!b\ \mathbf{in}\ f'(m\tau!f'a) \\
&=_{\text{gnd}} b(m\tau!ba) \\
&=_{\text{gnd}} b(fm)
\end{aligned}$$

Given any h with property (6.4), let $r \triangleq \{(M, M')|hM =_{\text{gnd}} M' : \tau\} \in Rel(\mathbb{N}, \tau)$. By lemma 5.32 $r = r^{\top\top}$.

We have $M =_{\text{gnd}} M : \mathbb{N}$, thus $(M, M) \in \Delta_{\mathbb{N}}$. Therefore

$$\forall\sigma, \sigma' \in Typ, \delta \in Rel(\sigma, \sigma').(M\sigma, M\sigma') \in \Delta_{!(\alpha\multimap\alpha)\multimap\alpha\multimap\alpha}(\delta^{\top\top}/\alpha)$$

As $r = r^{\top\top}$ we get

$$\begin{aligned}
(M\mathbb{N}, M\tau) \in &\Delta_{!(\alpha\multimap\alpha)\multimap\alpha\multimap\alpha}(r^{\top\top}/\alpha) = \\
&\Delta_{!(\alpha\multimap\alpha)\multimap\alpha\multimap\alpha}(r/\alpha) = \\
&(!(r \multimap r))^{\top\top} \multimap r \multimap r
\end{aligned}$$

We have

$$\begin{gathered}
\forall M : \mathbb{N}.h(sM) =_{\text{gnd}} b(hM) \\
\therefore \forall(M, M').hM =_{\text{gnd}} M' \Rightarrow h(sM) =_{\text{gnd}} bM' \\
\therefore \forall(M, M').(M, M') \in r \Rightarrow (sM, bM') \in r \\
\therefore (s, b) \in r \multimap r
\end{gathered}$$

As r respects contextually equivalence we have

$$(\mathbf{fix}\ !s, \mathbf{fix}\ !b) \in r \multimap r$$
$$\therefore (!s, !b) \in !(r \multimap r)$$
$$\therefore (!s, !b) \in (!(r \multimap r))^{\top\top}$$

We also have

$$h0 =_{\text{gnd}} a$$
$$\therefore (0, a) \in r$$

Hence we have

$$(M\mathbb{N}!s0, M\tau!ba) \in r$$

and $h(M\mathbb{N}!s0) =_{\text{gnd}} M\tau!ba$. In particular $M\tau!ba =_{\text{gnd}} f(M\mathbb{N}!s0) =_{\text{gnd}} (M\mathbb{N}!s!0)\tau!ba$. As a, b, τ was arbitrary we have

$$\forall M : \mathbb{N}, \tau \in Typ, a : \tau, b : \tau \multimap \tau. M\tau!ba =_{\text{gnd}} (M\mathbb{N}!s0)\tau!ba$$

Now for every type $\tau \in Typ$ and values $V_b :!(\tau \multimap \tau), V_a : \tau$ by lemma 5.30 there exists a fixpoint F_b such that $V_b =_{\text{gnd}}!F_b$, thus

$$(M\mathbb{N}!s0)\tau V_b V_a =_{\text{gnd}} (M\mathbb{N}!s0)\tau!F_b V_a =_{\text{gnd}} M\tau!F_b V_a =_{\text{gnd}} M\tau V_b V_a$$

Hence by lemma 5.27 $M\mathbb{N}!s0 =_{\text{gnd}} M$.

Given a, b, f and h as above we have

$$\forall V \in Val(\mathbb{N}). hV =_{\text{gnd}} h(V\mathbb{N}!s0) =_{\text{gnd}} V\tau!ba =_{\text{gnd}} f(V)$$

thus $h =_{\text{gnd}} f$ and f is unique. ∎

7 Interpreter

To get a feeling about programming in Lily using the the parametric encoding of data types given in [BPR00, Fig. 1], I constructed an interpreter for Lily.

I have chosen to use Standard ML and a LALR parser generator for SML, as I am familiar with the language and I believe that an implementation in SML will make it is easy to do experiments.

I split the task into these steps.

- Give a concrete syntax to the language.
- Type check the abstract syntax.
- Interpret the program.

In Lily, as presented in [BPR00], type variables, linear variables and intuitionistic variables are made up from three different alphabets. This is not user friendly, thus the alphabets must be joined. Unfortunately the given abstract syntax then becomes context dependent, as for a term M and a variable v, v might be a type variable or a term variable, and therefore Mv can either be a type- or a term application. This could be handled by introducing a symbol to distinguish the cases, eg. by using brackets or braces, but I have chosen to distinguish the cases using the context. As I am using a LALR parser generator, the context is not available doing parsing, so I have chosen to parse it as a type application and then afterward I correct the parse tree according to the context. To make it easy and to keep the original abstract syntax, the parser output is in a

data type only similar to Lilys abstract syntax. After parsing I then translate the result into the abstract syntax of Lily.

Besides the ambiguity described above, there are all the usual concerns of functional languages, like associativity of applications and types, etc. I have tried to resemble SML as much as possible, so for instance, applications are left associative and arrow types are right associative. In order to let the programmer force a different behavior I have added parenthesis that can be used in both types and terms.

I have chosen to parse types as I would parse logics. Thus $\forall\alpha.\sigma \multimap \tau = (\forall\alpha.\sigma) \multimap \tau$.

The type checker is basically a type inference algorithm, as in order to check $\Gamma; \Delta_1, \Delta_2 \vdash_{\vec{\alpha}} MN : \tau$ I would have to guess a σ such that $\Gamma; \Delta_1 \vdash_{\vec{\alpha}} M : \sigma \multimap \tau$ and $\Gamma; \Delta_2 \vdash_{\vec{\alpha}} N : \sigma$. This works very well as there is enough type annotations in the syntax. The algorithm works as a combination of top down and bottom up, as bindings are collected on the way down and types are inferred on the way up. This way almost all the typing rules translates nicely into a patten match, some recursive calls and then a combination of results is returned as the type. Unfortunately function application does not follow this scheme, as a non-deterministic split of the linear context is needed. This non-deterministic split can be handled efficiently by tracking the uses of linear variables in sub-terms. Thus my algorithm returns not only the type of a given term, but also the linear variables from the given linearly context not used in the sub-term.

Soon after I had the interpreter set up as described above, I realized that I would like to associate names to terms and types, and preserve these names as long as possible doing interpretation. Furthermore I would like to, first associate a name to a term and then later refer to that name. I have achieved this by adding a syntactic construct with a name and a term to the abstract syntax of terms, a syntactic construct with a name and a type to the abstract syntax of types and by adding syntax to bind names to terms or types for later usage. Thus the interpreter now takes a sequence of bindings, where each binding associates a lily term or type to a name. I would also like to specify how the interpreter should interpret a given term (eg. by call-by-value or call-by-name semantics). This is done by augmenting the binding construct with a keyword, telling the interpreter how it should interpret the term or type. The possibilities are as of today:

- Associate a name with a type.
- Associate a name with a term.
- Associate a name with the result of a call-by-name interpretation of a term.
- Associate a name with the result of a call-by-value interpretation of a term.

After a bit more experience with the interpreter I found that, I tend to use non-recursive thunks quite often. This was quite painfull as in order to create a non-recursive thunk, I had to come up with a fresh variable. I have chosen to solve this by making the name and type information in the abstract syntax of thunks optional. Furthermore I have added support of non-recursive thunks in the concrete syntax. Making the variable and type information optional in the abstract syntax, makes sense as in non-recursive thunks, the variable is intuitionistic and not free in the term.

With the discussion above in mind the implementation should be very easy to read and verify, although the implementation is of course messed up with code to handle α-equivalences.

8 Examples of Lily-programs

Trying to get a feeling about programming in Lily using the encoding of data types given in [BPR00, Fig. 1], I have coded up functions representing the usual operations on natural numbers. So fare I have coded up the following (see figure 1 on page 50)

- Constructors (zero, successor).
- predecessor.
- ifzero.
- addition.
- multiplication.
- Y-combinator.

Furthermore I have coded up the factorial function twice. The first encoding uses the Y-combinator to get a fixed point of $f0 = 1|fx = x * f(x-1)$. The other encoding iterate the function $(x, y) \to (x+1, x*y)$, n times starting with $(!1, 1)$ where x is intuitionistic and then throw away the final x. It turned out that the type of the first encoding is $!nat \multimap nat$, and the type of the second encoding is $nat \multimap nat$. The type of the first encoding seems quite natural as in the second clause x is used twice, so one could think that the factorial function must have this type, but the existence of the second encoding declines such ideas.

It seems natural to compare the encoding in Lilywith the encoding in seconds order lambda calculus. In F_2 I would code the functions like this:

$$
\begin{aligned}
nat &= \forall\alpha.((\alpha \to \alpha) \to \alpha \to \alpha) \\
zero &= \Lambda\alpha.\lambda f : \alpha \to \alpha.\lambda x : \alpha.x \\
succ &= \lambda n : nat.\Lambda\alpha.\lambda f : \alpha \to \alpha.\lambda x : \alpha.f(n\alpha f x) \\
add &= \lambda n : nat.\lambda m : nat.n\ nat\ succ\ m \\
mul &= \lambda n : nat.\lambda m : \mathrm{nat}.n\ nat\ (add\ m)\ zero \\
pred &= \lambda n : nat.\big(n\ \forall\alpha.((nat \to nat \to \alpha) \to \alpha) \\
&\quad (\lambda p : \forall\beta.((nat \to nat \to \beta) \to \beta).p\ \forall\alpha.((nat \to nat \to \alpha) \to \alpha) \\
&\quad\quad (\lambda x : nat.\lambda y : nat.\Lambda\gamma.\lambda f : nat \to nat \to \gamma.f\ (succ\ x)\ x)) \\
&\quad \Lambda\beta.\lambda f : nat \to nat \to \beta.f\ zero\ zero\big) \\
&\quad nat\ \lambda x : nat.\lambda y : nat.y
\end{aligned}
$$

Which looks very much like the encoding in Lily. Of course I cannot encode the Y-combinator as $Y\forall\beta.\beta\ Id_{\forall\beta.\beta}$ would have type $\forall\beta.\beta$ but this type is not inhabitated in F_2.

The difference between the encoding in Lily and the F_2 encoding, seems to be the tricks used to get rid of terms in Lily. This example might not be the best as church numerals easily degenerates to the identity function. Thus the linear types did not cuase much pain.

9 Conclusion

I have presented the language Lily, added a tensor product and unit and shown in full detail a strictness theorem and an unwinding theorem in the extended calculus. Furthermore, I have extended the Δ and obs relations to handle the new constructs and extended all the subsequent theorems. I have proved extensionality result and equivalence of β and η convertible terms for all well-typed terms (also the open ones).

I have constructed an interpreter and worked out an example with natural numbers.

Finally, I thank Lars Birkedal and Rasmus Lerchedahl for the great supervision and proof reading.

```
term Y = let !x = !(x = /\a.\f:!(!a -> a).
                        let !f' = f in f' (!x a (!f'))
                      : \/a.(!(!a->a) -> a))
         in x;

val pair = /\a./\b.\x:a.\y:b./\g.\f:a -> b -> g. f x y
type nat = \/a.(!(a -> a) -> a -> a)
val zero = /\a.\f:!(a -> a). \x:a. let !f' = f in x
val succ = \n : nat. /\a.\f:!(a->a).\x:a.let !f' = f in f' (n a !(y = f' : a->a) x)
val add = \n : nat. \m : nat. n nat !(y = succ : nat -> nat) m
val mul = \n : nat. \m : !nat. let !m' = m in n nat !(y = add m' : nat -> nat) zero

name incpair = \p: \/g.((nat -> nat -> g) -> g).p \/d.((nat -> nat -> d) -> d)
                                    \a:nat.\b:nat.pair nat nat (succ a) (succ b)

name makepair = \i:nat. i \/d.((nat -> nat -> d) -> d) (!incpair)
               (pair nat nat zero zero);

name dopair = \p : \/d.((nat -> nat -> d) -> d).
               p \/d.((nat -> nat -> d) -> d)
                 (\a : nat. \b : nat.
                    makepair (b nat (!(id nat)) a) \/d.((nat -> nat -> d) -> d)
                    (\a:nat.\b:nat. pair nat nat (succ a) b));

name pred =  \n : nat. (n \/d.((nat -> nat -> d) -> d)
                       (!dopair)
                       (pair nat nat zero zero)
                      nat
                      \a : nat. \b : nat. a nat (!id nat) b
                      )


name ifzero = /\a.\n:nat.\h':!(nat->a).\b:!(nat->a).
           let !h = h'
           in
             makepair n a
             \n1:nat.\n2:nat.
               let !g = n1 !(nat -> a) (!\f':!(nat->a).let !f = f' in !h) b
               in g n2;

name fact = Y !nat -> nat
               !\fact' : !(! nat -> nat). \n' : !nat.
                  let !n = n'
                  in let !fact = fact'
                     in ifzero nat n (!\a:nat.mul a (!fact (!pred n))) !succ

val fact2 = \n : nat. (n \/p.((!nat -> nat -> p) -> p)
                         (!\f:\/p.((!nat -> nat -> p) -> p).
                              f \/p.((!nat -> nat -> p) -> p)
                                \x : !nat.\y : nat.
                                  let !x' = x
                                  in pair (!nat) nat (!succ x') (mul y !x')
                         )
                         (pair (!nat) nat (!succ zero) (succ zero))
                       ) nat \x : !nat.\y : nat.let !x' = x in y
```


Figure 1: Lily encodings of natural numbers.

References

[Bar96] Andrew Barber, *Dual intuitionistic linear logic*, Tech. report, University of Edinburgh, 1996.

[BMP05] L. Birkedal, R.E. Møgelberg, and R.L. Petersen, *Parametric domain-theoretic models of linear Abadi-Plotkin logic*, Tech. Report TR-2005-57, IT University of Copenhagen, 2005.

[BPR00] G. M. Bierman, A. M. Pitts, and C. V. Russo, *Operational properties of Lily, a polymorphic linear lambda calculus with recursion*, Fourth International Workshop on Higher Order Operational Techniques in Semantics, Montréal, Electronic Notes in Theoretical Computer Science, vol. 41, Elsevier, September 2000.

[Pit00] A. M. Pitts, *Parametric polymorphism and operational equivalence*, Mathematical Structures in computer Science **10** (2000), 321–359.

[Pit02] Andrew M. Pitts, *Operational semantics and program equivalence*, Applied Semantics, International Summer School, APPSEM 2000, Caminha, Portugal, September 9-15, 2000, Advanced Lectures (London, UK), Springer-Verlag, 2002, pp. 378–412.

[Pit05] A. M. Pitts, *Typed operational reasoning*, Advanced Topics in Types and Programming Languages (B. C. Pierce, ed.), The MIT Press, 2005, pp. 245–289.

Chapter 5

An LAPL-structure from LILY

In this chapter we include a note by Lars Birkedal, Rasmus Lerchedahl Petersen and Rasmus Møgelberg describing how to construct a parametric LAPL-structure from the operational semantics of LILY described in Chapter 4.

Operational Semantics and Models of Linear Abadi and Plotkin Logic

L. Birkedal and R.L. Petersen and R.E. Møgelberg

February 17, 2006

Abstract

In this note we build a parametric LAPL-structure out of the operational semantics of LILY, thereby proving definability of recusive types and obtaining reasoning principles for these.

1 Introduction

Linear Abadi and Plotkin Logic (LAPL) is a logic for reasoning about parametricity for PILL_Y, a polymorphic dual intuitionistic / linear type theory with fixed points introduced in [6]. In *loc. cit* we showed in detail, following Plotkin's suggestions, that LAPL can be used to define a wide range of types, including recursive types. We also defined a sound and complete class of categorical models, called LAPL-structures, for LAPL.

In 2000 Bierman, Pitts, and Russo [2] presented a programming language called LILY. LILY is essentially a polymorphic intuitionistic / linear lambda calculus endowed with an operational semantics.

In this paper we present an LAPL-structure constructed from the operational semantics of LILY. In the model, types of PILL_Y are essentially modeled by sets of LILY terms modulo ground contextual equivalence, and predicates are modeled as subsets of such sets of terms. We prove that the constructed model of LAPL is parametric.

The remainder of this paper is organized as follows.

In Section 2 we introduce the main category, where objects are PILL_Y types and morphisms are ground contextual equivalence classes of PILL_Y terms. We also present the rest of the structure needed to obtain a PILL_Y-model.

In Section 3 we define the logic fibration needed to formulate parametricity. This will be based on ground contextual equivalence and the notion of

$\top\top$-closed relations. The latter will play the role of admissible relations and thus need to satisfy certain closure properties. We establish those in section 4 to obtain a pre-LAPL-structure modelling relations.

In Section 5 we describe the relational interpretation of types by defining the functor denoted J in [4]. This functor will be built out of the Δ-map described by Pitts.

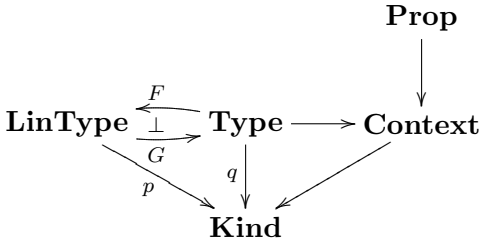
In Section 6 we then check that the parametricity schema does indeed hold, and thus conclude all its consequences. Among those consequences is the definability of the tensor type through LILY-types. We thus see that we have in fact a parametric model of LILY.

In Section 7 we describe the interpretation of PILL_Y into the model we have constructed out of the operational semantics of LILY.

In Section 9 we investigate a few consequences of the fact that the operational semantics of LILY constitues an LAPL-structure and finally conclude in Section 10.

2 The PILL_Y-model

We have this schema of categories to fill out:



We write $\overline{\alpha}^n$ for $\alpha_1, \ldots, \alpha_n$. We let Typ denote the set of closed LILY types, and $\mathsf{Terms}(\sigma)$ the set of (ground contextual equivalence classes of) closed LILY terms of type σ.

2.1 The Base: Kind

Terms and types will be found in **LinType** and purely intuitionistic terms in **Type** as well. Terms and types are sorted into fibers by their set of free type-variables. Thus p and q maps types to their set of free type-variables. Hence **Kind** has as objects sets of type-variables. As a type with free n free type-variables can be seen as a function from Typ^n to Typ, sets of appropriate types can serve as morphisms in **Kind**:

Definition (**Kind**):

Objects: $\overline{\alpha}^n$

Morphisms: $\overline{\sigma}^m : \overline{\alpha}^n \to \overline{\alpha}^m$ iff $\forall i \in \{1, \ldots, m\}.\ \overline{\alpha}^n \vdash \sigma_i$

Note, that we allow $\overline{\alpha}^0$ as an object of **Kind**.

2.1.1 Kind is cartesian

Kind is required to be a cartesian category. The product of $\overline{\alpha}^n$ and $\overline{\alpha}^m$ is given as $\overline{\alpha}^{n+m}$, while the product of $\overline{\sigma}^{n'} : \overline{\alpha}^n \to \overline{\alpha}^{n'}$ and $\overline{\tau}^{m'} : \overline{\alpha}^m \to \overline{\alpha}^{m'}$ needs to take renaming of the free type-variables into account and becomes the concatenation of $\overline{\sigma}^{n'}$ and $\overline{\tau[\alpha_{n+1}, \ldots, \alpha_{n+m}/\alpha_1, \ldots, \alpha_m]}^{m'}$. Projections are constructed by weakening.

2.2 The fibration with LinType

We actually might like to define **LinType** as an indexed (by the objects of **Kind**) category, because it would look very simple. The category for each set $\overline{\alpha}^n$ of type-variables would have as objects types with (at most) those free type-variables and as morphisms (ground contextual equivalence classes of) terms of appropriate type. But since we need a full fledged fibration things look a little less simple, in that we have to apply the Grothedieck construction to the sketched category. We write out the result here since that is the category we deal with. We do, however, write out the fiber categories as well right after we have defined p.

Definition (**LinType**):

Objects: $(\overline{\alpha}^n, \sigma)$ such that $\overline{\alpha}^n \vdash \sigma$

Morphisms: $(\overline{\omega}^m, [M]) : (\overline{\alpha}^n, \sigma) \to (\overline{\alpha}^m, \tau)$, $[M]$ is a ground contextual equivalence class of LILY-terms, such that

- $\overline{\omega}^m : \overline{\alpha}^n \to \overline{\alpha}^m$ in **Kind**
- $-; x : \sigma \vdash_{\overline{\alpha}^n} M : \tau[\overline{\omega}^m/\overline{\alpha}^m]$

Note that an object $(\overline{\alpha}^n,\sigma)$ of **LinType** is a morphism $\overline{\alpha}^n \to \overline{\alpha}^1$ of **Kind**.

Since we have applied the Grothendieck construction, p becomes a mere projection:

Definition ($p:$ **LinType** $\to$ **Kind**):

Objects: $(\overline{\alpha}^n, \sigma) \mapsto \overline{\alpha}^n$

Morphisms: $(\overline{\omega}^m, [M]) \mapsto \overline{\omega}^m$

And now for the promised fiber...
If we define $\mathbf{LinType}_n$ to be $p^{-1}(\overline{\alpha}^n, id_{\overline{\alpha}^n})$ we obtain:

Definition ($\mathbf{LinType}_n$):

Objects: σ such that $\overline{\alpha}^n \vdash \sigma$

Morphisms: $[M]:\ \sigma \to \tau$ such that $-; x : \sigma \vdash_{\overline{\alpha}^n} M : \tau$

Composition is by substitution.

2.2.1 $\mathbf{LinType}_n$ is SMCC

$\mathbf{LinType}_n$ is required to be symetric monoidal closed, and it is. The structure is easily defined in LILY and looks as expected (each fiber is essentially a copy of the term model for DILL in [1]). The tensor is given by:

$$\begin{array}{rcl} I & = & I \\ \sigma \otimes \tau & = & \sigma \otimes \tau \\ [M] \otimes [N] & = & [\text{let } x \otimes y \text{ be } z \text{ in } M \otimes N] \end{array}$$

where $-; x : \sigma \vdash_{\overline{\alpha}^n} M : \sigma'$, $-; y : \tau \vdash_{\overline{\alpha}^n} N : \tau'$ and $-; z : \sigma \otimes \tau \vdash_{\overline{\alpha}^n}$ let $x \otimes y$ be z in $M \otimes N : \sigma' \otimes \tau'$.

The monoidal structure is given by

$$\begin{array}{rcl} rI_\sigma : I \otimes \sigma \multimap \sigma & = & \text{let } x_1 \otimes x_2 \text{ be } x \text{ in let } \star \text{ be } x_1 \text{ in } x_2 \\ lI_\sigma : \sigma \otimes I \multimap \sigma & = & \text{let } x_1 \otimes x_2 \text{ be } x \text{ in let } \star \text{ be } x_2 \text{ in } x_1 \\ a_{\sigma,\tau,\omega} : \sigma \otimes (\tau \otimes \omega) \multimap (\sigma \otimes \tau) \otimes \omega & = & \text{let } x_1 \otimes t \text{ be } x \text{ in let } x_2 \otimes x_3 \text{ be } t \text{ in } (x_1 \otimes x_2) \otimes x_3 \\ s_{\sigma,\tau} : \sigma \otimes I \multimap \sigma & = & \text{let } x_1 \otimes x_2 \text{ be } x \text{ in let } \star \text{ be } x_2 \text{ in } x_1 \end{array}$$

The closed structure, $\sigma \to (-)$, is given by

$$\begin{array}{rcl} \sigma \to \tau & = & \sigma \multimap \tau \\ \sigma \to [M] & = & [\lambda x : \sigma.(M[(fx)/y])] \end{array}$$

where $-; y : \tau \vdash_{\overline{\alpha}^n} M : \tau'$ and $-; f : \sigma \multimap \tau \vdash_{\overline{\alpha}^n} \lambda x : \sigma.(M[(fx)/y]) : \sigma \multimap \tau'$.

2.3 The fibration with Type

The category **Type** has undergone the exact same obfuscation as **LinType**. Comparing the two we see that in **Type** the objects are lists of objects from **LinType** and morphisms in *Type* are lists of morphisms from **LinType**, where the free variables are now in the intuitionistic context. This should be thought of as a variation in to stages. First the free variables are moved to the intuitionistic context to obtain a category of purely intionistic terms. Then this category is closed under cartesian product by turning objects into lists of objects and morphisms into lists of morphisms. This might be easier to see in the fiber category appearing after the definition of q.

Definition (**Type**):

Objects: $(\overline{\alpha}^n, \overline{\sigma}^s)$ such that $\forall i \in \{1, \ldots, s\}.\ \overline{\alpha}^n \vdash \sigma_i$

Morphisms: $(\overline{\omega}^m, \overline{[M]}^r) :\ (\overline{\alpha}^n, \overline{\sigma}^s) \to (\overline{\alpha}^m, \overline{\tau}^r)$ such that

- $\overline{\omega}^m :\ \overline{\alpha}^n \to \overline{\alpha}^m$ in **Kind**
- $\forall i \in \{1, \ldots, r\}.\ \overline{x : \sigma}^s; - \vdash_{\overline{\alpha}^n} M_i : \tau_i[\overline{\omega}^m / \overline{\alpha}^m]$

As with p, q is a mere projection since all the objects of **Kind** have been inserted into the first component of the objects of **Type** by the Grothendieck construction:

Definition ($q :\ \mathbf{Type} \to \mathbf{Kind}$):

Objects: $(\overline{\alpha}^n, \overline{\sigma}^s) \mapsto \overline{\alpha}^n$

Morphisms: $(\overline{\omega}^m, \overline{[M]}^r) \mapsto \overline{\omega}^m$

If we define $\mathbf{Type}_n$ to be $q^{-1}(\overline{\alpha}^n, id_{\overline{\alpha}^n})$ we obtain:

Definition ($\mathbf{Type}_n$):

Objects: $\overline{\sigma}^s$ such that $\forall i \in \{1, \ldots, s\}.\ \overline{\alpha}^n \vdash \sigma_i$

Morphisms: $\overline{[M]}^r :\ \overline{\sigma}^s \to \overline{\tau}^r$ such that $\forall i \in \{1, \ldots, r\}.\ \overline{x : \sigma}^s; - \vdash_{\overline{\alpha}^n} M_i : \tau_i$

2.3.1 $\mathbf{Type}_n$ is CCC

$\mathbf{Type}_n$ is required to be a cartesian closed category. The product is given by juxtaposition (concatenation?) and projections by intuitionistic weakening, while the closed structure is given by:

$$\overline{\sigma}^s \to \overline{\tau}^r \quad = \quad \overline{\overline{!\sigma}^s \multimap \tau}^r$$

which is simply and adaption of the usual equation, $\sigma \to \tau = !\sigma \multimap \tau$, to our case with lists of types.

2.4 The adjuction $F \dashv G$

We also have to define the $F \dashv G$ adjunction. It is mostly given by the fact that **Type** is almost the coKleisli category of the !-monad on **LinType**. Thus G is almost a forgetfull functor and F is almost !.

Definition ($G :$ **LinType** $\to$ **Type**):

Objects: $(\overline{\alpha}^n, \sigma) \mapsto (\overline{\alpha}^n, \sigma)$

Morphisms: $(\overline{\omega}^m, [M]) \mapsto (\overline{\omega}^m, [M])$

Note that G is not the identity on terms, as $-; x : \sigma \vdash_{\overline{\alpha}^n} M : \tau[\overline{\omega}^m/\overline{\alpha}^m]$ is mapped to $x : \sigma; - \vdash_{\overline{\alpha}^n} M : \tau[\overline{\omega}^m/\overline{\alpha}^m]$

Definition ($F :$ **Type** $\to$ **LinType**):

Objects: $(\overline{\alpha}^n, \overline{\sigma}^s) \mapsto (\overline{\alpha}^n, !\sigma_1 \otimes \cdots \otimes !\sigma_s)$

Morphisms:
$(\overline{\omega}^m, \overline{[M]}^r) \mapsto (\overline{\omega}^m, [\text{let } \otimes_i x_i' : \otimes_i !\sigma_i \text{ be } y \text{ in let } \overline{!x}^m \text{ be } \overline{x'}^m \text{ in } \otimes_i !M_i])$

Notice, that $(F \circ G)(\sigma) = !\sigma$.

2.4.1 $F \circ G$ is a comonad

If we define $\mathbf{!} = F \circ G$ and calculate the resulting functor, we get

$$\begin{aligned} \mathbf{!}(\sigma) &= !\sigma \\ \mathbf{!}([M]) &= [\text{let } !y \text{ be } x \text{ in } !M] \end{aligned}$$

This defines a comonad on **LinType** with the following structure:

$$\begin{aligned} \delta_\sigma : !\sigma \to !!\sigma &= [\text{let } !y \text{ be } x \text{ in } !!y] \\ \epsilon_\sigma : !\sigma \to \sigma &= [\text{let } !y \text{ be } x \text{ in } y] \end{aligned}$$

2.5 We have a PILL_Y-model

We have simply build the syntactic model as in the completeness proof in [4] except that terms are identifyed if they are ground contextual equivalent rather than if they can be proven equal. We now argue that the former identification is stronger and that we thus obtain a PILL_Y-model:

From Chapter 4we know that ground contextual equivalence satisfy all the demands of the equational theory. We thus have a PILL_Y-model once we show that all functors are welldefined (respect ground contextual equivalence) and all adjunctions hold (transposition respect ground contextual equivalence)[1]. These demands are easily verified using that ground contextual equivalence is compatible and substitutive.

3 Logic

Logic in the model is based on sets. This gives us classical logic and most structure for free.

We would like to define the fibration **Context** $\to$ **Kind** as the familiar **Fam**(**Set**) $\to$ **Set**, but since **Kind** is not **Set**, we define a functor S : **Kind** $\to$ **Set** and the fibration **Context** $\to$ **Kind** will then be the pullback of **Fam**(**Set**) $\to$ **Set** along S. This just means that we only consider the fibers above images of S. We write out the description here.

Definition (S : **Kind** $\to$ **Set**):

Objects: $\overline{\alpha}^n \mapsto \mathsf{Typ}^n$

Morphisms: $\overline{\sigma}^m \mapsto (\overline{\tau}^n \mapsto \overline{\sigma[\overline{\tau}^n/\overline{\alpha}^n]}^m)$

This functor S tells us how to think of the objects of **Kind** in terms of sets: n free type variables are thought of as all choices of n closed types. The morphism $\overline{\sigma}^m$ is thought of as the corresponding transformation of choices.

[1] Formally we define translations between PILL_Y-terms and LILY-terms to obtain a bijection between LILY-terms modulo ground contextual equivalence and a quotient of PILL_Y-terms corresponding to ground contextual equivalence. Our structure is then isomorphic to a quotien of the PILL_Y-model from the completeness proof. This quotient is shown to be a PILL_Y-model by showing that all functors can be lifted to equivalence classes and that adjuctions still hold. Since all categories and functors are define by the same syntax in all three structures, the actual computations are the same as those one would perform verifying the intuitive argument given above.

Definition (**Context**):

Objects: $(\overline{\alpha}^n, (A_i)_{i \in S(\overline{\alpha}^n)})$, where each A_i is a set.

Morphisms: $(\overline{\sigma}^m, (f_i)_{i \in S(\overline{\alpha}^n)}) : \ (\overline{\alpha}^n, (A_i)_{i \in S(\overline{\alpha}^n)}) \to (\overline{\alpha}^m, (B_j)_{j \in S(\overline{\alpha}^m)})$, such that

- $\overline{\sigma}^m : \ \overline{\alpha}^n \to \overline{\alpha}^m$ in **Kind**
- $f_i : \ A_i \to B_{S(\overline{\sigma}^m)(i)}$

To understand the definition of context, it is easiest to look at the functor I, which describes the part of context in which we are interested, namely the interpretations of types. A type σ with free variables $\overline{\alpha}^n$ is interpreted as the set of closed terms of $\sigma[\overline{\tau}^n/\overline{\alpha}^n]$ for all choices of $\overline{\tau}^n$. Thus a type is mapped to a set of terms indexed by possible choices of closed types for the free type variables.

As the objects of **Type** are lists of types and these are thought of as products of types, the functor I is given by "product of sets of terms":

Definition ($I : \ \mathbf{Type} \to \mathbf{Context}$):

Objects:
$(\overline{\alpha}^n, \overline{\sigma}^s) \mapsto (\overline{\alpha}^n, (\mathsf{Terms}(\sigma_1[\overline{\tau}^n/\overline{\alpha}^n]) \times \cdots \times \mathsf{Terms}(\sigma_s[\overline{\tau}^n/\overline{\alpha}^n]))_{\overline{\tau}^n \in S(\overline{\alpha}^n)})$

Morphisms:
$(\overline{\omega}^m, \overline{[M]}^r) \mapsto (\overline{\omega}^m, (\overline{(t : \ \sigma_1[\overline{\tau}^n/\overline{\alpha}^n]}^s \mapsto \overline{[M[\overline{\tau}^n/\overline{\alpha}^n][\overline{t}^s/\overline{x}^s]]}^r)_{\overline{\tau}^n \in S(\overline{\alpha}^n)})$

I is product-preserving by definition and that it is faithfull follows from extensionality.

The interpretation of relations is "subsets":

$$U : \ \mathbf{LinType} \times_{\mathbf{Kind}} \mathbf{LinType} \to \mathbf{Context} = 2^{I(-) \times I(=)}.$$

Thus relations are simply subsets of the powerset of terms. This enables us to use Pitts original notion of $\top\top$-closed relations as our notion of admissible relations.

For this reason we use the notation $r \subseteq \sigma \times \tau$ to denote the fact that

$$r \subseteq \mathsf{Terms}(\sigma) \times \mathsf{Terms}(\tau)$$

while $r \subseteq_{Adm} \sigma \times \tau$ denotes the fact that

$$r \subseteq \sigma \times \tau \quad \wedge \quad r =^{\top\top} r$$

We then define the subfunctor V of U as "the set of $\top\top$-closed relations" by

Definition (V : **LinType** $\times_{\mathbf{Kind}}$ **LinType** $\to$ **Context**):

Objects: $(\overline{\alpha}^n, \sigma, \tau) \mapsto (\overline{\alpha}^n, (\{r \subseteq_{Adm} S(\sigma)(\vec{\tau}n) \times S(\tau)(\vec{\tau}n)\})_{\overline{\tau}^n \in S(\overline{\alpha}^n)})$

Morphisms: $(\overline{\omega}^m, [M], [N]) \mapsto U(\overline{\omega}^m, [M], [N])$

where the second component of the value on morphisms should be restricted to $\top\top$-closed relations. That this restriction results in a map of $\top\top$-closed relations is seen in section 4.

4 Admissible Relations

To justify that $\top\top$-closed relations can serve as a notion of admissible relations, we must show that they enjoy the closure properties of [4] figure 4. These axioms have been changed slightly since their first release to accomodate the current model. The reason this is not unreasonable is that they were originally simplified for ease-of-use: Rather than showing the rule

$$\frac{\Xi \mid \Gamma \mid \Theta \vdash \rho \subseteq_{Adm} \sigma \times \tau}{\Xi \mid \Gamma \mid \Theta \vdash !\rho \subseteq_{Adm} !\sigma \times !\tau}$$

it was thought easier to prove the three rules

$$\frac{\Xi \mid \Gamma \mid \Theta \vdash \rho \subseteq_{Adm} \sigma \times \tau \quad x, y \notin \Gamma}{\Xi \mid \Gamma \mid \Theta \vdash (x\colon \sigma, y\colon \tau).x \downarrow \supset \rho \subseteq_{Adm} \sigma \times \tau},$$

$$\frac{x, y \notin \Gamma}{\Xi \mid \Gamma \mid \Theta \vdash (x\colon !\sigma, y\colon !\tau).(x \downarrow \mathrel{\mathrlap{\supset}\subset} y \downarrow) \subseteq_{Adm} \sigma \times \tau}$$

and

$$\frac{\Xi \mid \Gamma \mid \Theta \vdash \phi\colon \mathsf{Prop} \quad x, y \notin \Gamma}{\Xi \mid \Gamma \mid \Theta \vdash (x\colon \sigma, y\colon \tau).(x \downarrow \wedge y \downarrow \supset \phi) \subseteq_{Adm} \sigma \times \tau},$$

where $x \downarrow \equiv \exists f\colon \sigma \multimap I.f(x) =_I *$, as these were more immediate in a domain-theoretic setting.

But since the first one is what is really needed, and the last one fails for this particular model, we have reverted to the unoptimized scheme of axioms. Thus, further down, we show the first rule to hold here.

We refer to figure 4 in [4] and provide only a part of a formula to hint at which construction we are debating:

$R \subseteq_{Adm} \sigma \times \tau$:A free admissible relational variable is interpreted as the projection into the set of admissible relations. Thus at every point it returns the admissble relation it receives as the last component of its input.

eq_σ: Equality is simply ground contextual equivalence, which by Chapter 4is the same as Δ, which is $\top\top$-closed.

$\rho(t\,x, u\,y)$: Assume ρ is $\top\top$-closed. We now wish to show $\rho(tM, uN)^{\top\top} \Rightarrow \rho^{\top\top}(tM, uN)$. Writing out the two formulae, we get $\rho(tM, uN)^{\top\top} \Leftrightarrow$

$$\forall f' : \sigma' \multimap !\omega_1, g' : \tau' \multimap !\omega_2.(\forall z' : \sigma', w' : \tau'.\rho(tz', uw') \supset f'z' \Downarrow\Leftrightarrow g'w' \Downarrow) \supset f'M \Downarrow\Leftrightarrow g'N \Downarrow$$

and $\rho^{\top\top}(tM, uN) \Leftrightarrow$

$$\forall f : \sigma \multimap !\omega_1, g : \tau \multimap !\omega_2.(\forall z : \sigma, w : \tau.\rho(z, w) \supset fz \Downarrow\Leftrightarrow gw \Downarrow) \supset f(tM) \Downarrow\Leftrightarrow g(uN) \Downarrow$$

Assume the formula for $\rho(tM, uN)^{\top\top}$ and that $f : \sigma \multimap !\omega_1, g : \tau \multimap !\omega_2$ satisfy $\forall z : \sigma, w : \tau.\rho(z, w) \supset f\ z \Downarrow\Leftrightarrow g\ w \Downarrow$. If we instantiate $\rho(tM, uN)^{\top\top}$ with $f' = f \circ t$, $g' = g \circ u$, we get $f(t\ M) \Downarrow\Leftrightarrow g(u\ N) \Downarrow$ as required.

$\rho(x, y) \wedge \rho'(x, y)$: Conjunction is modelled by intersection, so the following neat argument applies: Assume ρ and ρ' to be $\top\top$-closed. Since

$$\rho \cap \rho' \subset \rho \quad \text{and} \quad \rho \cap \rho' \subset \rho'$$

we get

$$\rho^\top \subset (\rho \cap \rho')^\top \quad \text{and} \quad \rho'^\top \subset (\rho \cap \rho')^\top$$

so

$$\rho^\top \cup \rho'^\top \subset (\rho \cap \rho')^\top$$

Thus

$$(\rho \cap \rho')^{\top\top} \subset (\rho^\top \cup \rho'^\top)^\top \subset \rho^{\top\top} = \rho$$

and likewise $(\rho \cap \rho')^{\top\top} \subset \rho'$, so $(\rho \cap \rho')^{\top\top} \subset \rho \cap \rho'$.

$(x\colon \tau, y\colon \sigma).\rho(y, x)$: Let $\widehat{\rho}$ denote $(x\colon \tau, y\colon \sigma).\rho(y, x)$. Calculation now shows, that $\widehat{\rho}^{\top\top} = \widehat{\rho^{\top\top}}$. Thus

$$\widehat{\rho}^{\top\top}(M, N) = \widehat{\rho^{\top\top}}(M, N) = \rho^{\top\top}(N, M) = \rho(N, M) = \widehat{\rho}(M, N)$$

where we obviously use, that ρ is $\top\top$-closed.

$!\rho \subseteq_{Adm} !\sigma \times !\tau$:Fortunately $!\rho$ is defined as an $\top\top$-closure and is thus $\top\top$ closed.

$\top$:Writing out $\top^{\top\top}(M, N)$ we get

$$\forall f : \sigma \multimap !\omega_1, g : \tau \multimap !\omega_2.(\forall z : \sigma, w : \tau.\top(z, w) \supset fz \Downarrow \Leftrightarrow gw \Downarrow) \supset fM \Downarrow \Leftrightarrow gN \Downarrow$$

Since $\top(z, w)$ holds in general, this is quickly shortened to

$$\forall f : \sigma \multimap !\omega_1, g : \tau \multimap !\omega_2.(\forall z : \sigma, w : \tau.f\ z \Downarrow \Leftrightarrow g\ w \Downarrow) \supset f\ M \Downarrow \Leftrightarrow g\ N \Downarrow$$

which again (due to the very strong requirements on f and g) can be shortened to

$$\forall f : \sigma \multimap !\omega_1, g : \tau \multimap !\omega_2.\top$$

which is of course just the same as $\top$.

$\phi \supset \rho(x, y)$: If ϕ does not hold we get $\top$. If ϕ does hold we get ρ which is admissible.

Quantifications: All quantifications are proved the same way. We will do $(\forall x\colon \omega.\rho)^{\top\top} \subset \forall x\colon \omega.\rho^{\top\top}$. Writing out the two formulae, we get $(\forall x\colon \omega.\rho)^{\top\top}(M, N) \Leftrightarrow$

$$\forall f : \sigma \multimap !\omega_1, g : \tau \multimap !\omega_2.(\forall z : \sigma, w : \tau.(\forall x\colon \omega.\rho_x(z, w)) \supset fz \Downarrow \Leftrightarrow gw \Downarrow) \supset fM \Downarrow \Leftrightarrow gN \Downarrow$$

and $(\forall x\colon \omega.\rho^{\top\top})(M, N) \Leftrightarrow$

$$\forall x\colon \omega.(\forall f : \sigma \multimap !\omega_1, g : \tau \multimap !\omega_2.(\forall z : \sigma, w : \tau.\rho_x(z, w) \supset fz \Downarrow \Leftrightarrow gw \Downarrow) \supset fM \Downarrow \Leftrightarrow g : N \Downarrow)$$

Take any $T\colon \sigma$. Assume the formula for $(\forall x\colon \omega.\rho)^{\top\top}(M, N)$ and that $f : \sigma \multimap !\omega, g : \tau \multimap !\omega$ satisfy $\forall z : \sigma, w : \tau.\rho_T(z, w) \supset f\ z \Downarrow \Leftrightarrow g\ w \Downarrow$. Then, as $\forall x\colon \omega.\rho_x(z, w)$ implies $\rho_T(z, w)$, we can plug f and g into $(\forall x\colon \omega.\rho)^{\top\top}(M, N)$, obtaining $f\ M \Downarrow \Leftrightarrow g : N \Downarrow$ as required.

$\rho \subseteq_{Adm} \sigma \times \tau \wedge \rho' \subseteq \sigma \times \tau \wedge \rho \equiv \rho' \Rightarrow \rho' \subseteq_{Adm} \sigma \times \tau$: We recall, that $\rho \equiv \rho'$ is short hand for $\forall x\colon \sigma, y\colon \tau.\rho(x, y) \supset\subset \rho'(x, y)$, which in our model translates to actual equality among the relations ρ and ρ'. The statement is then obvious.

We must also show, that the rule 2.18 holds:

$$\frac{\Xi \mid \Gamma \mid \Theta \vdash \rho \subseteq !\sigma \times !\tau, \rho' \subseteq_{Adm} !\sigma \times !\tau \quad x, y \notin \Gamma}{\begin{array}{l}\Xi \mid \Gamma \mid \Theta \mid \forall x\colon \sigma, y\colon \tau.\rho(!x, !y) \supset \rho'(!x, !y) \vdash \\ \qquad \forall x\colon !\sigma, y\colon !\tau.x \downarrow\supset\subset y \downarrow\supset (\rho(x, y) \supset \rho'(x, y))\end{array}}$$

We recall the definition $x \downarrow \equiv \exists f\colon \sigma \multimap I.f(x) =_I *$. We observe, that if $\sigma = !\sigma'$ then the strictness theorem applies, yielding $M \downarrow \Rightarrow M \Downarrow$. And the term $\lambda x\colon !\sigma.\mathsf{let}\ !y\ \mathsf{be}\ x\ \mathsf{in}\ *$ testifies, that $M \Downarrow \Rightarrow M \downarrow$. Thus the abstract notion of termination coincides with the concrete one.

We note, that the !'s in $\rho(!x, !y)$ are not the comonad, but the syntactic construction which in LILY is implemented as

$$!M \equiv !(z = M)$$

where z is a fresh variable not free in M.

Now pick $M\colon !\sigma$, $N\colon !\tau$ and assume

$$\begin{array}{l}\rho \subseteq !\sigma \times !\tau \\ \rho' \subseteq_{Adm} !\sigma \times !\tau \\ \forall x\colon \sigma, y\colon \tau.\rho(!x, !y) \supset \rho'(!x, !y) \\ M \Downarrow \between\!\!\!\subset N \Downarrow \\ \rho(M, N)\end{array}$$

We then wish to show $\rho'(M, N)$. We have two cases: Either neither M nor N terminates or they both do. If neither terminates, we have $\rho'(M, N)$ right away, since ρ' is $\top\top$-closed. If they both terminate, they are equivalent to thunks and the assumption $\forall x\colon \sigma, y\colon \tau.\rho(!x, !y) \supset \rho'(!x, !y)$ in conjunction with $\rho(M, N)$ gives us $\rho'(M, N)$.

5 Relational Interpretation

If we write out the definition of **AdmRelCtx** and **AdmRelations** as prescribed in [4] we get

Definition (**AdmRelCtx**):

Objects: $(\overline{\alpha}^n, \overline{\alpha}^m, (A_i)_{i \in S(\overline{\alpha}^{n+m})})$, where each A_i is a set.

Morphisms:

$(\overline{\sigma}^r, \overline{\tau}^s, (f_i)_{i \in S(\overline{\alpha}^{n+m})})$:

$$(\overline{\alpha}^n, \overline{\alpha}^m, (A_i)_{i \in S(\overline{\alpha}^{n+m})}) \to (\overline{\alpha}^r, \overline{\alpha}^s, (B_j)_{j \in S(\overline{\alpha}^{r+s})}),$$

such that

- $\overline{\sigma}^r :\ \overline{\alpha}^n \to \overline{\alpha}^r$ in **Kind**
- $\overline{\tau}^s :\ \overline{\alpha}^m \to \overline{\alpha}^s$ in **Kind**
- $f_i :\ A_i \to B_{S(\overline{\sigma}^m \times \overline{\tau}^s)(i)}$

Definition (**AdmRelations**):

Objects: $(\overline{\alpha}^n, \overline{\alpha}^m, (A_i)_{i \in S(\overline{\alpha}^{n+m})}, \sigma, \tau, (f_i)_{i \in S(\overline{\alpha}^{n+m})})$, such that

- $(\overline{\alpha}^n, \sigma)$ is an object of **LinType**
- $(\overline{\alpha}^m, \tau)$ is an object of **LinType**
- $(\overline{\alpha}^{n+m}, (f_i)_{i \in S(\overline{\alpha}^{n+m})}) :$
  $(\overline{\alpha}^{n+m}, (A_i)_{i \in S(\overline{\alpha}^{n+m})}) \to V(\overline{\alpha}^{n+m}, \sigma, \tau)$
  in **Context**, i.e
  $\forall \overline{\tau}^{n+m} \in S(\overline{\alpha}^{n+m}). \forall a \in A_{\overline{\tau}^{n+m}}.$
  $f_{\overline{\tau}^{n+m}}(a) \subseteq_{Adm} \sigma[\overline{\tau}^{n+m}/\overline{\alpha}^{n+m}] \times \tau[\overline{\tau}^{n+m}/\overline{\alpha}^{n+m}]$

Morphisms:
$(\overline{\sigma}^r, \overline{\tau}^s, (h_i)_{i \in S(\overline{\alpha}^{n+m})}, [M], [N]) :$
$(\overline{\alpha}^n, \overline{\alpha}^m, (A_i)_{i \in S(\overline{\alpha}^{n+m})}, \sigma, \tau, (f_i)_{i \in S(\overline{\alpha}^{n+m})}) \to$
$(\overline{\alpha}^r, \overline{\alpha}^s, (B_i)_{i \in S(\overline{\alpha}^{r+s})}, \omega, \rho, (g_i)_{i \in S(\overline{\alpha}^{r+s})})$
such that

- $(\overline{\sigma}^r, [M]) : \ (\overline{\alpha}^n, \sigma) \to (\overline{\alpha}^r, \omega)$ in **LinType**
- $(\overline{\tau}^s, [N]) : \ (\overline{\alpha}^m, \tau) \to (\overline{\alpha}^s, \rho)$ in **LinType**
- $\forall i \in S(\overline{\alpha}^{n+m}). \ h_i : \ A_i \to B_{S(\overline{\sigma}^r, \overline{\tau}^s)(i)}$
- $\forall i \in S(\overline{\alpha}^{n+m}). \forall a \in A_i.$

$$f_i(a) \subseteq (V(\overline{\omega}^{r+s}, [M], [N])_j \circ g_j \circ h_i)(a)$$

as illustrated by the diagram:

$$\begin{array}{ccccc} A_i & \xrightarrow[|\cap]{f_i} & V(\sigma,\tau)_i & \subseteq 2^{I(\sigma)\times I(\tau)} & \begin{array}{cc} \sigma & \tau \\ [M]\downarrow & \downarrow[N] \\ \omega & \rho \end{array} \\ {\scriptstyle h_i}\downarrow & & \uparrow{\scriptstyle V([M],[N])_j} & & \\ B_j & \xrightarrow{g_j} & V(\omega,\rho)_j & \subseteq 2^{I(\omega)\times I(\rho)} & \end{array}$$

where $\omega_1 \dots \omega_r$ are weakened versions of $\sigma_1 \dots \sigma_r$, $\omega_{r+1} \dots \omega_{r+s}$ are weakened versions of $\tau_1 \dots \tau_s$ and $j = S(\overline{\sigma}^r, \overline{\tau}^s)(i)$.

Note that in the definition of objects σ and τ has been weakened before V is applied to them, and in the definition of morphisms the same has happened to $[M]$ and $[N]$.

The relational interpretations of types is provided by the map of PILL-models J. J can be described by its functorial actions on **Kind** and **LinType**. It is merely defined as the Δ from Chapter 4:

Definition (J_{Base} : **Kind** $\rightarrow$ **AdmRelCtx**):

Objects: $\overline{\alpha}^n \mapsto (\overline{\alpha}^n, \overline{\alpha}^n,$
$(\Pi_{i=1}^n \{r \subseteq_{Adm} \tau_i \times \tau_{i+n}\})_{\overline{\tau}^{2n} \in \mathsf{Typ}^{2n}})$

Morphisms: $\overline{\sigma}^m \mapsto (\overline{\sigma}^m, \overline{\sigma}^m,$
$((r_1 \subseteq_{Adm} \tau_1 \times \tau_{n+1}, \ldots, r_n \subseteq_{Adm} \tau_n \times \tau_{2n}) \mapsto$
$(\Delta_{\sigma_1}(\overline{r}^n/\overline{\alpha}^n), \ldots, \Delta_{\sigma_m}(\overline{r}^n/\overline{\alpha}^n)))_{\overline{\tau}^{2n} \in \mathsf{Typ}^{2n}})$

Definition (J_{Total} : **LinType** $\rightarrow$ **AdmRelations**):

Objects: $(\overline{\alpha}^n, \sigma) \mapsto (J_{Base}(\overline{\alpha}^n), J_{Base}(\sigma))$

Morphisms: $(\overline{\omega}^m, [M]) \mapsto (J_{Base}(\overline{\omega}^m), [M], [M])$

We must show that J is indeed a map of PILL-models. So far J is merely a fibred functor, but according to [10] it is sufficient to show that J is a strong symmetric monoidal closed functor which preserves the comonad structure on the nose. Then J has an extension to a map of PILL-models.

That the comonad structure is preserved on the nose is an easy consequence of the very syntatic nature of our categories. That J is a strong SMC functor is almost as easy. Only the constructs !, $\otimes$ and I are not defined directly as J applied to their non-relational counterparts.

The case of tensor is proven like this: Given $\rho \subseteq_{Adm} \sigma \times \tau$ and $rho' \subseteq_{Adm} \sigma' \times \tau'$, the tensor relation $\rho \otimes \rho' \subseteq_{Adm} \sigma \otimes \sigma' \times \tau \otimes \tau'$ is defined as

$$(x : \sigma \otimes \sigma', y : \tau \otimes \tau').\forall \alpha, \beta, R \subseteq_{Adm} \alpha \times \beta.$$
$$\forall t : \sigma \multimap \tau \multimap \alpha, t' : \sigma' \multimap \tau' \multimap \beta.(\rho \multimap \rho' \multimap R)(t, t') \supset$$
$$R(\text{let } x' \otimes x'' \text{ be } x \text{ in } tx'x'', \text{let } y' \otimes y'' \text{ be } y \text{ in } t'y'y'').$$

Thus, if we for given types σ and τ (in the same fiber) and given terms

$M : \sigma \otimes \tau$ and $N : \sigma \otimes \tau$ wish to compare the statement $M\ J(\sigma \otimes \tau)\ N$ with $M\ J(\sigma) \otimes J(\tau)\ N$, the former is given by

$$M\ \Delta_{\sigma\otimes\tau}\ N$$

while the latter is given by

$$\begin{array}{l}\forall \alpha, \beta, R \subseteq_{Adm} \alpha \times \beta. \\ \forall t : \sigma \multimap \tau \multimap \alpha, t' : \sigma' \multimap \tau' \multimap \beta.(\rho \multimap \rho' \multimap R)(t, t') \supset \\ R(\text{let } x' \otimes x'' \text{ be } M \text{ in } tx'x'', \text{let } y' \otimes y'' \text{ be } N \text{ in } t'y'y'').\end{array}$$

(where $\sigma = \sigma'$ because $J(\sigma)$ is a relation on σ. Likewise $\tau = \tau'$.)

If we assume $M\ \Delta_{\sigma\otimes\tau}\ N$ and introduce the following names

$$\begin{array}{l} r_1 = \Delta_\sigma, \quad r_2 = \Delta_\tau, \quad r_3 = R \\ M_1 = M, \quad M_1' = N \\ M_3 = -; a_1 : \sigma, a_2 : \tau \vdash t\ a_1\ a_2 \\ M_3' = -; a_1' : \sigma, a_2' : \tau \vdash t'\ a_1'\ a_2' \end{array}$$

we know that $(M_1, M_1') \in (r_1 \otimes r_2)^{\top\top}$ and that

$$\begin{array}{rcll} (A, A') \in r_1 & \wedge & (B, B') \in r_2 & \Leftrightarrow \\ A\ \Delta_\sigma A' & \wedge & B\ \Delta_\tau B' & \Rightarrow \\ R(t\ A\ B, t'\ A'\ B') & & & \Leftrightarrow \\ (M_3[A/a_1, B/a_2], M_3'[A'/a_1', B'/a_2']) \in r_3 & & & \end{array}$$

Thus Lemma 5.12 from Chapter 4 applies, telling us that

$$\begin{array}{c} (\text{let } a_1 \otimes a_2 \text{ be } M_1 \text{ in } M_3, \text{let } a_1' \otimes a_2' \text{ be } M_1' \text{ in } M_3') \in r_3 \\ \Updownarrow \\ R(\text{let } x' \otimes x'' \text{ be } M \text{ in } t\ x'\ x'', \text{let } y' \otimes y'' \text{ be } N \text{ in } t'\ y'\ y'') \end{array}$$

Thus $M\ J(\sigma) \otimes J(\tau)\ N$.

For the converse direction choose

$$\begin{array}{l} \alpha = \sigma \otimes \tau, \quad \beta = \sigma \otimes \tau \\ t = t' = \lambda x : \sigma.\lambda y : \tau.x \otimes y \\ R = \Delta_{\sigma\otimes\tau} \end{array}$$

This gives

$$\text{let } x' \otimes x'' \text{ be } M \text{ in } x' \otimes x''\ \Delta_{\sigma\otimes\tau}\ \text{let } y' \otimes y'' \text{ be } N \text{ in } y' \otimes y''$$

which quickly reduces to

$$M\ \Delta_{\sigma\otimes\tau}\ N$$

The case of I is a simple version of the argument above using a similar extensionality property, namely that if $M\ I\ N$ then either neither M nor N terminate or they both evaluate to $\star$.

The case of ! is proven like so: Given an admissible relation $\rho \subseteq_{Adm} \sigma\times\tau$, the lifted relation $!\rho$ is given by

$$(x :!\sigma, y :!\tau).x \Downarrow \sqsupset\sqsubset y \Downarrow \wedge(x \Downarrow \supset \rho(\text{let } !x' \text{ be } x \text{ in } x', \text{let } !y' \text{ be } y \text{ in } y'))$$

Thus, considering a type σ and terms $M :!\sigma$ and $N :!\sigma$, we find that $M\ !(J(\sigma))\ N$ reduces to

$$M \Downarrow \Leftrightarrow N \Downarrow \wedge M \Downarrow \Rightarrow (M\ J(\sigma)\ N)$$

while $M\ J(!\sigma)\ N$ is given as $\Delta_{!\sigma}$ which by Chapter 4is equivalent to

$$(M \not\Downarrow \wedge N \not\Downarrow)\vee\exists x, M', y, N'.M \Downarrow !(x = M' : \sigma)\wedge N \Downarrow !(y = N' : \sigma)\wedge fixx : \sigma.M' =_{\mathsf{gnd}} fixy : \sigma N' \ : \sigma$$

This is equivalent to expression for $M\ !(J(\sigma))\ N$ since

$$\Delta_{!\sigma} = \{((!(x = M : \sigma), !(y = N : \sigma))|fixx : \sigma.M\ \Delta_\sigma\ fixy : \sigma N\ \ : \sigma\}$$

6 Parametricity

The parametricity schema states that for any type

$$\alpha_1, \ldots, \alpha_n \vdash \sigma : \mathbf{Type}$$

equality on σ is given by the relational interpretaion of σ applied to equality relations:

$$\forall\tau_1 : \mathbf{Type}, \ldots, \tau_n : \mathbf{Type}.eq_{\sigma[\tau_1/\alpha_1,\ldots,\tau_n/\alpha_n]} = \sigma(eq_{\tau_1}, \ldots, eq_{\tau_n})$$

Equality is given by contextual equivalence and the relational interpretation is given by the functor J. Since contextual equivalence coincides with the relation Δ, we can rewrite the above equation to

$$\Delta_{\sigma[\tau_1/\alpha_1,\ldots,\tau_n/\alpha_n]} = J(\sigma)(\Delta_{\tau_1}, \ldots, \Delta_{\tau_n}) = \Delta_\sigma[\Delta_{\tau_1}/\alpha_1, \ldots, \Delta_{\tau_n}/\alpha_n]$$

which is the content of lemma 4.13.

Theorem 6.1. *The LAPL-structure considered is a parametric LAPL-structure, i.e., satisfies identity extension, extensionality and very strong equality.*

7 Pitts original category

Since **LinType** has a closed structure, morphisms $\sigma \to \tau$ corresponds to morphisms $1 \to (\sigma \multimap \tau)$ i.e. ground contextual equivalence classes of closed terms of type $\sigma \multimap \tau$. Thus[2] we find the category considered by Pitts in [2] as the fiber over 1 (as he had no free type variables). The proof given there of the definability of coproducts, we thus get for free along with all the other inductive and coinductive types. And now we know they work with free type variables as well.

8 Interpretation

In this section we describe the interpretation of PILL_Y into the model we have constructed out of the operational semantics of LILY.[3]

A PILL_Y kind context $\overline{\alpha}^n$ is interpreted as the object $\overline{\alpha}^n$ in **Kind**.

A PILL_Y type $\overline{\alpha}^n \vdash \sigma$ with n free type variables is modeled as an object in **LinType** in the fiber over $\overline{\alpha}^n$:

$$[\![\overline{\alpha}^n \vdash \sigma]\!] \quad = \quad (\overline{\alpha}^n, \sigma),$$

which we abbreviate as:

$$[\![\sigma]\!] = \sigma.$$

These definitions look deceptively simple. To verify that this is indeed the interpretation one obtains in the LILY LAPL-structure one must of course calculate the interpretation of types in the LILY LAPL-structure, and then one quickly sees that the interpretation is as shown above.

Type contexts are modeled by the tensor and comonad structure on **LinType**. A context $x_1 : \sigma_1, \ldots, x'_n : \sigma_n; a_1 : \tau_1, \ldots, a_m : \tau_m$ is modeled as

$$!{[\![\sigma_1]\!]} \otimes \ldots \otimes !{[\![\sigma_n]\!]} \otimes [\![\tau_1]\!] \otimes \ldots \otimes [\![\tau_m]\!]$$

Thus we use tensor to concatenate contexts and we use **!** to make the types in the intuitionistic context behave intuitionistically. Of course the entire context will be inside one fiber of **LinType**.

Terms with n free type variables are modeled as morphisms in **LinType** in the fiber over $\overline{\alpha}^n$.

[2] One should check, that composition matches.

[3] Note that by the results in the preceding section, we already know that we have a well-defined interpretation of PILL_Y — that is a direct consequence of the fact that we have constructed an LAPL structure. Here we merely try to provide an intuitive description of the resulting interpretation.

A term $\Xi \mid \Gamma; \Delta \vdash t : \sigma$ is modeled as a morphism

$$[\![t]\!] : [\![\Gamma; \Delta]\!] \to [\![\sigma]\!]$$

in $\mathbf{LinType}_n$, that is, as a ground contextual equivalence class of LILY terms, containing a representative M of the form

$$-; x : (!{[\![\sigma_1]\!]} \otimes \ldots \otimes !{[\![\sigma_n]\!]} \otimes [\![\tau_1]\!] \otimes \ldots \otimes [\![\tau_m]\!]) \vdash_\Xi M : \sigma.$$

The inductive description of $[\![t]\!]$ follows from [9, 1]. For example, $[\![\Xi \mid \Gamma; \Delta, \Delta' \vdash t \otimes s \colon \sigma \otimes \tau]\!]$ is the equivalence class of the LILY term

$$\begin{aligned} -; x \colon !\Gamma \otimes \Delta \vdash \ & \text{let } \gamma \otimes \delta \otimes \delta' = \text{split } x \\ & \text{in } [\![t]\!](\gamma \otimes \delta) \otimes [\![s]\!](\gamma \otimes \delta'), \end{aligned}$$

where split is a term definable in LILY such that split x is a tensor product in the obvious way.

Now consider the interpretation of formulas in context:

$$\vec{\alpha} \mid \vec{x} \colon \vec{\sigma} \mid \vec{R} \colon \mathsf{Rel}(\vec{\sigma}', \vec{\sigma}''), \vec{S} \colon \mathsf{AdmRel}(\vec{\tau}', \vec{\tau}'') \vdash \phi \colon \mathsf{Prop}.$$

The interpretation of the above formula, abbreviated as simply $[\![\phi]\!]$, is a family of subsets, indexed by closed types $\vec{\tau}$:

$$\begin{array}{lrl} [\![\phi]\!] & \subseteq & \prod_{\sigma \in \vec{\sigma}} \ \mathsf{Terms}(\sigma[\vec{\tau}/\vec{\alpha}]) \\ & \times & \prod_{\sigma' \in \vec{\sigma}', \sigma'' \in \vec{\sigma}''} \ P(\mathsf{Terms}(\sigma'[\vec{\tau}/\vec{\alpha}]) \times \mathsf{Terms}(\sigma''[\vec{\tau}/\vec{\alpha}])) \\ & \times & \prod_{\tau' \in \vec{\tau}', \tau'' \in \vec{\tau}''} \ P^{\top\top}(\mathsf{Terms}(\tau'[\vec{\tau}/\vec{\alpha}]) \times \mathsf{Terms}(\tau''[\vec{\tau}/\vec{\alpha}])), \end{array}$$

where $P^{\top\top}$ denotes the function that yields the set of all $\top\top$-closed subsets. In plain words, $[\![\phi]\!]$ is a subset of ground contextual equivalence classes of terms of types σ, of relations (on ground contextual equivalence classes of terms) and of $\top\top$-closed relations (on ground contextual equivalence classes of terms). Thus it is a very natural interpretation.

The connectives and quantifiers are interpreted just as we ordinarily do in sets. For example, the interpretation of conjunction $[\![\phi \wedge \phi']\!]$ is the intersection of $[\![\phi]\!]$ and $[\![\phi']\!]$.

The only remaining point to note is the interpretation of substitution of terms into formulas: the interpretation of $[\![\phi[M/x]]\!]$, is obtained as follows. As explained above, the term M is interpreted as an equivalence class of LILY terms $[M]$ and hence it induces an obvious function, which works by substitution, between sets of ground contextual equivalence classes of terms $\mathsf{Terms}(\ldots) \to \mathsf{Terms}(\ldots)$ (formally, the function is obtained via the I functor and the adjunction $F \dashv G$). That function is used to reindex the interpretation of ϕ.

9 Consequences

We briefly recall a few theroems from Chapter 3:

Proposition 9.1 ([6]). *Suppose $\alpha \vdash \sigma(\alpha)$ is a type in pure PILL_Y in which α occurs only positively. In any parametric LAPL-structure* in *is interpreted as an initial algebra and* out *as a final coalgebra for $[\![\sigma]\!]\colon \mathbf{LinType}_1 \to \mathbf{LinType}_1$.*

Proposition 9.2 ([6]). *Suppose $\alpha \vdash \sigma(\alpha)\colon \mathbf{Type}$ is a type in pure PILL_Y (α may appear both positively and negatively). There exists a closed type $rec\ \alpha.\sigma(\alpha)$ in PILL_Y and terms*

$$f\colon rec\ \alpha.\sigma(\alpha) \multimap \sigma(rec\ \alpha.\sigma(\alpha)),$$
$$g\colon \sigma(rec\ \alpha.\sigma(\alpha)) \multimap rec\ \alpha.\sigma(\alpha)$$

such that in any parametric LAPL-structure, f, g are interpreted as each others inverses.

We now consider a few consequences of Theorem 6.1.

Consider the category whose objects are the closed types of LILY and whose morphisms from σ to τ are closed terms of type $\sigma \multimap \tau$ of LILY identified up to ground contextual equivalence. We call this category **Lily**.

As always, type expressions $\alpha \vdash \sigma(\alpha)$ in LILY for which α only appears positively in σ induce endofunctors on **Lily**.

Theorem 9.3. *All functors* **Lily** $\to$ **Lily** *induced by types $\sigma(\alpha)$ in LILY have initial algebras and final coalgebras.*

Proof. This is a simple corollary of Theorems 9.1 and 6.1 once we observe that $\mathbf{LinType}_1$ is equivalent to **Lily** and that types in PILL_Y are simply interpreted as the corresponding types in LILY. □

Likewise we have:

Theorem 9.4. *For all types $\alpha \vdash \sigma(\alpha)\colon \mathbf{Type}$ of LILY, there exists a closed type τ of LILY such that τ and $\sigma(\tau)$ are isomorphic as objects of* **Lily**.

This way we get formal proofs of all the claimed isomorphisms in [2, Figure 1] (*loc. cit.* only includes a formal proof of definability of coproducts).[4] Moreover, it shows that our model can be used to prove correct program transformations based on parametricity for a language with general recursive types, an improvement over earlier work [8], which only dealt with algebraic data types.

[4] See [9] for the details of all the relevant proofs in LAPL.

Example 9.5. As an immediate corollary of the definability of $\otimes$ types in PILL_Y [5], and the observation that $\mathbf{LinType}_1$ is equivalent to **Lily**, we get that in **Lily**, the object $\sigma \otimes \tau$ is isomorphic to $\prod \alpha.(\sigma \multimap \tau \multimap \alpha) \multimap \alpha$. Phrased in purely operational semantics terms, it means that in LILY there are terms f and g, with types $f : \sigma \otimes \tau \to \prod \alpha.(\sigma \multimap \tau \multimap \alpha) \multimap \alpha$ and $g : \prod \alpha.(\sigma \multimap \tau \multimap \alpha) \multimap \alpha \to \sigma \otimes \tau$ such that the LILY terms corresponding to the composition of f and g are ground contextually equivalent to the identity terms.

For general parametric LAPL-structures we may derive *reasoning principles* for the definable inductive, coinductive, and recursive types [7]. The principles look similar to the ones considered by Pitts [11] for classical domain theory. For instance, we have the following principle for coinductive types:

Theorem 9.6. *Suppose that* $R : Rel(\nu\alpha.\sigma(\alpha), \nu\alpha.\sigma(\alpha))$ *is such that* $(out, out) \colon R \to \sigma[R]$. *Then*

$$\forall x, y \colon \nu\alpha.\sigma(\alpha).R(x, y) \supset x =_{\nu\alpha.\sigma(\alpha)} y.$$

For the LILY LAPL-structure this theorem provides us with a coinduction principles for proving contextual equivalence of elements of coinductive types defined via parametric polymorphism in LILY.

Similar results were proved in operational semantics for a language with one top-level recursive type in [3]. However, we want to stress that the definability of recursive types in LAPL, Theorems 9.1 and 9.2, also works for recursive types with parameters [6], as does the reasoning principles for the resulting types. Thus, without involving any form of classical or synthetic domain theory, but just relying on purely operational semantics and general properties of LAPL-structures, we have derived general reasoning principles for recursive types. We are not aware of any other work where such general principles are derived directly from operational semantics.[5]

10 Conclusion

A parametric model of LILY has been established. This proves that all the consequences of parametricity as presented in [4] aplly to LILY. Some of these were already proved in [2], but only in the case of terms with no free type variables.

[5]Note that the use of an LAPL-structure here does not replace conventional use of domain theory in that we do not give a denotational semantics of LILY in a LAPL-structure (and thus do not require an adequacy proof), but rather construct a concrete LAPL-structure directly from the operational semantics of LILY.

References

[1] A. Barber. *Linear Type Theories, Semantics and Action Calculi*. PhD thesis, Edinburgh University, 1997.

[2] G. M. Bierman, A. M. Pitts, and C. V. Russo. Operational properties of Lily, a polymorphic linear lambda calculus with recursion. In *Fourth International Workshop on Higher Order Operational Techniques in Semantics, Montréal*, volume 41 of *Electronic Notes in Theoretical Computer Science*. Elsevier, September 2000.

[3] L. Birkedal and R. Harper. Constructing interpretations of recursive types in an operational setting. *Information and Computation*, 155:3–63, 1999.

[4] L. Birkedal, R. E. Møgelberg, and R. L. Petersen. Parametric domain-theoretic models of linear Abadi & Plotkin logic. Technical Report TR-2005-57, IT University of Copenhagen, February 2005.

[5] L. Birkedal, R.E. Møgelberg, and R.L. Petersen. Parametric domain-theoretic models of linear Abadi & Plotkin logic. Technical Report TR-2005-57, IT University of Copenhagen, February 2005.

[6] L. Birkedal, R.E. Møgelberg, and R.L. Petersen. Parametric domain-theoretic models of polymorphic intuitionistic / linear lambda calculus. In *Proceedings of Mathematical Foundations of Programming Semantics 2005*, 2005.

[7] L. Birkedal, R.L. Petersen, R.E. Møgelberg, and C. Varming. Lily operational semantics and models of linear Abadi-Plotkin logic. Technical Report TR-2006-83, IT University of Copenhagen, 2006. Available at `www.itu.dk/people/birkedal/papers/lily-lapl-tr.pdf`.

[8] P. Johann. On proving the correctness of program transformations based on free theorems for higher-order polymorphic calculi. *Mathematical Structures in Computer Science*, 15(2):201–229, 2005.

[9] R. E. Møgelberg. *Category theoretic and domain theoretic models of parametric polymorphism*. PhD thesis, IT University of Copenhagen, 2005.

[10] R. E. Møgelberg, L. Birkedal, and R. L. Petersen. Categorical models of PILL. Technical Report TR-2005-58, IT University of Copenhagen, February 2005.

[11] A.M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.

Chapter 6

Conclusion

We have constructed an LAPL-structure from the operational semantics of LILY and used it to establish formally definability of a wide range of types in LILY. Moreover, we have derived reasoning principles for definable inductive, coinductive, and recursive types.

In recent work, Møgelberg has investigated the application of LAPL in denotational semantics [11]. In particular, he has shown how one may use any LAPL-structure to define a denotational semantics of FPC. By studying the resulting semantics of FPC in the LILY LAPL structure constructed here, we conjecture that one may extend the semantics to a polymorphic version of FPC and show it adequate with respect to the operational semantics of polymorphic FPC. In operational terms the semantics amounts to a translation of polymorphic FPC into LILY.

Future work also includes studying the interaction of parametric polymorphism with other effects than non-termination, in particular with references.

Acknowledgments

We thank Andy Pitts and Alex Simpson for inspiring discussions.

Bibliography

[1] A. Appel and D. McAllester. An indexed model of recursive types for foundational proof-carrying code. *Transactions on Programming Languages and Systems*, 23(5):657–683, 2001.

[2] A. Barber. *Linear Type Theories, Semantics and Action Calculi*. PhD thesis, Edinburgh University, 1997.

[3] G. M. Bierman, A. M. Pitts, and C. V. Russo. Operational properties of Lily, a polymorphic linear lambda calculus with recursion. In *Fourth International Workshop on Higher Order Operational Techniques in Semantics, Montréal*, volume 41 of *Electronic Notes in Theoretical Computer Science*. Elsevier, September 2000.

[4] L. Birkedal and R. Harper. Constructing interpretations of recursive types in an operational setting. *Information and Computation*, 155:3–63, 1999.

[5] L. Birkedal and R. Møgelberg. Categorical models for Abadi-Plotkin's logic for parametricity. *Mathematical Structures in Computer Science*, 15:709–772, 2005.

[6] L. Birkedal, R.E. Møgelberg, and R.L. Petersen. Parametric domain-theoretic models of polymorphic intuitionistic / linear lambda calculus. In *Proceedings of Mathematical Foundations of Programming Semantics 2005*, 2005.

[7] P. Johann. On proving the correctness of program transformations based on free theorems for higher-order polymorphic calculi. *Mathematical Structures in Computer Science*, 15(2):201–229, 2005.

[8] P. Mellies and J. Vouillon. Recursive polymorphic types and parametricity in an operational framework. In *Proceedings of the 21st Conference on Logic in Computer Science*, 2005.

[9] R. E. Møgelberg. *Category theoretic and domain theoretic models of parametric polymorphism*. PhD thesis, IT University of Copenhagen, 2005.

[10] R. E. Møgelberg, L. Birkedal, and R. L. Petersen. Categorical models of PILL. Technical Report TR-2005-58, IT University of Copenhagen, February 2005.

[11] R.E. Møgelberg. Interpreting polymorphic FPC into domain-theoretic models of parametric polymorphism, February 2006. Manuscript. Submitted for publication.

[12] R.E. Møgelberg, L. Birkedal, and R. L. Petersen. Synthetic domain theory and models of linear Abadi and Plotkin logic. In *Proceedings of Mathematical Foundations of Programming Semantics 2005*, 2005.

[13] A.M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.

[14] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Typed lambda calculi and applications (Utrecht, 1993)*, pages 361–375. Springer, Berlin, 1993.

[15] G.D. Plotkin. Second order type theory and recursion. Notes for a talk at the Scott Fest, February 1993.

[16] Gordon Plotkin and Martín Abadi. A logic for parametric polymorphism. In *Typed lambda calculi and applications (Utrecht, 1993)*, volume 664 of *Lecture Notes in Comput. Sci.*, pages 361–375. Springer, Berlin, 1993.

[17] J.C. Reynolds. Types, abstraction, and parametric polymorphism. *Information Processing*, 83:513–523, 1983.