On Semantics and Applications of Guarded Recursion

Aleš Bizjak

PhD Dissertation

Department of Computer Science
Aarhus University
Denmark

On Semantics and Applications of Guarded Recursion

A Dissertation
Presented to the Faculty of Science and Technology
of Aarhus University
in Partial Fulfillment of the Requirements
for the PhD Degree

by
Aleš Bizjak
Friday 29th January, 2016

Abstract

In this dissertation we study applications and semantics of guarded recursion, which is a method for ensuring that self-referential descriptions of objects define a unique object.

The first two chapters are devoted to applications. We use guarded recursion, first in the form of explicit step-indexing and then in the form of the internal language of particular sheaf topos, to construct logical relations for reasoning about contextual approximation of probabilistic and nondeterministic programs. These logical relations are sound and complete and useful for showing a range of example equivalences.

In the third chapter we study a simply typed calculus with additional "later" and "constant" modalities and a guarded fixed-point combinator. These are used for encoding and working with *guarded recursive* and coinductive types in a *modular* way. We develop a normalising operational semantics, provide an adequate denotational model and a logic for reasoning about program equivalence.

In the last three chapters we study syntax and semantics of a dependent type theory with a family of later modalities indexed by the set of *clocks*, and clock quantifiers. In the fourth and fifth chapters we provide two model constructions, one using a family of presheaf categories and one using a generalisation of the category of partial equilogical spaces. These model constructions are used to design the rules and prove consistency of the type theory presented in the last chapter.

The type theory is a version of polymorphic dependent type theory with one kind, the kind of clocks, and a family of universes. The modalities and clock quantifiers are used for defining coinductive types and functions on them. The type theory is interesting because the *guardedness condition*, ensuring that (co)recursive definitions define unique objects, is expressed using types, in contrast to a syntactic guardedness condition. This allows for a modular treatment of (co)recursive definitions.

Resumé

I denne afhandling studerer vi anvendelser og semantik for “guarded” rekursion, en metode, der kan bruges til at sikre at selvrefererende beskrivelser af object definerer et unikt objekt.

De første to kapitler er helliget anvendelser. We bruger “guarded” rekursion, først i form af eksplicit “step-indexing” og senere i form af det interne sprog for bestemte “sheaf toposes”, til at konstruere logiske relationer til at ræsonnere om kontekstuel ækvivalens af probabilistiske og ikke-deterministiske programmer. Disse logiske relationer er sunde og komplette og nyttige til at vise program ækvivalenser.

I det tredie kapitel studerer vi en simpelt-typet lambdakalkyle med nye “later” og “constant” modaliteter samt en “guarded” fixpunktsoperator. Disse bruges til at indkode og arbejde med “guarded” rekursive og coinduktive typer på en modulær facon. We præsenterer en normaliserende operationel semantik, giver en adækvat denotational model og en loik til at ræonnere om program ævkivalens.

I de sidste tre kapitler studerer vi syntax og semantik for afhængig typeteori med en familie af “later” modaliteter, indekseret over en mænde af ure og ure kvantorer. I det fjerde og femte kapitel præsenterer vi to model konstruktioner, en via en familie af “presheaf” kategorier og en via en generalisering af “equilogical spaces”. Disse model konstruktioner anvendes til at designe typeregler og bevise konsistens af typeteorien præsenteret i det sidste kapitel.

Typeteorien er en version af polymorf afhængig typeteorie med en “kind” af ure og en familie af universer. Modaliteterne og kvantorerne for ure bruges til at definere koinduktive typer og funktioner mellem disse. Typeteorien er interessant fordi betingelsen for at en definition er korrekt “guarded” udtrykkes via typer, i modsætning til via syntaktiske begrænsninger på programmer i typeteorien. Dette muliggør en modulær beskrivelse af korekursive definitioner.

Acknowledgements

First and foremost I would like to thank my supervisor Lars Birkedal for all the guidance and support and for always finding the time to answer my questions. Thanks to Rasmus Møgelberg for hosting me for three months at the ITU, where the work contained in the second half of the dissertation was started, and for continued collaboration. Thanks to all my coauthors for valuable collaboration.

Thanks to all the members of the Logic and Semantics group for making it a good work environment and for all the discussions about work and non-work related topics.

Aleš Bizjak,
Aarhus, Friday 29th January, 2016.

Contents

Part I

Overview

Chapter 1

Introduction

This PhD dissertation is a collection of six papers dealing with applications and semantics of *guarded recursion* in its various forms. In this introduction we hope to describe the background and motivation for considering the problems addressed in the included papers.

We aim to explain the applications of *guarded recursion* first and then provide some mathematical motivation, deriving the basics of guarded recursion using some well-known mathematical facts. This derivation will not be entirely correct in details, and, indeed, there will be some amount of handwaving, but hopefully the intuitions provided will demystify the nature of guarded recursion as understood in this dissertation and make it easier to understand the ideas behind the technical details considered in the papers.

In this dissertation, by *guarded recursion* we mean self-referential constructions where the self-reference is somehow *guarded* by a "later" modality, either a modality on *types* or a modality on *propositions*. This is in contrast to other kinds of "guarded recursion" [34] which are syntactic restrictions on the forms of definitions allowed. This latter kind of guarded recursion can, for instance, be found in some proof assistant based on type theory, like Coq [68].

In Section 1.4 of this introduction we provide a more detailed discussion of contributions of individual papers contained in the dissertation. In Section 1.5 we discuss some problems left open in the included articles and possible directions for future work.

We will discuss some related work throughout the introduction, but more detailed comparisons are left to individual chapters. Section 1.7 defines some common notations used throughout the dissertation.

1.1 Background

In recent years considerable progress has been made on reasoning about programming languages containing a wide array of features: local and higher-order store, recursive types, impredicative polymorphism, nondeterminism, etc. [5, 39, 40, 55, 91, 93] Different methods have been developed. Ranging from relational methods for reasoning about contextual approximation and equivalence in these languages to higher-order separation logics for reasoning about safety properties of single programs. These methods now allow very general specifications and proofs of correctness of sophisticated libraries that utilise a combination of local higher-order state, shared higher-order state and concurrency. Such libraries are commonly used to efficiently implement other, more high-level, constructs and libraries.

A central technical device behind a lot of these advances is *step-indexing*, in various forms, from very elementary presentations to the use of internal languages of particular toposes. Viewed in the most pragmatic way, step-indexing simply means adding natural numbers (the *steps*[1]) at different places in definitions in order to get a handle on recursion. These logical steps are not arbitrary, but they need to be connected to some notion of a "step" related to the behaviour of programs in the programming language being modelled. For instance, the logical steps could be related to the number of concrete steps in the operational semantics it takes a program to terminate.

The simplest use of step-indexing is when we consider a language with general recursive types (this means recursive types where the type variable can appear positively, negatively or both). Let us write $\mu\alpha.\tau$ for the recursive type where α is the recursion variable and let **fold** and **unfold** be the introduction and elimination forms. If the reader is unfamiliar with these Pierce [75] provides a good introduction.

Suppose we wish to construct an operationally based logical relation in the style of Pitts [78] for proving contextual equivalence of programs. Then the first idea would be to define the interpretation of the recursive type $\mu\alpha.\tau$ as

$$[\![\mu\alpha.\tau]\!]_\varphi = \left\{(\mathbf{fold}\,v, \mathbf{fold}\,v') \;\middle|\; (v, v') \in [\![\tau\,[\mu\alpha.\tau/\alpha]]\!]_\varphi\right\}$$

which states that two values are related at type $\mu\alpha.\tau$ if they are canonical forms of this type and the values v and v' are related at the unfolded type. Here φ is the valuation, providing relations for the free type variables of $\mu\alpha.\tau$. This is arguably the most intuitive definition. The problem, however, is that the unfolded type is (in general) bigger than the original type, and so we cannot appeal to induction on the size of the type to ensure that the logical relation is well-defined. Since the type variable α can appear positively or negatively or both we cannot appeal to Knaster-Tarski's fixed point theorem in the interpretation either.

The solution using step-indexing is to add an additional component to the interpretation of the type, which intuitively indicates for how many steps

[1] Sometimes these steps are called abstract, or logical steps to differentiate them from concrete steps which are given by the small-step operational semantics.

the values are related. Thus the definition becomes (approximately)

$$[\![\mu\alpha.\tau]\!]_\varphi = \left\{(n, \mathbf{fold}\, v, \mathbf{fold}\, v') \;\middle|\; n > 0 \to (n-1, v, v') \in [\![\tau[\mu\alpha.\tau/\alpha]]\!]_\varphi\right\}. \tag{1.1}$$

It can be shown that this definition is well-formed by well-founded induction on the lexicographic order of the index n and the size of the type: the unfolded type gets bigger, but the step-index gets smaller. Amal Ahmed [5, 6] provides a good example of this form of use of step-indexing, explaining constructions in detail. In this dissertation, Chapter 2 contains an application of this technique to construct a logical relation for a probabilistic language.

A more intricate example of the use of step-indexing arises if we wish to construct a logical relation for a language with higher-order local store. In this case we get a so-called *type-world circularity* as observed by Ahmed [4].

The idea is that now a type is not only a relation on values, but because of local state it needs to be indexed by a *world* as well. The world can be thought of as the "abstract" state of the program. It specifies in particular the set of locations which are currently allocated and some information about the values that can be stored in those locations. If we are building a logical relation for reasoning about contextual equivalence then the world should specify for each valid location the equality on the values stored there. But the equality on the values stored there also depends on the current world, since locations can store functions or other references!

So, simplifying a bit, we have the following two desiderata where **Val** is the set of values of the language and **Loc** is some countably infinite set of "locations" (not necessarily physical locations).

$$\begin{aligned} \mathcal{T} &\cong \mathcal{W} \to_{\text{mon}} \mathcal{P}(\mathbf{Val} \times \mathbf{Val}) \\ \mathcal{W} &\cong \mathbf{Loc} \to_{\text{fin}} \mathcal{T} \end{aligned} \tag{1.2}$$

The monotonicity requirement in the first equation is needed, intuitively, because if two values are related now, allocating some more references should not invalidate this. If we only have first-order state then this recursive definition of types and worlds is unnecessary since only, e.g., integers can be stored and so the relations stored in the world need not depend on the world, as they have to if we have the ability to store other locations or whole functions.

As an alternative to solving (1.2) we could just solve the following recursive equation

$$\mathcal{T} \cong (\mathbf{Loc} \to_{\text{fin}} \mathcal{T}) \to_{\text{mon}} \mathcal{P}(\mathbf{Val} \times \mathbf{Val}) \tag{1.3}$$

and then define the worlds $\mathcal{W}$ to be as required in (1.3). It is an exercise in set theory to show that *sets* $\mathcal{T}$ and $\mathcal{W}$ satisfying either (1.2) or (1.3) do not exist. The reason is that there are too many functions between sets, and taking only the monotone ones does not weed out enough of them. Thus we either need a different approach to modelling or we need to modify the equation in order to solve it. Step-indexing has been used successfully in doing the latter.

One classical approach [90] to solving such domain equations is offered by domain theory. In such a setup $\mathcal{T}$ and $\mathcal{W}$ would be some kinds of domains, e.g., they could be ω-cpo's. This approach has not been used in recent applications because when using $\mathcal{T}$ to define, e.g., a logical relation, the intuitively correct interpretation of, e.g., the reference type, would not give a Scott continuous function. This of course does not necessarily mean that it is not possible to use classical domain theory to get a useful solution and thus a useful model, but it does appear that the kind of approximation one gets by using domains is not the correct one.

1.2 Abstracting Step-indexing

The approach which works quite well in connection with step-indexing is to solve the equation in some category of metric spaces. Let us see why metric spaces appear naturally in connection with step-indexing. Recall above that with step-indexed logical relations, a type is interpreted not as a relation on values (or terms), but rather as an indexed relation, thus (ignoring the valuation φ for the moment)

$$[\![\tau]\!] \in \mathcal{P}(\mathbb{N} \times \mathbf{Val} \times \mathbf{Val}).$$

More precisely, though, because the step-indices are connected to the behaviour of the program, it makes sense to require "monotonicity", which is the property that if $(n, v_1, v_2) \in [\![\tau]\!]$ then also $(m, v_1, v_2) \in [\![\tau]\!]$ for all $m \leq n$. Intuitively, this makes sense because $(n, v_1, v_2) \in [\![\tau]\!]$ should mean that v_1 and v_2 are indistinguishable at type τ if we only have *n computation steps* available for observing the *behaviour* of v_1 and v_2. What computation steps and what behaviours are depends, of course, on the particular application. In the simplest case a computation step simply means one step in the operational semantics and observation can be termination.

Thus we can refine the interpretation $[\![\tau]\!]$ of a type to be an element of the set

$$\mathcal{P}^{\downarrow}(\mathbb{N} \times \mathbf{Val} \times \mathbf{Val})$$

which is the set of those A satisfying the "monotonicity" property described above.

The set $\mathcal{P}^{\downarrow}(\mathbb{N} \times \mathbf{Val} \times \mathbf{Val})$ comes equipped with a natural metric d which measures for how many steps the sets agree:

$$d(X, Y) = \inf\{2^{-n} \mid \forall j < n, \forall v_1, v_2 \in \mathbf{Val}, (j, v_1, v_2) \in X \leftrightarrow (j, v_1, v_2) \in Y\}.$$

Or in words, X and Y are at a distance 2^{-n} if they have the same elements with indices $j < n$. It is an elementary exercise to show that d is a metric which makes $\mathcal{P}^{\downarrow}(\mathbb{N} \times \mathbf{Val} \times \mathbf{Val})$ into a metric space, which is easily seen to be complete. The reader will have, of course, observed, that there is nothing particular about the set $\mathbf{Val}\times\mathbf{Val}$ with regards to the metric. It can be replaced by any other set X and $\mathcal{P}^{\downarrow}(\mathbb{N} \times X)$ is still a complete metric space.

We have seen that changing the codomain in the equation (1.3) allows us to equip it with a non-trivial complete metric. However to stay inside the (suitable) category of metric spaces we need to also equip the whole right-hand side with a structure of a metric space.

Let us fix a sufficiently large category of complete metric spaces. Notice that the metric space $\mathcal{P}^{\downarrow}(\mathbb{N} \times \mathbf{Val} \times \mathbf{Val})$ has the distance function of a very restricted form. All of its non-zero distances are of the form 2^{-n}. Such metric spaces are called *bisected*.

Further, note that it satisfies a stronger version of the triangle inequality. Such spaces are sufficiently important to have a name.

Definition 1.2.1. A metric space $(\mathcal{M}, d)$ is an *ultrametric* space if d satisfies the strong triangle inequality: for all $x, y, z \in \mathcal{M}$,

$$d(x,z) \leq \max\{d(x,y), d(y,z)\}.$$

In this case, the metric d is called an *ultrametric*. ♦

Let us call the category of complete bisected ultrametric spaces and non-expansive functions between them $\mathfrak{M}$. This category is complete and cartesian closed [20]. The exponential of $\mathcal{M}_1$ and $\mathcal{M}_2$ consists of *non-expansive* functions equipped with the supremum metric. Recall that the function $f : (\mathcal{M}_1, d_1) \to (\mathcal{M}_2, d_2)$ between metric spaces is non-expansive if it does not increase distances: for any two $x, y \in \mathcal{M}_1$

$$d_2(f(x), f(y)) \leq d_1(x,y).$$

For any $\mathcal{T} \in \mathfrak{M}$ the set $\mathbf{Loc} \to_{\text{fin}} \mathcal{T}$ is a complete metric space for a very natural metric: the distance between two finite maps f and f' is either 1 if their domains differ or the maximum of the distances of $f(\ell)$ and $f'(\ell)$ for ℓ in the domain of f and f'.

The equation thus becomes

$$\mathcal{T} \cong (\mathbf{Loc} \to_{\text{fin}} \mathcal{T}) \to_{\text{mon, n.e.}} \mathcal{P}^{\downarrow}(\mathbb{N} \times \mathbf{Val} \times \mathbf{Val}). \tag{1.4}$$

where “n.e.” is the set of non-expansive functions. If we did not use non-expansive functions we would not have achieved much since the equation would be the same as (1.3). The main insight when solving domain equations with mixed variance as in Smyth and Plotkin [90] or in categories of metric spaces is to cut down the number of functions to consider. The difficult part is to cut down the number of functions sufficiently for the solution to exist while still retaining sufficiently many of them for the application at hand. In domain theory this is achieved by considering only continuous functions. In categories of metric spaces this is achieved by considering the non-expansive functions, since this is the natural choice of morphisms between metric spaces.

With regards to step-indexing, non-expansive functions are sufficient because if we only have n steps available for observing the behaviour and we wish to know whether two values are related, then it suffices to know the invariant (the equality of values stored in the heap) for n steps as well.

Thus if we could solve equation (1.4) we would be well on the way of constructing models of languages with higher-order state. Unfortunately, this equation does not appear to have a solution. The existence of a solution to a recursive domain equation in categories of metric spaces typically relies on the functor which describes the equation being *locally contractive*, whereas the right-hand side of (1.4) only gives rise to a *locally non-expansive* functor. We explain now what this means.

Definition 1.2.2. Let $F : \mathfrak{M}^{\mathrm{op}} \times \mathfrak{M} \to \mathfrak{M}$ be a functor. The functor F is *locally non-expansive* if for any $X, Y, Z, W \in \mathfrak{M}$ and any $f, f' : X \to Y$ and $g, g' : Z \to W$ we have

$$d(F(f,g), F(f',g')) \leq \max\{d(f,f'), d(g,g')\}.$$

The functor F is *locally contractive* if for any $X, Y, Z, W \in \mathfrak{M}$ and any $f, f' : X \to Y$ and $g, g' : Z \to W$ we have

$$d(F(f,g), F(f',g')) \leq \frac{1}{2} \cdot \max\{d(f,f'), d(g,g')\}. \qquad ♦$$

Remark 1.2.3. In categorical terms, locally non-expansive functors are precisely the functors *enriched* in $\mathfrak{M}$, since their defining property means precisely that the action of F on morphisms is a morphism in $\mathfrak{M}$. ♦

Example 1.2.4. There is the functor $\frac{1}{2} \cdot -$: it maps the metric space (X, d_X) to the metric space $(X, \frac{1}{2} \cdot d_X)$ where $(\frac{1}{2} \cdot d_X)(x,y) = \frac{1}{2} \cdot (x,y)$. On morphisms it acts as the identity. ♦

It is an easy fact that composing any locally non-expansive functor with the functor $\frac{1}{2} \cdot -$ will give a locally contractive functor.

The reason *locally contractive* functors are interesting is that they have fixed points in the appropriate sense.

Theorem 1.2.5 ([7, 20])**.** *Let $F : \mathfrak{M}^{op} \times \mathfrak{M} \to \mathfrak{M}$ be a locally contractive functor such that $F(1,1)$ is inhabited. Then there exists an inhabited metric space $X \in \mathfrak{M}$ such that $F(X,X) \cong X$. If moreover $F(\emptyset, \emptyset)$ is inhabited then the solution is unique up to isomorphism in $\mathfrak{M}$.* ◊

Thus, coming back to the equation (1.4), if we modify it by composing the right-hand side with the functor $\frac{1}{2}\cdot -$ to get the equation

$$\mathcal{T} \cong \frac{1}{2}\cdot\Big((\mathbf{Loc} \to_{\text{fin}} \mathcal{T}) \to_{\text{mon, n.e.}} \mathcal{P}^{\downarrow}(\mathbb{N}\times\mathbf{Val}\times\mathbf{Val})\Big) \tag{1.5}$$

we can use Theorem 1.2.5 to show that it has a *unique* (up to isomorphism) solution.

Remark 1.2.6. Alternatively, we could have chosen to precompose with the functor $\frac{1}{2}\cdot$ to get the equation

$$\mathcal{T} \cong \left(\mathbf{Loc} \to_{\text{fin}} \frac{1}{2}\cdot\mathcal{T}\right) \to_{\text{mon, n.e.}} \mathcal{P}^{\downarrow}(\mathbb{N}\times\mathbf{Val}\times\mathbf{Val}). \tag{1.6}$$

which also has a solution. The solutions $\mathcal{T}_1$ and $\mathcal{T}_2$ to (1.5) and (1.6) are *not* isomorphic, however the choice of which one to use in applications seems to be largely a matter of convenience, rather than expressiveness, which suggests a degree of arbitrariness in composing with $\frac{1}{2}\cdot$. Moreover, uniqueness of the solution is not needed in the applications to models of type systems and logics. In fact, the only property that is needed is that $\mathcal{T}$ is a solution and even this can often be relaxed to the right-hand side of (1.5) being a retract of $\mathcal{T}$. ♦

The crucial result underlying the construction in Theorem 1.2.5 is the basic result about metric spaces and contractive functions, the Banach's fixed point theorem.

Theorem 1.2.7 (Banach's fixed point theorem)**.** *Let* $(\mathcal{M}, d)$ *be an* inhabited *and* complete *metric space and* $f : (\mathcal{M}, d) \to (\mathcal{M}, d)$ *a function. If there exists a constant* $c < 1$ *such that for all* $x, y \in \mathcal{M}$

$$d(f(x), f(y)) \leq c \cdot d(x, y)$$

then f *has a* unique fixed point. ◊

Using the solution Having the solution $\mathcal{T}$ to either of equations (1.5) or (1.6) is only the first step. We need to be able to use it. Because $\mathcal{T}$ is a solution we have by Theorem 1.2.5 in particular that $\mathcal{T}$ is a *complete and inhabited metric* space. Hence Banach's fixed point theorem applies to $\mathcal{T}$.

This can be utilised when defining the logical relation. Recall that this normally proceeds by induction on types, except that in case of recursive types we cannot do this, since the unfolded type is bigger. However, by its very definition recursive types are fixed points of their defining shapes. That is, consider the type of lists of integers which satisfies $[\mathbb{Z}] \cong 1 + \mathbb{Z} \times [\mathbb{Z}]$. By definition then $[\mathbb{Z}]$ is a fixed point of the operation $\alpha \mapsto 1 + \mathbb{Z} \times \alpha$ on syntactic types. Thus in the semantics we should interpret the recursive types in an analogous way. Since $\mathcal{T}$ is an inhabited and complete metric space it is most natural to try to use Banach's fixed point theorem to construct the interpretation of types. And indeed this works out well and this is one way of understanding what the the construction (1.1) achieves by "decreasing the step-index". For more details see Chapter 2 where Banach's fixed point theorem is used in just this way.

An alternative approach Some researchers, e.g. [93], have also used an alternative approach when constructing step-indexed logical relations for languages with higher-order state without using the machinery of metric spaces. This approach involves simultaneously defining worlds and the interpretation of types by carefully tracking the step-indices to ensure that the definition is well-founded.

This is doable, but since the construction is monolithic and relies crucially on getting the step-indexing exactly right, it is very difficult to make sure that what was defined is indeed correct. Using metric spaces and separating the construction of "semantic types" and worlds from the definition of the logical relation provides a more modular construction which is easier to understand and, more importantly, it is easier to argue that it is correct.

Generalising the metric spaces

Ultrametric spaces behave quite differently from metric spaces one encounters, e.g., in analysis, such as the space of reals. In particular, in ultrametric spaces, given a closed ball $\mathcal{B}$, every point $x \in \mathcal{B}$ is its centre and given two closed balls $\mathcal{B}_1$ and $\mathcal{B}_2$, they are either the same or they are disjoint. This means that given any $n \in \mathbb{N}$, closed balls of radius 2^{-n} partition the space, or equivalently, give rise to an equivalence relation $=_n$ defined as

$$x =_n y \leftrightarrow d(x,y) \leq 2^{-n}$$

or, equivalently, $x =_n y$ if x and y are elements of the same 2^{-n}-ball. Obviously for any $n \in \mathbb{N}$ the relation $=_{n+1}$ is included in the relation $=_n$. Finally, if $x =_n y$ for all n then we have that $d(x,y) = 0$ which means $x = y$.

With these relations we can decompose every bisected ultrametric space $(\mathcal{M}, d)$ into a sequence of its approximations $X(n)$, for $n \in \mathbb{N}$. That is, define $X(n)$ to be the set of closed balls of $\mathcal{M}$ of radius 2^{-n}. Further, for each n there is the function $r_n^X : X(n+1) \to X(n)$ which takes an 2^{-n-1}-ball to the uniquely determined 2^{-n} ball containing it. Such a ball exists precisely because every point x of a closed ball is its centre.

Thus, for each bisected metric space we have a presheaf on ω, the first infinite ordinal. Moreover, if $f : (\mathcal{M}_1, d_1) \to (\mathcal{M}_2, d_2)$ is a non-expansive function and X and Y are presheaves assigned to $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively then the function f gives for each n a function $f_n : X(n) \to Y(n)$ which maps a ball with centre x to the ball with centre $f(x)$. Because f is non-expansive this is well-defined, that is, independent of which x was chosen as the centre. Moreover, it is easy to see that we have

$$r^Y_n \circ f_{n+1} = f_n \circ r^X_n$$

for all n, which means that f gives rise to a natural transformation $X \to Y$.

Thus, we have a functor, call it F, from the category $\mathfrak{M}$ to the category of presheaves over ω, the first infinite ordinal.

Conversely, any presheaf X over ω gives rise to a bisected ultrametric space $(\mathcal{M}, d)$ in the following way. Take the underlying set $\mathcal{M}$ to be the set of global elements $\mathrm{Hom}(1, X)$ and define the distance $d(x,y)$ to be

$$d(x,y) = \inf\left\{2^{-n} \,\middle|\, \forall j < n, x_j(\star) = y_j(\star)\right\}$$

where $\star$ is the unique element of $1(n)$. The pair $(\mathcal{M}, d)$ can easily be seen to be an ultrametric space and moreover, it is easy to see that it is complete.

A precise statement is the following.

Theorem 1.2.8 ([22])**.** *The functor F is full and faithful. It has a right adjoint whose action on objects is described above. In brief, the category of* complete *bisected ultrametric spaces is co-reflective in the category* **PSh** (ω).

Moreover, the inclusion F restricts to an equivalence between the category of complete bisected ultrametric spaces and the full subcategory of **PSh** (ω) *on presheaves X whose restrictions are surjective functions.* ◊

This result allows us to embed the metric spaces of interest into a larger universe of presheaves on ω. One reason this is useful is that **PSh** (ω) has more structure. It is a *topos*. Toposes are interesting because they have a very rich *internal language* [59, 66], and this internal language can be used, in our case, to construct step-indexed models where the indices do not appear explicitly. Instead, one uses a single modality in order to "decrease the step". This is the $\triangleright$, called "later" or "delay", modality.

The internal language of **PSh** (ω) was originally used by Birkedal et al. [22] to construct a step-indexed predicate showing type safety of programs written in an ML-like language with higher-order state, but the realisation that step-indexing is intimately related to modal logic goes back further [9]. In Chapter 3 we use the internal language of the topos of *sheaves* over ω_1, the first *uncountable* ordinal, equipped with the Alexandrov topology to construct a step-indexed logical relation for reasoning about must equivalence, without working with steps explicitly.

Let us come back to our two running examples and see how working in the internal language is helpful. There is a functor $\blacktriangleright : \mathbf{PSh}(\omega) \to \mathbf{PSh}(\omega)$. It maps the object

$$\begin{tikzcd} X(0) & X(1) \arrow[l] & X(2) \arrow[l] & \cdots \arrow[l] \end{tikzcd}$$

of $\mathbf{PSh}(\omega)$ to the object

$$\begin{tikzcd}
1 & X(0) \arrow[l, "!"'] & X(1) \arrow[l] & X(2) \arrow[l] & \cdots \arrow[l]
\end{tikzcd}$$

of $\mathbf{PSh}(\omega)$ where ! is the unique function from $X(0)$ to the singleton set 1.

Recall the equivalence in Theorem 1.2.8. If all of X's restriction maps are surjective *and* $X(0)$ is inhabited, then also all of $\blacktriangleright X$'s restriction maps are surjective. Thus $\blacktriangleright$ gives rise, through the equivalence in Theorem 1.2.8, to a functor on the category of *inhabited* complete bisected ultrametric spaces.

What is this functor? It is, up to isomorphism, the functor $\frac{1}{2} \cdot -$ mentioned above, which leaves the underlying set the same, but multiplies distances by $\frac{1}{2}$.

Now recall the “typical” domain equation in metric spaces

$$\mathcal{T} \cong (\mathbf{Loc} \to_{\text{fin}} \mathcal{T}) \to_{\text{mon, n.e.}} \mathcal{P}^{\downarrow}(\mathbb{N} \times \mathbf{Val} \times \mathbf{Val}).$$

First, the metric space $\mathcal{P}^{\downarrow}(\mathbb{N} \times \mathbf{Val} \times \mathbf{Val})$ is isomorphic to the space of functions $\mathbf{Val} \times \mathbf{Val} \to \mathcal{P}^{\downarrow}(\mathbb{N})$ and $\mathcal{P}^{\downarrow}(\mathbb{N})$, as a metric space, is precisely the set of global elements of the subobject classifier Ω of the topos $\mathbf{PSh}(\omega)$. The morphism in $\mathbf{PSh}(\omega)$ are inherently the non-expansive functions. Hence we can express the equation as the following equation in the topos $\mathbf{PSh}(\omega)$.

$$\mathcal{T} \cong (\mathbf{Loc} \to_{\text{fin}} \mathcal{T}) \to_{\text{mon}} (\mathbf{Val} \times \mathbf{Val}) \to \Omega.$$

Further, the object $(\mathbf{Val} \times \mathbf{Val}) \to \Omega$ is the internal power set, so we have the equation

$$\mathcal{T} \cong (\mathbf{Loc} \to_{\text{fin}} \mathcal{T}) \to_{\text{mon}} \mathcal{P}(\mathbf{Val} \times \mathbf{Val}) \tag{1.7}$$

where $\to_{mon}$ is used to denote the subobject of the exponential consisting of monotone functions, as expressed in the internal language of $\mathbf{PSh}(\omega)$. To summarise, the equation as stated in $\mathbf{PSh}(\omega)$ looks simpler, since there is no more explicit indexing and the codomain of the equation is just the (internal) set of relations on values.

Again, we cannot expect to solve (1.7) as such. However solutions exist [22] for suitably *guarded* equations, which is a condition completely analogous to the functor defining the equation being locally contractive. In brief, the equation

$$\mathcal{T} \cong (\mathbf{Loc} \to_{\text{fin}} \blacktriangleright \mathcal{T}) \to_{\text{mon}} \mathcal{P}(\mathbf{Val} \times \mathbf{Val}) \tag{1.8}$$

has a unique (up to isomorphism) solution.

One of the benefits in solving the equation in $\mathbf{PSh}(\omega)$ is that we can keep working internally. And internally elements of $\mathcal{P}(\mathbf{Val} \times \mathbf{Val})$ are simply relations on values. Indexing is not visible anymore. Hiding the indexing makes a lot of the definitions more straightforward and more familiar. The price to pay is that we must now use the $\triangleright$ modality at certain places and that we are less flexible since we cannot manipulate the indices directly and all the constructions we do must make sense as objects of $\mathbf{PSh}(\omega)$. An example of this appears in Chapter 3 where we must use a "stratified" divergence predicate instead of arguably more natural stratified termination predicate, which is what is used in previous work. The reason is that the stratified termination predicate is not a predicate in the internal logic of the topos $\mathbf{Sh}(\omega_1)$.

This modality $\triangleright$ is a modality on *propositions*, that is, *subobjects*, as opposed to $\blacktriangleright$ which is an operation on *types*. Now propositions in the internal language of $\mathbf{PSh}(\omega)$ are, intuitively, also indexed. That is, whether they hold or not depends on the current step-index. Intuitively, if p is a proposition, $\triangleright p$ holds for $n+1$ steps if p holds for n steps, and $\triangleright p$ also holds when there are no more steps available.

This modality allows us to state an internal version of Banach's fixed point theorem which can then be used, for instance, to construct the logical relation.

Definition 1.2.9 ([22])**.** Internally, a function $f : X \to Y$ is contractive if the following formula holds

$$\forall x, x' : X, \triangleright(x = x') \to f(x) = f(x'). \qquad \blacklozenge$$

For internally contractive functions we have the following internal Banach's fixed point theorem.

Theorem 1.2.10 ([22])**.** *Internally, if $f : X \to X$ is contractive and X is inhabited then f has a unique fixed point.* ◊

Note that this holds even if X's restrictions are not surjective in which case Theorem 1.2.10 is simply the original Banach's fixed point theorem stated in different terms.

Remark 1.2.11. There is also a version of "Banach's fixed point theorem" expressible using the $\blacktriangleright$, which we will describe in Section 1.3 below. This states that any morphism $f : X \to X$ that factors through $\blacktriangleright X$ has a unique fixed point, which is a global element $u : 1 \to X$ satisfying $f \circ u = u$.

That version of the Banach's fixed point theorem is useful when working with *terms*, and is used extensively in Chapters 4 and 7. In contrast, Theorem 1.2.10 is useful also when working with relations in the internal higher-order logic.

These two theorems are intimately related, of course, and in higher-order logic relations are just terms of an appropriate type. The difference however is that higher-order logic is a much more expressive language than simple type theory, which is the language for working with terms.

For a more thorough description of the relationship between the two internal fixed-point theorems we refer to Birkedal et al. [22]. ◆

With these concepts, the interpretation of the recursive type $\mu\alpha.\tau$ can be defined to be the unique fixed point of the function mapping the relation R on values to the relation

$$\left\{(\mathbf{fold}\,v, \mathbf{fold}\,v') \;\middle|\; \triangleright\left((v,v') \in [\![\tau]\!]_{\varphi,\alpha\mapsto R}\right)\right\}$$

which is easily seen to be contractive according to Definition 1.2.9. In Chapter 3 we use essentially this fixed point theorem, but modified slightly since we are working in the topos of sheaves over ω_1.

Thus, to conclude this section, step-indexing is an important tool, however explicit indexing quickly becomes tedious and hard to manage. Working in the internal language of a topos like $\mathbf{PSh}(\omega)$, or in the internal language of related hyperdoctrines, as done by Jung et al. [55], the construction of step-indexed logical relations and logics can be somewhat simplified by hiding the indexing and using the $\triangleright$ and $\blacktriangleright$ modalities.

Of course every topos is also a model of extensional dependent type theory [51, Chapter 10], and so we could also use this as an internal language, i.e., as a language for describing objects and morphisms. Because we are working with a particular model, namely the topos $\mathbf{PSh}(\omega)$, we can extend the type theory with additional modalities. In connection with this, we come to the second part of the introduction and, indeed, the second part of the dissertation.

1.3 Guarded Recursion and Coinductive Types

Proof assistants based on dependent type theory such as Coq [68] do have support for working with coinductive types. However coinductive types present inherent difficulties. Chief among these is that inhabitants of coinductive types are inherently infinite objects which need to be represented and manipulated in a finitary way and ensuring that recursive equations describing elements of coinductive types have unique, or at least principled, solutions is a difficult problem. It is hoped that an approach based on *guarded recursion* could be used for working with coinductive types, however there are still many questions about its practicality. But let us not get ahead of ourselves.

Semantically, coinductive types are final coalgebras of suitable functors describing their shape. From this[2] we get a definition principle for defining functions whose codomains are coinductive types. The simplest example of a coinductive type is the type of streams[3] (say, streams of integers). The shape

[2] For defining elements it would suffice if they were only *weakly* final coalgebras and indeed, *in* the type theory of Coq coinductive types can only be proved to be weakly final.

[3] Although the type of streams is perhaps too simple. Coinductive types involving the sum type are somewhat more troublesome, but the example of streams is less messy.

of streams of integers is described by the functor $\mathbb{S}$

$$\mathbb{S}(X) = \mathbb{Z} \times X.$$

Let $s : [\mathbb{Z}] \to \mathbb{S}([\mathbb{Z}])$ be its final coalgebra. Recall that being a final coalgebra means that for any morphism $\varphi : X \to \mathbb{S}(X)$ there is a *unique* morphism $\textbf{unfold}\,\varphi : X \to [\mathbb{Z}]$ which makes the following diagram commute.

$$\begin{tikzcd}
X \arrow[r, "\textbf{unfold}\,\varphi"] \arrow[d, "\varphi"] & {[\mathbb{Z}]} \arrow[d, "s"] \\
\mathbb{S}(X) \arrow[r, "\mathbb{S}(\textbf{unfold}\,\varphi)"] & \mathbb{S}([\mathbb{Z}])
\end{tikzcd}$$

This works well for a large class of functions one wishes to define, however it is not *modular* in the following sense.

Suppose we have defined the **map** function on streams, which can easily be done using the universal property described above, and we wish to use this function to define the stream **nats** containing all the natural numbers. In a lazy language such as Haskell this can easily be done, and the resulting program has correct operational behaviour, as

$$\textbf{nats} = 0 : \textbf{map succ nats} \tag{1.9}$$

where **succ** is the successor function and : is "cons", the stream constructor.

However using just the universal property we cannot define **nats** directly. We can only define *constant* streams directly. By directly we mean to define an algebra map φ such that $\textbf{unfold}\ \varphi : 1 \to [\mathbb{Z}]$ maps the unique element of 1 to the stream of natural numbers.

Of course, we can define a more general function **intsfrom** which takes as argument an integer $n \in \mathbb{Z}$ and produces the stream of integers starting at n. Then $\textbf{nats} = \textbf{intsfrom}\,0$. However when defining **intsfrom** we are essentially also defining a very special case of the **map** function at the same time. In this sense, definitions using just the universal property are not *modular*.

The definition principle for coinductive types in, for instance, Coq, is somewhat different from using the final coalgebra property. However it has similar limitations and in particular it does not accept (1.9) as a valid definition because it violates the *guardedness condition*. This is a *syntactic* condition which is quite elaborate, but it one of its requirements is that the valid self-references in a coinductive definition must appear *immediately* under a constructor. The right-hand side of (1.9) clearly violates this since the recursive occurrence of **nats** does not appear immediately beneath the stream constructor :, but rather is first passed to the function **map**.

There are good reasons for this guardedness check. The reason why the definition (1.9) is *good* is that the function **map** is well-behaved. For instance if we replaced **map succ nats** by **tail nats** on the right-hand side of (1.9) the definition would not be good, which is to say there would be infinitely many streams satisfying the equation (these would be all the streams starting with 0). What is lacking is a way to distinguish good and bad functions based on types alone. Guarded recursion achieves just that. It enriches the type system which then allows us to express when a function can be used in a recursive definition.

Recall the discussion above in Section 1.2 which related the use of ▶ and ▷ modalities to metric spaces. Let us see why coinductive types are intimately related to complete metric spaces.

Let $F : \mathbf{Set} \to \mathbf{Set}$ be a functor that preserves ω^{op} limits. There are many such functors. In particular all the polynomial functors (in general, not only finitary ones) satisfy this property. Recall that in such a case the functor F has a final coalgebra and its carrier is the limit of the following chain.

$$\begin{tikzcd} F(1) & F^2(1) \arrow[l, "F(!)"'] & F^3(1) \arrow[l, "F^2(!)"'] & \cdots \arrow[l, "F^3(!)"'] \end{tikzcd} \tag{1.10}$$

Call this carrier νF. Recall the construction of limits in **Set**. It gives us the following concrete description of νF as a set of compatible sequences.

$$\nu F = \{\{x_n\}_{n\in\mathbb{N}} \mid \forall n, x_n \in F^n(1) \wedge F^n(!)(x_{n+1}) = x_n\}$$

Sets of sequences can be equipped with a very natural metric based on how far the sequences agree. Define a metric d_F on νF as

$$d_F\left(\{x_n\}_{n\in\mathbb{N}}, \{y_n\}_{n\in\mathbb{N}}\right) = \inf\left\{2^{-k} \;\middle|\; \forall j < k, x_j = y_j\right\}.$$

It is easy to see that the metric d_F makes νF into a bisected metric space which is also *complete*. So perhaps we could use Banach's fixed point theorem to define elements of coinductive types.

Let us see that this works on an example. Take the functor $\mathbb{S}(X) = \mathbb{Z} \times X$ describing the shape of streams of integers. A simple calculation shows that the set $\nu\mathbb{S}$ is the set of streams of integers and the metric d_F compares how far the streams agree:

$$d_F(xs, ys) = \inf\left\{2^{-k} \;\middle|\; \forall j < k, xs_j = ys_j\right\}.$$

What is a non-expansive function $\nu\mathbb{S} \to \nu\mathbb{S}$? A simple calculation shows that these are precisely the functions f, such that for any stream xs, the n-th element of $f(xs)$ only depends on the elements xs_j for $j \leq n$. There are plenty of such functions. For instance for any $f : \mathbb{Z} \to \mathbb{Z}$ the function **map** f is non-expansive, but note that for instance the tail function **tail** is *not* non-expansive.

Remark 1.3.1. Non-expansive functions in this context are often also called causal functions which makes sense if we think of streams as time-indexed collections of elements. At time t_0 we have only the first element available, at time $t_1 > t_0$ the first and second element and so on. Causality then means that output at time t_n does not depend on the input which will only be available in the future, say at time t_{n+1}. ♦

What is a contractive function $\nu\mathbb{S} \to \nu\mathbb{S}$? Recall from Section 1.2 that these are precisely the functions that factor through $\frac{1}{2} \cdot \nu\mathbb{S}$ so we might as well ask what are the functions

$$\frac{1}{2} \cdot \nu\mathbb{S} \to \nu\mathbb{S}.$$

A simple calculation shows that such functions are precisely the functions f such that for any stream xs, the n-th element of $f(xs)$ depends only on the elements xs_j for $j < n$. Note the strictly less than relation.

With these concepts we can explain why (1.9) is a good definition of the stream of natural numbers. The equation (1.9) defines a function $\nu\mathbb{S} \to \nu\mathbb{S}$ which is a composition of functions **map succ** and the function $0 : -$. The function **map succ** is non-expansive as explained above, and the function $0 : -$ is contractive. Hence their composition is a contractive function on $\nu\mathbb{S}$. Since $\nu\mathbb{S}$ is clearly inhabited we can use Banach's fixed point theorem to show that there exists a unique stream satisfying equation (1.9).

In contrast, if we replace **map succ** with the function **tail** in the right-hand side of (1.9) then the composition of $0 : -$ and **tail** would only be non-expansive, but not contractive. Non expansive functions in general do not have fixed points, and even if they do, the fixed points are not necessarily unique, as we can clearly see on this example.

To summarise, a recursive definition of a stream xs as

$$xs = \varphi(xs)$$

will be *good* if φ is contractive.[4]

Thus what we need in our type system is to be able to express when a function is contractive. But recall that if $\mathcal{M}$ is a bisected metric space, then contractive functions from $\mathcal{M}$ to $\mathcal{M}$ are precisely the non-expansive functions from $\frac{1}{2} \cdot \mathcal{M}$ to $\mathcal{M}$. And the functor $\frac{1}{2} \cdot -$ is closely related to the $\blacktriangleright$ modality of $\mathbf{PSh}(\omega)$, except that $\blacktriangleright$ is better behaved on empty spaces.

As we discussed in Section 1.2 bisected complete ultrametric spaces live as a full subcategory in the topos $\mathbf{PSh}(\omega)$ and coincidentally the diagram (1.10) defining the carrier of the final coalgebra of F is an object of $\mathbf{PSh}(\omega)$. In fact, for a large class of functors F we can construct the diagram (1.10), as

[4] This is not an if and only if. In general there are functions which are *not* contractive but still have unique fixed points.

an object of $\mathbf{PSh}(\omega)$, by using the $\blacktriangleright$ modality of $\mathbf{PSh}(\omega)$. This is best seen on an example, so we shall do just that.

Suppose $F : \mathbf{Set} \to \mathbf{Set}$ is the functor $F(X) = \mathbb{Z} \times X$ describing the shape of streams. We can define an analogous functor $F' : \mathbf{PSh}(\omega) \to \mathbf{PSh}(\omega)$ as $F'(X) = \Delta(\mathbb{Z}) \times X$ where $\Delta : \mathbf{Set} \to \mathbf{PSh}(\omega)$ is the constant presheaf functor. Finally, let $G = F' \circ \blacktriangleright$. This functor has a unique (up to isomorphism) fixed point, so let us see what it is. We compute

$$\begin{aligned} G(0) &\cong \mathbb{Z} \\ G(1) &\cong \mathbb{Z} \times \mathbb{Z} \\ G(2) &\cong \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \\ &\vdots \\ G(n) &\cong \mathbb{Z}^{n+1} \end{aligned} \tag{1.11}$$

We can thus see that G is isomorphic to the diagram (1.10). Indeed, this holds in general for any polynomial functor $F : \mathbf{Set} \to \mathbf{Set}$. Any such functor can be lifted to a functor $F' : \mathbf{PSh}(\omega) \to \mathbf{PSh}(\omega)$ and after precomposing with $\blacktriangleright$ we get the diagram (1.10) as the unique fixed point of the resulting functor in $\mathbf{PSh}(\omega)$.

This is interesting because in $\mathbf{PSh}(\omega)$ for each X and each $f : X \to X$ that factors through $\blacktriangleright X$ there is a unique global element $u : 1 \to X$ such that $f \circ u = u$, i.e., f has a unique fixed point. In fact more holds. For each X there is a morphism $\mathbf{next}^X : X \to \blacktriangleright X$. We then have that for any object X of $\mathbf{PSh}(\omega)$ a morphism $\mathbf{fix}^X$ of type $(\blacktriangleright X \to X) \to X$ satisfying for all $f : \blacktriangleright X \to X$ the equality

$$\mathbf{fix}^X f = f\Big(\mathbf{next}(\mathbf{fix}^X f)\Big).$$

This $\mathbf{fix}^X$ is precisely what we need. It takes a *contractive* function and produces its fixed point. Thus we can model the “guarded” fixed point combinator.

Compared to, say, the fixed-point combinator of PCF, which has the type $(A \to A) \to A$ the guarded fixed point combinator cannot be used to vacuously construct inhabitants of types. For instance, there is no term of type $\blacktriangleright 0 \to 0$ where 0 is the initial object in $\mathbf{PSh}(\omega)$. Observe as well that now we have guaranteed existence of fixed points purely based on types. There are no more side-conditions, like the type X being inhabited, which are necessary if we are working with metric spaces. This is because the modality $\blacktriangleright$ on $\mathbf{PSh}(\omega)$ is better behaved on the initial object as the functor $\frac{1}{2} \cdot -$ is on the initial object of the category of bisected complete ultrametric spaces. Though these two are equivalent on inhabited objects as explained in the preceding sections.

However by adding the $\blacktriangleright$ modality and the fixed-point combinator **fix** we have introduced new types and terms. Thus we also need a new equational theory and possibly some more auxiliary constructs. This is one of the problems addressed in Chapters 4 and 7.

Generalising guarded recursion

The use of a guarded fixed point combinator to get some benefits of general recursion, but still disallowing vacuous non-termination, is due to Nakano [72] who added such a fixed point combinator to the simply typed calculus and proved normalisation of the calculus.

As above, the fixed-point combinator in this calculus is a term of type

$$(\blacktriangleright A \to A) \to A$$

and one way to think of it is that it is reflecting Banach's fixed point theorem in a type system. In fact, Birkedal et al. [19] constructed a model of the calculus using the category of *inhabited* complete bisected ultrametric spaces, following motivation we have outlined above, where this fixed point combinator is justified precisely by appeal to Banach's fixed point theorem.

However such a calculus on its own is not so useful for working with coinductive types. To see this recall the type of (guarded) streams of integers $[\mathbb{Z}]$, which is the unique fixed point the functor G described in (1.11). This type satisfies

$$[\mathbb{Z}] \cong \mathbb{Z} \times \blacktriangleright [\mathbb{Z}].$$

What if we wish to take the tail of this stream? We can certainly do that, but the type of the tail function is $[\mathbb{Z}] \to \blacktriangleright [\mathbb{Z}]$ and there is no way of removing the $\blacktriangleright$ modality in general without making it redundant. If we think of types as metric spaces again for a moment, we can see that the canonical function from $[\mathbb{Z}]$ to $\frac{1}{2} \cdot [\mathbb{Z}]$, which is the function mapping x to x, does not have an inverse. To be more precise, it does not have an inverse in $\mathfrak{M}$, that is, it does not have a non-expansive inverse. Thus to get rid of the $\blacktriangleright$ we need to allow more morphisms, however this needs to be done in a controlled way lest we trivialise the modality. We still wish the functions $\blacktriangleright X \to X$ to be the ones with unique fixed points.

This suggests that perhaps we should work in two different settings. In one we have something akin to Banach's fixed point theorem available for defining functions and elements of coinductive types, but once these elements are defined, we should pass to a setting where we have more morphisms available so we can use the defined element more liberally.

A Nakano [72] like calculus with guarded recursion was extended with *clocks* and *clock* quantifiers in order to get just this ability by Atkey and McBride [11]. *Clocks* are simply names, drawn from some countably infinite set. The idea is that for each clock κ there is a modality $\blacktriangleright^\kappa$ completely analogous to the $\blacktriangleright$ modality discussed above. Where multiple different modalities are useful is when working with *nested* coinductive types, but we shall not dwell on that here. The intuition behind naming the set of names “clocks” is that elements of the type $\blacktriangleright^\kappa \tau$ are thought to be elements of type τ which we only have available *later*, that is, after we have done some work. Different clocks then correspond to the ability to delay along different time streams.

The main new idea of their calculus are *clock quantifiers* $\forall\kappa$. These allow for a *controlled elimination* of $\blacktriangleright^\kappa$ using a term of type $(\forall\kappa.\ \blacktriangleright^\kappa \tau) \to (\forall\kappa.\tau)$. The intuition is that elements of the type $\forall\kappa.\tau$ are elements of τ that are always available. These clock quantifiers behave very similarly to polymorphic quantification $\forall\alpha$ in System F, although in Atkey and McBride’s calculus there is one important difference and it is to do with the elimination rule, since in their calculus there is no clock substitution in general, but only clock weakening and permutation. This causes significant problems when trying to extend their approach to a dependently typed calculus. This led to a more refined model [28] supporting clock substitution in general. This model is explained in Chapter 5.

Of course, if we have a term t of type τ we cannot necessarily form a term of type $\forall\kappa.\tau$. We can only form a term $\Lambda\kappa.t$ of type $\forall\kappa.\tau$ if the term t does not depend on any variables that vary along the κ dimension. This is completely analogous to how one introduces terms of type $\forall\alpha$ in a System F like setting and this restriction is the reason why $\forall\kappa$ allows for a *controlled* elimination of $\blacktriangleright$.

In such a calculus Atkey and McBride showed that final coalgebras of a large class of functors can be encoded with the help of $\blacktriangleright$ as we have argued in the previous section. In particular, using clock quantifiers, we can get real streams in the calculus which in particular means we can type the ordinary **tail** function.

This works as follows. If we start with the functor $F : \mathbf{Set} \to \mathbf{Set}$ and we wish to construct its final coalgebra what we do is first take the (unique) fixed point of F lifted to $\mathbf{PSh}(\omega)$ and precomposed with $\blacktriangleright$. We get the diagram (1.10) (page 16) as an object of $\mathbf{PSh}(\omega)$. What the $\forall\kappa$ does is take the limit of this diagram to get an object of $\mathbf{Set}$ again. As we have mentioned above, this is one way to construct final coalgebras of a wide class of functors in $\mathbf{Set}$. If we instead think in terms of metric spaces then $\forall\kappa$ is, intuitively[5], simply the forgetful functor from the category $\mathfrak{M}$ to $\mathbf{Set}$.

Of course to reflect this semantic construction into a type theory we need to add sufficiently many rules for working with $\forall\kappa$ and $\blacktriangleright$. The calculus in Chapter 7 is expressive enough to encode final coalgebras for polynomial functors using $\blacktriangleright$ and $\forall\kappa$ and it is expressive enough to reason about elements

[5] When there are multiple clocks we can sadly no longer think in terms of ordinary metric spaces.

of final coalgebras defined in such a way.

One alternative to clocks and clock quantifiers is to use another modality, called $\square$ or $\blacksquare$. The idea is the same as for the clock quantifier $\forall\kappa$, but the technicalities are somewhat different, since we are not switching between different settings, as with the $\forall\kappa$, but we always stay inside one universe, the topos $\mathbf{PSh}(\omega)$.

Such a modality on proposition in context of guarded recursion was first introduced in [26], an extended version of which appears as Chapter 3 of this dissertation. In [31], an extended version of which appears as Chapter 4, we have developed a simply typed calculus with two modalities, $\blacktriangleright$ and $\blacksquare$ and provided a normalising operational semantics, together with a program logic, which also has two modalities, $\triangleright$ and $\square$.

1.4 Outline of the Dissertation

This dissertation consists of six papers, five of which have been presented at peer-reviewed conferences or accepted for publication, and one of the papers (Chapter 6) is currently under review. Each of the following chapters is based on one publication, but some have been considerably extended to include more explanations and proofs. We will discuss the main themes and contributions of these publications below, but first we mention dependencies between the chapters.

The first three chapters are independent of the last three. Chapters 2, 3 and 4 can be read independently, except that it might be helpful to read Chapter 2 before reading Chapter 3 if the reader is not familiar with step-indexed logical relations. However there is no formal dependency.

Chapter 6 relies heavily on results in Chapter 5, which can be read independently of other chapters.

Chapter 7 can be read independently of Chapters 5 and 6 if the reader is not too interested in the semantics, but to understand the motivations for the design of the rules understanding the two chapters on the semantics is necessary.

Chapter 2: Step-indexed Logical Relations for Probability

Chapter 2 is an updated and extended version of [25]. It is extended with additional proofs and examples which only appeared in the appendix to the published version.

The paper constructs a step-indexed logical relation for a call-by-value probabilistic language with a rich type system, including in particular impredicative polymorphism and general iso-recursive types. We only consider a language with primitives for discrete probability distributions. The main part of the paper deals with a language without references. In the end we extend the language with local ground store.

The main result of the paper is that the logical relation we construct provides a sound and complete method for reasoning about contextual approximation and equivalence of programs (in both languages, with and without state). This method moreover appears to be useful for reasoning about a range of examples, although there are examples of equivalences which we do not know how to deal with. See Section 1.5 below for a discussion of one such example and the reasons it is beyond the reach of our logical relation.

One of the interesting points about this logical relation is how little needs to change compared to a step-indexed logical relation for a deterministic language. Indeed, most of the construction stays the same, it is only in the lifting of relations on values to relations on expressions that we need to change the definitions. This part is not surprising, since this lifting uses the operational behaviour of terms. The reason why very little needs to change seems to be the use of *biorthogonality*, or $\top\top$-*closure*.

The logical relation uses step-indexing to ensure that the interpretation of types is well-defined in presence of general recursive types.

The motivation for considering the problem of extending step-indexed logical relations to the probabilistic case were results by Dal Lago et al. [36] and Crubillé and Lago [35] who developed (bi)simulation techniques for reasoning about similar languages, although they did not consider state.

Chapter 3: A Model of Countable Nondeterminism in Guarded Type Theory

This chapter is a version of [26] extended with an appendix which provides more details and proofs of claims made in the paper.

We consider a System F like language with general recursive types and with a countable choice primitive and we use the internal language of the sheaf topos $\mathbf{Sh}(\omega_1)$, where ω_1 is the first uncountable ordinal equipped with the Alexandrov topology, to construct the step-indexed logical relation and prove it sound and complete with respect to *must* contextual equivalence. The logical relation itself is based on previous work [23] which uses step-indexing using the ordinal ω_1.

This research was prompted by the following question: What breaks if we try to construct the logical relation in the internal language of $\mathbf{PSh}(\omega)$? In retrospect, this is quite embarrassingly clear, as is often the case, and it is to do with adequacy, but we are getting ahead of ourselves.

The construction of the logical relation and the proof of the fundamental lemma (more generally, the proof of congruence) can be done in the internal language of $\mathbf{PSh}(\omega)$. Recall that one way to define contextual equivalence is as the largest *adequate* congruence relation, thus to show soundness of the logical relation we need it to be adequate. This property fails if we work in the internal language of $\mathbf{PSh}(\omega)$ and the reason is that the must-termination predicate $\Downarrow$ is not continuous, i.e., it does not preserve unions, and so to define $\Downarrow$ by iteration we need to iterate up to some higher ordinal. The ordinal ω_1 suffices [10, 38], but in fact it does not matter that the ordinal is ω_1. What matters is that an ordinal exists where the iteration of the defining functional of $\Downarrow$ stabilises.

But how is this reflected in the internal language? If our internal language is just higher-order logic together with the $\triangleright$ modality it is not visible. In fact, there is a *logical functor* from $\mathbf{Sh}(\omega_1)$ to $\mathbf{PSh}(\omega)$ which moreover preserves $\triangleright$, so just in this fragment we cannot hope to find a property of the internal language of $\mathbf{Sh}(\omega_1)$ that would allow us to prove adequacy.

Recall that with step-indexed logical relations we construct approximations of the desired relations on values and expressions. Two expressions are then related if they are related for *all* steps. It is this ability to say that two expressions are related for *all* steps that we need to bring into the internal language. The $\square$ modality introduced in the paper achieves this and it is in connection with this modality that we find a property of $\mathbf{Sh}(\omega_1)$ that does not hold in $\mathbf{PSh}(\omega)$.

The main result of the paper is the introduction of the $\square$ modality and the proof of soundness and completeness of the constructed logical relation entirely in the internal language of $\mathbf{Sh}(\omega_1)$.

Chapter 4: Programming and Reasoning with Guarded Recursion for Coinductive Types

This chapter is a version of [31] extended with an appendix which provides proofs of some claims made in the paper and also an extension of the calculus with sums.

In this paper we consider a simply typed λ-calculus extended with two modalities, $\blacktriangleright$ and $\blacksquare$ together with the ability to form *guarded* recursive types. Guarded recursive types are those where all occurrences of the relevant type variable appear only under the $\blacktriangleright$ modality.

We develop an operational semantics and prove normalisation. We also show that the topos $\mathbf{PSh}(\omega)$ provides an *adequate* model of the calculus and we use this fact to get a program logic for reasoning about equality (contextual equivalence) of terms of the calculus. This program logic is the internal language of the presheaf topos $\mathbf{PSh}(\omega)$. In particular it contains two modalities $\triangleright$ and $\square$. The $\triangleright$ is the “later” modality used in step-indexed models and the $\square$ modality is taken almost verbatim from [26] (Chapter 3). Finally, we show that the calculus is expressive enough to encode Rutten’s behavioural differential equations [83]. The internal logic and *Löb induction* can then be used in place of coinduction in the form of bisimulations to show properties of defined functions.

My contribution to this paper consists mostly of Sections 4.4 and 4.5 on the logic and encoding of behavioural differential equations of the paper and Sections 4.B, 4.D and 4.E of the appendix. The design of the operational semantics and the proof of adequacy of the denotational semantics and normalisation was done by my coauthors.

Chapter 5: A Model of Guarded Recursion with Clock Synchronisation

This paper is a considerably extended version of [28], except that the section of the paper on the syntax of the type theory is removed since it is now superseded by Chapter 7 of this dissertation.

In brief, this paper provides semantic justification for allowing clock synchronisations. In previous work [11, 71] clock synchronisation was disallowed and so the elimination rule for clock quantification had *freshness* side-conditions. This was necessary because the models used to justify the calculi did not support clock substitution in general, but only clock *permutation* and clock weakening.

These freshness side-conditions cause problems with the calculi. In particular, it is not clear that the calculi they considered enjoy a substitution property: substituting well-typed terms (of correct types) into well-typed terms yields well-typed terms. Now, it could be that clock synchronisation is forced upon us if we wish to have some other desirable properties. This is not the case and the model construction in Chapter 5 shows that we can get all the properties required of the calculi in previous work [11, 71], but with a much simpler elimination rule for clock quantification.

The main technical idea of the new model is to replace the previous indexing posets, which were just products of the poset ω, with more refined ones which essentially build in the ability to identify, or synchronise, clocks. The crucial new technical property is Lemma 5.2.11 on page 189.

To be precise, this paper does not provide a model of the calculus in Chapter 7 and the reason for this is that we do not construct a (refinement of a) *split* PDTT-structure [51]. However we think that this is only a technical problem which can be resolved. Indeed, Chapter 6 provides a split PDTT-structure which can be used to model the subset of the calculus presented in Chapter 7 without universes. See also Section 1.5 below for more discussion and a (conjectural at this stage) solution.

Chapter 6: A Model of Guarded Recursion via Generalised Equilogical Spaces

This chapter consists of a paper [24] which is currently under review.

This paper provides another construction which can be used to model a subset of the calculus presented in [27]. It is the subset without universes. The motivation for considering this model is that the construction in [28] does not give rise to a *split* model. Moreover, there are, in the model, no productivity guarantees for functions, e.g., on coinductive streams. To be more precise, in the model in [28] the type $\mathbb{Z}^\omega \to \mathbb{Z}^\omega$, where $\mathbb{Z}^\omega$ is the type of streams encoded using clock quantification, is interpreted as the set of *all* functions from streams of integers to streams of integers, which is unsatisfactory since intuitively, all computable functions on streams should be continuous: finite amount of output should only depend on finite amount of input. And all functions definable in the calculus are computable.

The model in Chapter 6 gives such a guarantee. However we currently do not know how to extend it to model universes, although there might be a solution following a construction of Beeson [15], which would only construct *closed* universes, but would still be enough to formally show consistency of the type theory. However this belongs to future work.

A nice feature of this construction is that it is relatively simple and direct and we get a split PDTT-structure [51] by construction. Moreover, the constructions used are quite natural generalisations of constructions used in realizability models of type theory (see e.g., Jacobs [51]).

Thus, the main idea in this paper is the generalisation of the usual PER models of dependent type theory (see, for example, Jacobs [51]) to *indexed* PER models. Technically, this is phrased as a generalisation of the category PEqu of partial equilogical spaces [14], but inspection of it shows that one could take other kinds of realisers as well. The construction is closely related to Atkey and McBride's model [11] of a simply typed calculus and can be seen as a generalisation of their construction to a model of dependent type theory with guarded recursive types and clock quantifiers.

Chapter 7: Guarded Dependent Type Theory with Coinductive Types

This chapter consists of a paper [27] with an appendix providing more detailed derivations of examples.

This paper develops an extensional dependent type theory with guarded recursive types and clock quantification. It uses the model in Chapter 5 as justification for designing the rules. The paper builds on previous work [11, 71] but extends it by allowing clock synchronisation and, more importantly, by introducing a new concept of *delayed substitutions*. These are needed for working with guarded dependent types which come up, for instance, when proving properties of guarded streams.

The paper is heavily based on examples which illustrate the need for and the use of the new rules involving delayed substitutions. We do not prove any syntactic properties of the type theory like strong or weak normalisation or decidability of type checking. Indeed, type checking is undecidable, due to equality reflection, and without restrictions the calculus is not normalising, since it contains a (guarded) fixed-point combinator. Note however that the type theory is still consistent. Non-termination is only a source of problems for a type-checking algorithm and there are several proposals on how to restrict fixed-point unfolding to get a decidable typechecking algorithm, but this is future work.

I contributed to the design of the rules for delayed substitutions, developing examples which show the need for them and checking that the rules are validated by the intended model [28].

1.5 Open Problems

This section is by necessity somewhat technical and should only be read after reading the rest of the dissertation.

Probabilistic programming

The logical relation constructed in Chapter 2 seems to be useful for proving a number of equivalences, also including equivalences involving probabilistic choice. Of course, because the method is (sound and) complete with respect to contextual equivalence (and approximation) for any two contextually equivalent terms e_1 and e_2 there is a proof that they are also related by the logical relation.

However there are limitations to the proof method. There are examples of contextually equivalent programs which we are not able to show equivalent. Here is a concrete one due to Sangiorgi and Vignudelli [86, Example 5.3].

Let H and K be the terms

$$\begin{aligned} H &\equiv \texttt{let}\ x = \texttt{ref}\ 0\ \texttt{in}\ \lambda\_.(M \oplus N) \\ K &\equiv \texttt{let}\ x = \texttt{ref}\ 0\ \texttt{in}\ (\lambda\_.M) \oplus (\lambda\_.N)) \end{aligned}$$

with M and N the terms

$$\begin{aligned} M &\equiv \texttt{if}\ !x\ =\ 0\ \texttt{then}\ x := 1; \texttt{true}\ \texttt{else}\ \Omega \\ N &\equiv \texttt{if}\ !x\ =\ 0\ \texttt{then}\ x := 1; \texttt{false}\ \texttt{else}\ \Omega \end{aligned}$$

where ; is sequencing and Ω is some diverging term.

The programs H and K are contextually equivalent (in a call-by-value language with local state) for quite a subtle reason. Note that M and N share a local reference cell. They use this cell to ensure that the "user" of H or K will only get a useful result the first time she runs the programs. This prevents the usual trick of distinguishing the programs $\lambda x.P \oplus Q$ and $(\lambda x.P) \oplus (\lambda x.Q)$ (in a call-by-value language) from working and indeed Sangiorgi and Vignudelli [86] show that the programs are equivalent.

With the logical relation in Chapter 2 we are not able to show that H and K are equivalent.

Remark 1.5.1. This problem is not directly caused by probabilistic choice. The same example also shows limitations of these logical relations if $\oplus$ is just nondeterministic (angelic or demonic) choice. ♦

Davide Sangiorgi and Valeria Vignudelli's environmental bisimulations can be used to prove this particular equivalence because they are working directly with relations on distributions (equivalently, measures), whereas our logical relation only relates terms.

We have also briefly investigated defining logical relations directly on distributions, since this appeared to be useful, but in the end we did not have example equivalences that needed the expressiveness. Now we do, so it might be useful to try to work it out.

Another reason why such an extension would be useful is that if we try to extend the logical relations method to a language with primitives for continuous probability distributions, we would like to verify interesting equivalences that are used to rewrite probabilistic programs so that inference is more efficient. It appears that a lot of the really interesting examples will require us to work with relations on measures directly, for similar reasons as the example above. Behind these equivalences there are often deep theorems of probability and statistics and the method should allow us to reduce checking equivalence of programs to those theorems.

In a quite limited form this is also what happens with the construction in Chapter 2, see Section 2.D for some examples, but it appears that more is needed.

Coherence for the model of Chapter 5

As it stands, the construction in Chapter 5 does not give rise to a *split* PDTT-structure. Hence it is not immediately clear that the syntax of the type theory as developed in Chapter 7 can be interpreted while validating all the equations. Thus to make a stronger argument for consistency of the type theory we must solve this problem.

Now, coherence problems in interpretations of dependent type theory have a long history and there are now several different recipes for addressing the coherence problem [47, 56, 64].

However none of these are directly applicable in our case. Our model is a family of presheaf categories together with functors between them. Because each $\mathfrak{GR}(\Delta)$ is a presheaf category there is a well-known construction, see e.g. Hofmann [48] for details, for getting, say, a split closed comprehension category [51] or a category with families. This is not the problem.

The problem is switching between the different $\mathfrak{GR}(\Delta)$ categories. In more detail, the problem is that the "higher-order" constructions, in particular universes and (dependent) function spaces reflect the indexing poset into the objects (to see this recall the definition of exponentials in presheaf toposes which uses "Kripke" quantification to get functoriality). Thus for these higher-order constructs we only get preservation of structure up to (canonical) isomorphism.

It would not help to list all the solutions that *almost work* here. Instead, we only briefly mention construction which seems to give a split model. However the construction is quite intricate and we have not managed to verify all the details yet.

Recall that $\mathfrak{GR}$ is an indexed category and indexed categories are equivalent to (Grothendieck) fibrations. There is a quite well-known construction for replacing an arbitrary fibration with a split one due to Bénabou [16] (see also [51, Corollary 5.2.5]). Of course our fibration is already split (that is, $\mathfrak{GR}$ is already a functor, not only a pseudofunctor), but its products are not split. However the equivalent split fibration obtained by Bénabou's construction appears to give us more freedom to define all the constructions in such a way that they are preserved by clock substitution, i.e., are split as well. The details, however, are quite involved due to several different levels of indexing and thus we leave the problem of coherence for future work.

Finally, using Benabou's construction does seem less than ideal. This is a construction applicable to an arbitrary fibration, whereas ours is a very particular one, which should hopefully mean that a simpler and more principled solution is available. However we have have not found it yet.

Guarded type theory

The type theory presented in Chapter 7 is good as an extensional type theory and as an internal language of the particular model, but it currently lacks other desirable properties a type theory should have. It appears that the type theory is useful for working with coinductive types in a modular way. However, one of the original motivations for developing the type theory [22] with the $\blacktriangleright$ modality was to be able to define guarded recursive types with negative occurrences of the relevant type variable. This would allow us to define a type that is a solution to the domain equation such as (1.8) (page 12).

In the calculus as presented in Chapter 7 defining such a type is possible, however we have not yet investigated if the type theory is sufficient to use the defined type in the desired applications. If it is, then this application would be outside of the scope of other approaches to ensuring *guardedness* in a modular way, like sized-types [1, 50]. These only allow definitions of recursive types with strictly positive occurrences of the relevant type variable.

Svendsen and Birkedal [91] have used the internal language of the topos **PSh** (ω) to construct a model of their separation logic. This gives some evidence that using the type theory in Chapter 7 a similar development might be possible. However there are some important differences. Chief among them is they use higher-order logic as the language for constructing the model and using the $\triangleright$ modality of the logic seems to be substantially simpler than using the $\blacktriangleright$ modality of the type theory. The main question with regards to applications is whether the delayed substitutions and the rules for them introduced in Chapter 7 are sufficient.

The problem with trying to see whether this is the case is that the examples are complex. Verifying that the terms, which grow quite large, type-check by hand is tedious and error-prone. For this reason it would be quite useful to have a prototype type-checker, but this runs into the current problems of undecidability of type-checking of the calculus presented in Chapter 7. Hans Bugge Grathwohl and Andrea Vezzosi have made a prototype implementation of a version of the type theory presented in Chapter 7 and the type-checker they have implemented is sufficient to type-check most examples presented there. However the implemented type theory is not precisely the same as the one presented and its semantics is currently not entirely understood.

1.6 List of Publications

During my PhD studies I have coauthored eight articles, seven of which have either been published or accepted for publication and one of which is currently under review, and a set of tutorial notes on categorical logic.

Articles included in the dissertation

Revised and in some cases extended versions of the following articles are included in the dissertation.

- Step-indexed Logical Relations for Probability
  FoSSaCS 2015 [25]
  Joint work with Lars Birkedal.

- A Model of Countable Nondeterminism in Guarded Type Theory
  RTA-TLCA 2014 [26]
  Joint work with Lars Birkedal and Marino Miculan.

- Programming and Reasoning with Guarded Recursion for Coinductive Types
  FoSSaCS 2015 [31]
  Joint work with Lars Birkedal, Ranald Clouston, and Hans Bugge Grathwohl.

- A Model of Guarded Recursion with Clock Synchronisation
  MFPS 2015 [28]
  Joint work with Rasmus Ejlers Møgelberg.

- A Model of Guarded Recursion via Generalised Equilogical Spaces
  Submitted (currently under review)
  Joint work with Lars Birkedal.

- Guarded Dependent Type Theory with Coinductive Types
  FoSSaCS 2016 [27]
  Joint work with Lars Birkedal, Ranald Clouston, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg.

Articles not included in the dissertation

The following articles are not included in the dissertation because my contributions to them are minor.

- Step-Indexed Relational Reasoning for Countable Nondeterminism
  Logical Methods in Computer Science [23]
  Joint work with Lars Birkedal and Jan Schwinghammer.

- ModuRes: a Coq Library for Modular Reasoning about Concurrent Higher-Order Imperative Programming Languages
  ITP 2015 [89]
  Joint work with Lars Birkedal and Filip Sieczkowski.

Tutorial notes

I wrote the following tutorial notes together with Lars Birkedal.

- A Taste of Categorical Logic - Tutorial Notes

They are available online at

http://cs.au.dk/~abizjak/tutorials/1-categorical-logic/.

1.7 Notations

The ends of Theorem, Lemma, Proposition and Corollary environments are marked with the symbol $\lozenge$. The ends of Definition, Example and Remark environments are marked with the symbol ♦. The ends of proofs are marked with $\mathfrak{QED}$.

We use $\mathbb{N}$ for the set of natural numbers, $\mathbb{Z}$ for the set of integers ans $\mathbb{R}$ for the set of real numbers.

Following is the list of some common notation that are used throughout the dissertation.

- $f[A]$ image of the set A under the function f.

- $\operatorname{im} f$ is the range of the function f, i.e., the image of the domain.

- $f^{-1}[A]$ is the preimage of the set A under the function f.

- $A \subseteq^{\text{fin}} B$ means that the set A is a finite subset of the set B.

- B^A and $A \Rightarrow B$ are used for exponentials (in the relevant category).
- $A \to_{\text{fin}} B$ is the set of finite maps from the set A to the set B, i.e., the set

  $$\bigsqcup_{D \subseteq^{\text{fin}} A} B^D$$

  where $\bigsqcup$ is the disjoint union.
- $\mathbf{PSh}(\mathbb{C})$ is the category of presheaves on the small category $\mathbb{C}$.
- $\mathbf{Sh}(P)$ for a poset P is the category of sheaves on P considered as a topological space with the (downwards) Alexandrov topology: opens are downwards closed sets.
- 1 is the terminal object of the relevant category.
- 0 is the initial object of the relevant category.
- ! is the unique morphism *into* the terminal object or the unique morphism *out of* the initial object.
- $\star$ is the unique element of the chosen terminal object in the category **Set**.
- f^n for a morphism $f : X \to X$ and natural number n is the n-th iteration of f: f^0 is the identity function and $f^{n+1} = f \circ f^n$.
- $\bot$ and $\top$ are, respectively, the least and greatest element of the given poset.
- $\mathcal{P}(X)$ is the power set of the set X and more generally, the power object of the object X in a topos.

Part II

Publications

Chapter 2

Step-Indexed Logical Relations for Probability

This chapter is a revised version of

> Aleš Bizjak and Lars Birkedal.
>
> Step-indexed logical relations for probability.
>
> In Andrew Pitts, editor, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pages 279–294. Springer-VS, 2015.

Abstract

It is well-known that constructing models of higher-order probabilistic programming languages is challenging. We show how to construct step-indexed logical relations for a probabilistic extension of a higher-order programming language with impredicative polymorphism and recursive types. We show that the resulting logical relation is sound and complete with respect to the contextual preorder and, moreover, that it is convenient for reasoning about concrete program equivalences. Finally, we extend the language with dynamically allocated first-order references and show how to extend the logical relation to this language. We show that the resulting relation remains useful for reasoning about examples involving both state and probabilistic choice.

2.1 Introduction

It is well known that it is challenging to develop techniques for reasoning about programs written in probabilistic higher-order programming languages. A probabilistic program evaluates to a distribution of values, as opposed to a set of values in the case of nondeterminism or a single value in the case of deterministic computation. Probability distributions form a monad. This observation has been used as a basis for several denotational domain-theoretic models of probabilistic languages and also as a guide for designing probabilistic languages with monadic types [54, 81, 85]. Game semantics has also been used to give models of probabilistic programming languages [37, 41] and a fully abstract model using coherence spaces for PCF with probabilistic choice was recently presented [42].

The majority of models of probabilistic programming languages have been developed using denotational semantics. However, Johann et.al. [52] developed operationally-based logical relations for a polymorphic programming language with effects. Two of the effects they considered were probabilistic choice and *global* ground store. However, as pointed out by the authors [52], extending their construction to local store and, in particular, higher-order local store, is likely to be problematic. Recently, operationally-based bisimulation techniques have been extended to probabilistic extensions of PCF [35, 36]. The operational semantics of probabilistic higher-order programming languages has been investigated in [58].

Step-indexed logical relations [5, 8] have proved to be a useful method for proving contextual approximation and equivalence for programming languages with a wide range of features, including computational effects.

In this paper we show how to extend the method of step-indexed logical relations to reason about contextual approximation and equivalence of probabilistic higher-order programs. To define the logical relation we employ biorthogonality [76, 79] and step-indexing. Biorthogonality is used to ensure completeness of the logical relation with respect to contextual equivalence, but it also makes it possible to keep the value relations simple, see Figure 2.1. Moreover, the definition using biorthogonality makes it possible to "externalise" the reasoning in many cases when proving example equivalences. By this we mean that the reasoning reduces to algebraic manipulations of probabilities. This way, the quantitative aspects do not complicate the reasoning much, compared to the usual reasoning with step-indexed logical relations. To define the biorthogonal lifting we use two notions of observation; the termination probability and its stratified version approximating it. We define these and prove the required properties in Section 2.3.

We develop our step-indexed logical relations for the call-by-value language $\mathbf{F}^{\mu,\oplus}$. This is system $\mathbf{F}$ with recursive types, extended with a single probabilistic choice primitive $\texttt{rand}$. The primitive $\texttt{rand}$ takes a natural number n and reduces with uniform probability to one of $1, 2, \ldots, n$. Thus $\texttt{rand}\, n$ represents the uniform probability distribution on the set $\{1, 2, \ldots, n\}$. We choose to add $\texttt{rand}$ instead of just a single coin flip primitive to make the examples easier to write.

To show that the model is useful we use it to prove some example equivalences in Section 2.5. We show two examples based on parametricity. In the first example, we characterise elements of the universal type $\forall\alpha.\alpha \to \alpha$. In a deterministic language, and even in a language with nondeterministic choice, the only interesting element of this type is the identity function. However, since in a probabilistic language we not only observe the end result, but also the likelihood with which it is returned, it turns out that there are many more elements. Concretely, we show that the elements of the type $\forall\alpha.\alpha \to \alpha$ that are of the form $\Lambda\alpha.\lambda x.e$, correspond precisely to *left-computable* real numbers in the interval $[0,1]$. In the second example we show a free theorem involving functions on lists. We show additional equivalences in Section 2.D in the appendix, including the correctness of von Neumann's procedure for generating a fair sequence of coin tosses from an unfair coin, and some equivalences from the recent papers using bisimulations [35, 36].

We add dynamically allocated references to the language and extend the logical relation to the new language in Section 2.6. For simplicity we only sketch how to extend the construction with first-order state. This already suggests that an extension with general references can be done in the usual way for step-indexed logical relations. We conclude the section by proving a representation independence result involving both state and probabilistic choice.

2.2 The Language $\mathbf{F}^{\mu,\oplus}$

The language is a standard pure functional language with recursive, universal and existential types with an additional choice primitive `rand`. The base types include the type of natural numbers `nat` with some primitive operations. The grammar of terms e is

$$\begin{aligned} e ::= {} & x \mid \langle\rangle \mid \mathtt{rand}\, e \mid \underline{n} \mid \mathtt{if}_1\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \mid \mathtt{P}\, e \mid \mathtt{S}\, e \mid \langle e_1, e_2\rangle \mid \mathtt{proj}_i\, e \\ & \mid \lambda x.e \mid e_1\, e_2 \mid \mathtt{inl}\ e \mid \mathtt{inr}\ e \mid \mathtt{match}(e, x_1.e_1, x_2.e_2) \mid \Lambda.e \mid e[\,] \\ & \mid \mathtt{pack}\, e \mid \mathtt{unpack}\ e_1\ \mathtt{as}\ x\ \mathtt{in}\ e_2 \mid \mathtt{fold}\, e \mid \mathtt{unfold}\, e \end{aligned}$$

We write $\underline{n}$ for the numeral representing the natural number n and `S` and `P` are the successor and predecessor functions, respectively. For convenience, numerals start at $\underline{1}$. Given a numeral $\underline{n}$, the term $\mathtt{rand}\,\underline{n}$ evaluates to one of the numerals $\underline{1},\ldots,\underline{n}$ with uniform probability. There are no types in the syntax of terms, e.g., instead of $\Lambda\alpha.e$ and $e\,\tau$ we have $\Lambda.e$ and $e[\,]$. This is for convenience only.

We write $\alpha, \beta, \ldots$ for *type variables* and $x, y, \ldots$ for *term variables*. The notation $\tau[\vec{\tau}/\vec{\alpha}]$ denotes the simultaneous capture-avoiding substitution of types $\vec{\tau}$ for the free type variables $\vec{\alpha}$ in the type τ; $e[\vec{v}/\vec{x}]$ denotes simultaneous capture-avoiding substitution of values $\vec{v}$ for the free term variables $\vec{x}$ in the term e.

We write **Stk** for the set of evaluation contexts given by the call-by-value reduction strategy. Given two evaluation contexts E, E' we define their composition $E \circ E'$ by induction on E in the natural way. Given an evaluation context E and expression e we write $E[e]$ for the term obtained by plugging

e into E. For any two evaluation contexts E and E' and a term e we have $E[E'[e]] = (E \circ E')[e]$.

For a type variable context Δ, the judgement $\Delta \vdash \tau$ expresses that the free type variables in τ are included in Δ. The typing judgements are entirely standard with the addition of the typing of `rand` which is given by the rule

$$\frac{\Delta \mid \Gamma \vdash e : \mathtt{nat}}{\Delta \mid \Gamma \vdash \mathtt{rand}\, e : \mathtt{nat}}.$$

The complete set of typing rules are in Figure 2.4 on page 57 in the appendix. We write $\mathcal{T}(\Delta)$ for the set of types well-formed in context Δ, and $\mathcal{T}$ for the set of *closed* types τ. We write $\mathbf{Val}\,(\tau)$ and $\mathbf{Tm}\,(\tau)$ for the sets of *closed* values and terms of type τ, respectively. We write **Val** and **Tm** for the set of *all*[1] *closed* values and closed terms, respectively. $\mathbf{Stk}\,(\tau)$ denotes the set of τ-accepting evaluation contexts, i.e., evaluation contexts E, such that given any closed term e of type τ, $E[e]$ is a typeable term. **Stk** denotes the set of all evaluation contexts.

For a typing context $\Gamma = x_1{:}\tau_1, \ldots, x_n{:}\tau_n$ with $\tau_1, \ldots, \tau_n \in \mathcal{T}$, let $\mathbf{Subst}(\Gamma)$ denote the set of type-respecting value substitutions, i.e. for all i, $\gamma(x_i) \in \mathbf{Val}\,(\tau_i)$. In particular, if $\Delta \mid \Gamma \vdash e : \tau$ then $\varnothing \mid \varnothing \vdash e\gamma : \tau\delta$ for any $\delta \in \mathcal{T}^{\Delta}$ and $\gamma \in \mathbf{Subst}(\Gamma\delta)$, and the type system satisfies standard properties of progress and preservation and a canonical forms lemma.

The operational semantics of the language is a standard call-by-value semantics but weighted with $p \in [0,1]$ which denotes the likelihood of that reduction. We write $\overset{p}{\rightsquigarrow}$ for the one-step reduction relation. All the usual β reductions have weight equal to 1 and the reduction from $\mathtt{rand}\,\underline{n}$ is

$$\mathtt{rand}\,\underline{n} \overset{\frac{1}{n}}{\rightsquigarrow} \underline{k} \qquad \text{for } k \in \{1, 2, \ldots, n\}.$$

The rest of the rules are given in Figure 2.5 on page 58 in the appendix. The operational semantics thus gives rise to a Markov chain with closed terms as states. In particular for each term e we have $\sum_{e' \mid e \overset{p}{\rightsquigarrow} e'} p \leq 1$.

2.3 Observations and Biorthogonality

We will use biorthogonality to define the logical relation. This section provides the necessary observation predicates used in the definition of the biorthogonal lifting of value relations to expression relations. Because of the use of biorthogonality the value relations (see Figure 2.1 on page 45) remain as simple as for a language without probabilistic choice. The new quantitative aspects only appear in the definition of the biorthogonal lifting ($\top\top$-closure) defined in Section 2.4. Two kinds of observations are used. The probability of

[1] In particular, we do not require them to be typeable.

termination, $\mathfrak{P}^{\Downarrow}(e)$, which is the actual probability that e terminates, and its approximation, the *stratified* termination probability $\mathfrak{P}^{\Downarrow}_{k}(e)$, where $k \in \mathbb{N}$ denotes, intuitively, the number of computation steps. The stratified termination probability provides the link between steps in the operational semantics and the indexing in the definition of the interpretation of types.

The probability of termination, $\mathfrak{P}^{\Downarrow}(\cdot)$, is a function of type $\mathbf{Tm} \to \mathcal{I}$ where $\mathcal{I}$ is the unit interval $[0,1]$. Since $\mathcal{I}$ is a pointed ω-cpo for the usual order, so is the space of all functions $\mathbf{Tm} \to \mathcal{I}$ with pointwise ordering. We define $\mathfrak{P}^{\Downarrow}(\cdot)$ as a fixed point of the continuous function Φ on this ω-cpo: Let $\mathcal{F} = \mathbf{Tm} \to \mathcal{I}$ and define $\Phi : \mathcal{F} \to \mathcal{F}$ as

$$\Phi(f)(e) = \begin{cases} 1 & \text{if } e \in \mathbf{Val} \\ \displaystyle\sum_{e \overset{p}{\rightsquigarrow} e'} p \cdot f(e') & \text{otherwise} \end{cases}$$

Note that if e is stuck then $\Phi(f)(e) = 0$ since the empty sum is by definition 0.

The function Φ is monotone and preserves suprema of ω-chains. The proof is straightforward and can be found in Section 2.A in the appendix. Thus Φ has a least fixed point in $\mathcal{F}$ and we denote this fixed point by $\mathfrak{P}^{\Downarrow}(\cdot)$, i.e., $\mathfrak{P}^{\Downarrow}(e) = \sup_{n\in\omega} \Phi^{n}(\bot)(e)$.

To define the stratified observations we need the notion of a path. Given terms e and e' a path π from e to e', written $\pi : e \rightsquigarrow^{*} e'$, is a sequence $e \overset{p_1}{\rightsquigarrow} e_1 \overset{p_2}{\rightsquigarrow} e_2 \overset{p_3}{\rightsquigarrow} \cdots \overset{p_n}{\rightsquigarrow} e'$. The *weight* $\mathfrak{W}(\pi)$ of a path π is the product of the weights of reductions in π. We write $\mathfrak{R}$ for the set of all paths and $\cdot$ for their concatenation (when defined). For a non-empty path $\pi \in \mathfrak{R}$ we write $\ell(\pi)$ for its last expression.

Note that the fact that the weight of the path is the product builds in the assumption that successive probabilistic choices are independent.

We call reductions of the form $\texttt{unfold}(\texttt{fold}\, v) \overset{1}{\rightsquigarrow} v$ *unfold-fold* reductions and reductions of the form $\texttt{rand}\,\underline{n} \overset{\frac{1}{n}}{\rightsquigarrow} \underline{k}$ *choice* reductions. If *none* of the reductions in a path π is a choice reduction we call π *choice-free* and similarly if none of the reductions in π is an unfold-fold reductions we call π *unfold-fold free*.

We define the following types of multi-step reductions which we use in the definition of the logical relation.

- $e \overset{\text{cf}}{\Longrightarrow} e'$ if there is a *choice-free* path from e to e'
- $e \overset{\text{uff}}{\Longrightarrow} e'$ if there is an *unfold-fold* free path from e to e'.
- $e \overset{\text{cuff}}{\Longrightarrow} e'$ if $e \overset{\text{cf}}{\Longrightarrow} e'$ and $e \overset{\text{uff}}{\Longrightarrow} e'$.

The following useful lemma states that all but choice reductions preserve the probability of termination. As a consequence, we will see that all but choice reductions preserve equivalence.

Lemma 2.3.1. *Let $e, e' \in \mathbf{Tm}$ and $e \overset{cf}{\Longrightarrow} e'$. Then $\mathfrak{P}^{\Downarrow}(e) = \mathfrak{P}^{\Downarrow}(e')$.* ◊

The proof proceeds on the length of the reduction path with the strengthened induction hypothesis stating that the probabilities of termination of all elements on the path are the same. To define the stratified probability of termination that approximates $\mathfrak{P}^{\Downarrow}(\cdot)$ we need an auxiliary notion.

Definition 2.3.2. For a closed expression $e \in \mathbf{Tm}$ we define $\mathbf{Red}(e)$ as the (unique) set of paths containing *exactly one* unfold-fold or choice reduction and *ending* with such a reduction. More precisely, we define the function $\mathbf{Red} : \mathbf{Tm} \to \mathcal{P}(\mathfrak{R})$ as the least function satisfying

$$\mathbf{Red}(e) = \begin{cases} \{e \overset{1}{\rightsquigarrow} e'\} & \text{if } e = E[\mathtt{unfold}\,(\mathtt{fold}\,v)] \\ \{e \overset{p}{\rightsquigarrow} E[\underline{k}] \mid p = \frac{1}{n}, k \in \{1, 2, \ldots, n\}\} & \text{if } e = E[\mathtt{rand}\,\underline{n}] \\ \left\{(e \overset{1}{\rightsquigarrow} e') \cdot \pi \,\middle|\, \pi \in \mathbf{Red}(e')\right\} & \text{if } e \overset{1}{\rightsquigarrow} e' \text{ and } e \overset{\text{cuff}}{\Longrightarrow} e' \\ \emptyset & \text{otherwise} \end{cases}$$

where we order the power set $\mathcal{P}(\mathfrak{R})$ by subset inclusion. ♦

Using $\mathbf{Red}(\cdot)$ we define a monotone map $\Psi : \mathcal{F} \to \mathcal{F}$ that preserves ω-chains.

$$\Psi(f)(e) = \begin{cases} 1 & \text{if } \exists v \in \mathbf{Val}, e \overset{\text{cuff}}{\Longrightarrow} v \\ \displaystyle\sum_{\pi \in \mathbf{Red}(e)} \mathfrak{W}(\pi) \cdot f(\ell(\pi)) & \text{otherwise} \end{cases}$$

and then define $\mathfrak{P}_k^{\Downarrow}(e) = \Psi^k(\bot)(e)$. The intended meaning of $\mathfrak{P}_k^{\Downarrow}(e)$ is the probability that e terminates within k unfold-fold and choice reductions. Since Ψ is monotone we have that $\mathfrak{P}_k^{\Downarrow}(e) \leq \mathfrak{P}_{k+1}^{\Downarrow}(e)$ for any k and e.

The following lemma is the reason for counting only certain reductions, cf. [39]. It allows us to stay at the same step-index even when taking steps in the operational semantics. As a consequence we will get a more extensional logical relation.

Lemma 2.3.3. *Let $e, e' \in \mathbf{Tm}$. If $e \overset{cuff}{\Longrightarrow} e'$ then for all k, $\mathfrak{P}_k^{\Downarrow}(e) = \mathfrak{P}_k^{\Downarrow}(e')$.* ◊

Proof. When k is 0 the result is immediate. So assume $k > 0$. We need to distinguish two cases.

- If there exists $v' \in \mathbf{Val}$ such that $e' \overset{\text{cuff}}{\Longrightarrow} v'$ then we also have $e \overset{\text{cuff}}{\Longrightarrow} v'$ and we are done.

- If not, then we need to inspect the definition of $\mathbf{Red}(e)$ and $\mathbf{Red}(e')$. It is easy to see that any path $\pi \in \mathbf{Red}(e')$ corresponds to a unique path $\pi' \cdot \pi$ in $\mathbf{Red}(e)$. It is similarly easy to see that $\mathfrak{W}(\pi) = \mathfrak{W}(\pi' \cdot \pi)$ and that $\ell(\pi) = \ell(\pi' \cdot \pi)$. Thus we have that $\mathfrak{P}_k^{\Downarrow}(e) = \mathfrak{P}_k^{\Downarrow}(e')$.

$\mathfrak{QED}$

The following is immediate from the definition of the chain $\left\{\mathfrak{P}_k^{\Downarrow}(e)\right\}_{k=0}^{\infty}$ and the fact that $\mathtt{rand}\,\underline{n}$ reduces with uniform probability.

Lemma 2.3.4. *Let e be a closed term. If $e \overset{1}{\rightsquigarrow} e'$ and the reduction is an unfold-fold reduction then $\mathfrak{P}_{k+1}^{\Downarrow}(e) = \mathfrak{P}_k^{\Downarrow}(e')$. If the reduction from e is a choice reduction, then $\mathfrak{P}_{k+1}^{\Downarrow}(e) = \frac{1}{|\mathbf{Red}(e)|}\sum_{\pi\in\mathbf{Red}(e)} \mathfrak{P}_k^{\Downarrow}(\ell(\pi))$.* ◊

The following proposition is needed to prove adequacy of the logical relation with respect to contextual equivalence. It is analogous to the property used to prove adequacy of step-indexed logical relations for deterministic and nondeterministic languages. Consider the case of may-equivalence. To prove adequacy in this case (cf. [23, Theorem 4.8]) we use the fact that if e may-terminates, then there is a natural number n such that e terminates in n steps. This property does not hold in the probabilistic case, but the property analogous to it that is sufficient to prove adequacy still holds.

Proposition 2.3.5. *For each $e \in \mathbf{Tm}$ we have $\mathfrak{P}^{\Downarrow}(e) \leq \sup_{k\in\omega}\left(\mathfrak{P}_k^{\Downarrow}(e)\right)$.* ◊

Proof. We use Scott induction. Let $\mathcal{S}$ be the set

$$\mathcal{S} = \left\{ f \in \mathcal{F} \;\middle|\; \forall e, f(e) \leq \sup_{k\in\omega}\left(\mathfrak{P}_k^{\Downarrow}(e)\right)\right\}$$

It is easy to see that $\mathcal{S}$ is closed under limits of ω-chains and that $\perp \in \mathcal{S}$. The last property to check is that $\mathcal{S}$ is closed under Φ. Let $f \in \mathcal{S}$ and e an expression. We have

$$\Phi(f)(e) = \begin{cases} 1 & \text{if } e \in \mathbf{Val} \\ \displaystyle\sum_{e \overset{p}{\rightsquigarrow} e'} p \cdot f(e') & \text{otherwise} \end{cases}$$

and we consider 4 cases.

- $e \in \mathbf{Val}$. We always have $e \overset{\text{cuff}}{\Longrightarrow} e$ and so we have that for any $k > 0$, $\mathfrak{P}_k^{\Downarrow}(e) = 1$ which is the top element.

- $e \overset{p}{\rightsquigarrow} e'$ and the reduction is not unfold-fold or choice. Then we use Lemma 2.3.3 to get $\mathfrak{P}_k^{\Downarrow}(e) = \mathfrak{P}_k^{\Downarrow}(e')$ for all k. Similarly we have that $\Phi(f)(e) = f(e')$ from the definition of Φ. Thus we can use the assumption that $f \in \mathcal{S}$.

- $e \overset{1}{\rightsquigarrow} e'$ and the reduction is unfold-fold. This follows directly from the definition of $\mathbf{Red}(\cdot)$, Ψ and the assumption that $f \in \mathcal{S}$.
- The reduction from e is a choice reduction. Suppose e reduces to the terms $e_1, e_2, \ldots, e_n$. Then we know from the operational semantics that the weights are all $\frac{1}{n}$. We get

$$\Phi(f)(e) = \sum_{i=1}^{n} \frac{1}{n} f(e_i) \qquad \text{and} \qquad \mathfrak{P}_{k+1}^{\Downarrow}(e) = \sum_{i=1}^{n} \frac{1}{n} \mathfrak{P}_{k}^{\Downarrow}(e_i). \tag{2.1}$$

Using the fact that $\mathfrak{P}_{k}^{\Downarrow}(e_i)$ is an increasing chain in k for each e_i we have

$$\sup_{k \in \omega} \left(\mathfrak{P}_{k}^{\Downarrow}(e) \right) = \sum_{i=1}^{n} \frac{1}{n} \sup_{k \in \omega} \left(\mathfrak{P}_{k}^{\Downarrow}(e_i) \right) \tag{2.2}$$

By assumption $f(e_i) \le \sup_{k \in \omega} \left(\mathfrak{P}_{k}^{\Downarrow}(e_i) \right)$ for all $i \in \{1, 2, \ldots, n\}$ which concludes the proof using (2.1) and (2.2). $\mathfrak{QED}$

2.4 Logical, CIU and Contextual Approximation Relations

The contextual and CIU (**c**losed **i**nstantiations of **u**ses [78]) approximations are defined in a way analogous to the one for deterministic programming languages. We require some auxiliary notions. A *type-indexed relation* $\mathcal{R}$ is a set of tuples $(\Delta, \Gamma, e, e', \tau)$ such that $\Delta \vdash \Gamma$ and $\Delta \vdash \tau$ and $\Delta \mid \Gamma \vdash e : \tau$ and $\Delta \mid \Gamma \vdash e' : \tau$. We write $\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau$ for $(\Delta, \Gamma, e, e', \tau) \in \mathcal{R}$.

Definition 2.4.1 (Precongruence)**.** A type-indexed relation $\mathcal{R}$ is *reflexive* if $\Delta \mid \Gamma \vdash e : \tau$ implies $\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e : \tau$. It is *transitive* if $\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau$ and $\Delta \mid \Gamma \vdash e' \mathrel{\mathcal{R}} e'' : \tau$ implies $\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e'' : \tau$. It is *compatible* if it is closed under the term forming rules, e.g., [2]

$$\frac{\Delta \mid \Gamma, x{:}\tau_1 \vdash e \mathrel{\mathcal{R}} e' : \tau_2}{\Delta \mid \Gamma \vdash \lambda x.e \mathrel{\mathcal{R}} \lambda x.e' : \tau_1 \to \tau_2} \qquad \frac{\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \mathtt{nat}}{\Delta \mid \Gamma \vdash \mathtt{rand}\, e \mathrel{\mathcal{R}} \mathtt{rand}\, e' : \mathtt{nat}}$$

A *precongruence* is a reflexive, transitive and compatible type-indexed relation. ♦

Finally, to relate the operational semantics with the relation we have the notion of adequacy. In the deterministic case, a relation $\mathcal{R}$ is adequate if when $e \mathrel{\mathcal{R}} e'$ are two related closed terms, then if e terminates so does e'. Here we need to compare probabilities of termination instead, since these are our observations.

[2] We only show a few rules, the rest are analogous and can be found in Figure 2.6 on page 77 in the appendix.

Definition 2.4.2. A type-indexed relation $\mathcal{R}$ is *adequate* if for all e, e' such that $\emptyset \mid \emptyset \vdash e \mathcal{R} e' : \tau$ we have $\mathfrak{P}^{\Downarrow}(e) \leq \mathfrak{P}^{\Downarrow}(e')$. ♦

The *contextual approximation relation*, written $\Delta \mid \Gamma \vdash e \lesssim^{ctx} e' : \tau$, is defined as the *largest adequate precongruence* and the *CIU approximation relation*, written $\Delta \mid \Gamma \vdash e \lesssim^{CIU} e' : \tau$, is defined using evaluation contexts in the usual way, e.g. [78], using $\mathfrak{P}^{\Downarrow}(\cdot)$ for observations. The fact that the largest adequate precongruence exists is proved as in [78].

Logical relation

We now define the step-indexed logical relation. We present the construction in the elementary way with explicit indexing instead of using a logic with guarded recursion as in [39]. We do this partly to remain self-contained and partly to avoid issues with validity of manipulations of finite and infinite sums in an constructive logic. It is likely that with sufficient all theorems and examples can be constructively proved, and so would be valid in the internal logic of, e.g., the topos of trees, but this would detract needlessly from the simplicity of the construction.

Interpretations of types will be defined as decreasing sequences of relations on *typeable* values. For *closed types* τ and σ we define the sets $\mathbf{VRel}(\tau,\sigma)$, $\mathbf{SRel}(\tau,\sigma)$ and $\mathbf{TRel}(\tau,\sigma)$ to be the sets of decreasing sequences of relations on typeable values, evaluation contexts and expressions respectively. The types τ and σ denote the types of the left-hand side and the right-hand side respectively, i.e. if $(v,u) \in \varphi(n)$ for $\varphi \in \mathbf{VRel}(\tau,\sigma)$ then v has type τ and u has type σ. The order relation $\leq$ on these sets is defined pointwise, e.g. for $\varphi, \psi \in \mathbf{VRel}(\tau,\sigma)$ we write $\varphi \leq \psi$ if $\forall n \in \mathbb{N}, \varphi(n) \subseteq \psi(n)$. We implicitly use the inclusion from $\mathbf{VRel}(\tau,\sigma)$ to $\mathbf{TRel}(\tau,\sigma)$. The reason for having relations on values and terms of different types on the left and right-hand sides is so we are able to prove parametricity properties in Section 2.5.

We define maps $\cdot^{\top}_{\tau,\sigma} : \mathbf{VRel}(\tau,\sigma) \to \mathbf{SRel}(\tau,\sigma)$ and $\cdot^{\perp}_{\tau,\sigma} : \mathbf{SRel}(\tau,\sigma) \to \mathbf{TRel}(\tau,\sigma)$. We usually omit the type indices when they can be inferred from the context. The maps are defined as follows

$$r^{\top}_{\tau,\sigma}(n) = \left\{(E,E') \,\middle|\, \forall k \leq n, \forall (v,v') \in r(k), \mathfrak{P}^{\Downarrow}_{k}(E[v]) \leq \mathfrak{P}^{\Downarrow}(E'[v'])\right\}$$

and

$$r^{\perp}_{\tau,\sigma}(n) = \left\{(e,e') \,\middle|\, \forall k \leq n, \forall (E,E') \in r(k), \mathfrak{P}^{\Downarrow}_{k}(E[e]) \leq \mathfrak{P}^{\Downarrow}(E'[e'])\right\}.$$

Note that we only count steps evaluating the left term in defining $r^{\top}$ and $r^{\perp}$. We write $r^{\top\top} = r^{\top\perp}$ for their composition from $\mathbf{VRel}(\tau,\sigma)$ to $\mathbf{TRel}(\tau,\sigma)$. The function $\cdot^{\top}$ is order-reversing and $\cdot^{\top\top}$ is order-preserving and inflationary.

Lemma 2.4.3. *Let τ,σ be closed types and $r, s \in \mathbf{VRel}(\tau,\sigma)$. Then $r \leq r^{\top\top}$ and if $r \leq s$ then $s^{\top} \leq r^{\top}$ and $r^{\top\top} \leq s^{\top\top}$.* ◊

For a type-variable context Δ we define $\mathbf{VRel}(\Delta)$ using $\mathbf{VRel}(\cdot,\cdot)$ as

$$\mathbf{VRel}(\Delta) = \left\{(\varphi_1, \varphi_2, \varphi_r) \,\middle|\, \varphi_1, \varphi_2 \in \mathfrak{T}^{\Delta}, \forall \alpha \in \Delta, \varphi_r(\alpha) \in \mathbf{VRel}(\varphi_1(\alpha), \varphi_2(\alpha))\right\}$$

where the first two components give syntactic types for the left and right hand sides of the relation and the third component is a relation between those types.

The interpretation of types, $[\![\cdot \vdash \cdot]\!]$ is by induction on the judgement $\Delta \vdash \tau$ (this judgement is defined precisely in Figure 2.3 on page 56 in the appendix). For a judgement $\Delta \vdash \tau$ and $\varphi \in \mathbf{VRel}(\Delta)$ we have

$$[\![\Delta \vdash \tau]\!](\varphi) \in \mathbf{VRel}(\varphi_1(\tau), \varphi_2(\tau))$$

where the φ_1 and φ_2 are the first two components of φ and $\varphi_1(\tau)$ denotes substitution. Moreover $[\![\cdot]\!]$ is *non-expansive* in the sense that $[\![\Delta \vdash \tau]\!](\varphi)(n)$ can depend only on the values of $\varphi_r(\alpha)(k)$ for $k \leq n$, see [21] for this metric view of step-indexing. The interpretation of types is defined in Figure 2.1. Observe that the value relations are as simple as for a language without probabilistic choice. The crucial difference is hidden in the $\top\top$-closure of value relations.

Context extension lemmas To prove soundness and completeness we need lemmas stating how extending evaluation contexts preserves relatedness. We only show the case for `rand`. The rest are similarly simple.

Lemma 2.4.4. *Let $n \in \mathbb{N}$. If $(E, E') \in [\![\Delta \vdash \mathtt{nat}]\!](\varphi)^{\top}(n)$ are related evaluation contexts then $(E \circ (\mathtt{rand}[\,]), E' \circ (\mathtt{rand}[\,])) \in [\![\Delta \vdash \mathtt{nat}]\!](\varphi)^{\top}(n)$.* ◊

Proof. Let $n \in \mathbb{N}$ and $(v, v') \in [\![\Delta \vdash \tau]\!](\varphi)(n)$. By construction we have $v = v' = \underline{m}$ for some $m \in \mathbb{N}$, $m \geq 1$. Let $k \leq n$. If $k = 0$ the result is immediate, so assume $k = \ell + 1$. Using Lemma 2.3.4 we have $\mathfrak{P}_k^{\Downarrow}(E[\mathtt{rand}\,\underline{m}]) = \frac{1}{m}\sum_{i=1}^{m} \mathfrak{P}_{\ell}^{\Downarrow}(E[\underline{i}])$ and using the assumption $(E, E') \in [\![\Delta \vdash \mathtt{nat}]\!](\varphi)^{\top}(n)$, the fact that $k \leq n$ and monotonicity in the step-index the latter term is less than $\frac{1}{m}\sum_{i=1}^{m} \mathfrak{P}^{\Downarrow}(E'[\underline{i}])$ which by definition of $\mathfrak{P}^{\Downarrow}(\cdot)$ is equal to $\mathfrak{P}^{\Downarrow}(E'[\mathtt{rand}\,\underline{m}])$. $\mathfrak{QED}$

We define the logical approximation relation for open terms given the interpretations of types in Figure 2.1. We define $\Delta \mid \Gamma \vdash e \lesssim^{log} e' : \tau$ to mean

$$\forall n \in \mathbb{N}, \forall \varphi \in \mathbf{VRel}(\Delta), \forall (\gamma, \gamma') \in [\![\Delta \vdash \Gamma]\!](\varphi)(n), (e\gamma, e'\gamma) \in [\![\Delta \vdash \tau]\!]\varphi^{\top\top}(n).$$

Here $[\![\Delta \vdash \Gamma]\!]$ is the obvious extension of interpretation of types to interpretation of contexts which relates substitutions, mapping variables to related values. We have

Proposition 2.4.5 (Fundamental property)**.** *The logical approximation relation $\lesssim^{log}$ is compatible. In particular it is reflexive.* ◊

$$
\begin{aligned}
[\![\Delta \vdash \mathtt{nat}]\!](\varphi)(n) &= \{(\underline{k},\underline{k}) \mid k \in \mathbb{N}, k > 0\} \\
[\![\Delta \vdash \tau \times \sigma]\!](\varphi)(n) &= \left\{ (\langle v,u\rangle, \langle v',u'\rangle) \;\middle|\; \begin{array}{l} (v,v') \in [\![\Delta \vdash \tau]\!](\varphi)(n), \\ (u,u') \in [\![\Delta \vdash \sigma]\!](\varphi)(n) \end{array} \right\} \\
[\![\Delta \vdash \tau + \sigma]\!](\varphi)(n) &= \{(\mathtt{inl}\; v, \mathtt{inl}\; v') \mid (v,v') \in [\![\Delta \vdash \tau]\!](\varphi)(n)\} \\
&\quad \cup \{(\mathtt{inr}\; v, \mathtt{inr}\; v') \mid (v,v') \in [\![\Delta \vdash \sigma]\!](\varphi)(n)\} \\
[\![\Delta \vdash \tau \to \sigma]\!](\varphi)(n) &= \left\{ \begin{array}{l} (\lambda x.e, \lambda y.e') \mid \forall j \le n, \forall (v,v') \in [\![\Delta \vdash \tau]\!](\varphi)(j), \\ \qquad\qquad ((\lambda x.e)\, v, (\lambda y.e')\, v') \in [\![\Delta \vdash \sigma]\!](\varphi)^{\top\top}(j) \end{array} \right\} \\
[\![\Delta \vdash \forall \alpha.\tau]\!](\varphi)(n) &= \left\{ \begin{array}{l} (\Lambda.e, \Lambda.e') \mid \forall \sigma, \sigma' \in \mathcal{T}, \forall r \in \mathbf{VRel}(\sigma,\sigma'), \\ \qquad (e,e') \in [\![\Delta, \alpha \vdash \tau]\!](\varphi[\alpha \mapsto r])^{\top\top}(n) \end{array} \right\} \\
[\![\Delta \vdash \exists \alpha.\tau]\!](\varphi)(n) &= \left\{ \begin{array}{l} (\mathtt{pack}\, v, \mathtt{pack}\, v') \mid \exists \sigma, \sigma' \in \mathcal{T}, \exists r \in \mathbf{VRel}(\sigma,\sigma'), \\ \qquad\qquad (v,v') \in [\![\Delta, \alpha \vdash \tau]\!](\varphi[\alpha \mapsto r])(n) \end{array} \right\} \\
[\![\Delta \vdash \mu\alpha.\tau]\!](\varphi)(0) &= \mathbf{Val}(\varphi_1(\mu\alpha.\tau)) \times \mathbf{Val}(\varphi_2(\mu\alpha.\tau)) \\
[\![\Delta \vdash \mu\alpha.\tau]\!](\varphi)(n+1) &= \left\{ \begin{array}{l} (\mathtt{fold}\, v, \mathtt{fold}\, v') \mid \\ \qquad (v,v') \in [\![\Delta, \alpha \vdash \tau]\!](\varphi[\alpha \mapsto [\![\Delta \vdash \mu\alpha.\tau]\!](\varphi)])(n) \end{array} \right\}
\end{aligned}
$$

Figure 2.1: Interpretation of types.

Proof. The proof is a simple consequence of the context extension lemmas. We show the case for $\mathtt{rand}$. We have to show that $\Delta \mid \Gamma \vdash e \lesssim^{log} e' : \mathtt{nat}$ implies $\Delta \mid \Gamma \vdash \mathtt{rand}\, e \lesssim^{log} \mathtt{rand}\, e' : \mathtt{nat}$. Let $n \in \mathbb{N}$, $\varphi \in \mathbf{VRel}(\Delta)$ and $(\gamma, \gamma') \in [\![\Delta \vdash \Gamma]\!](\varphi)(n)$. Let $f = e\gamma$ and $f' = e'\gamma'$. Then our assumption gives us

$$(f, f') \in [\![\Delta \vdash \mathtt{nat}]\!](\varphi)^{\top\top}(n) \tag{2.3}$$

and we are to show

$$(\mathtt{rand}\, f, \mathtt{rand}\, f') \in [\![\Delta \vdash \mathtt{nat}]\!](\varphi)^{\top\top}(n).$$

Let $j \le n$ and $(E, E') \in [\![\Delta \vdash \mathtt{nat}]\!](\varphi)^{\top}(j)$. From Lemma 2.4.4 we have

$$(E \circ (\mathtt{rand}\,[]), E' \circ (\mathtt{rand}\,[])) \in [\![\Delta \vdash \mathtt{nat}]\!](\varphi)^{\top}(j)$$

which suffices by the definition of the orthogonality relation and assumption (2.3). $\mathfrak{QED}$

We now want to relate logical, CIU and contextual approximation relations.

Corollary 2.4.6. *Logical approximation relation $\lesssim^{log}$ is adequate.* $\Diamond$

Proof. Assume $\varnothing \mid \varnothing \vdash e \lesssim^{log} e' : \tau$. We are to show that $\mathfrak{P}^{\Downarrow}(e) \leq \mathfrak{P}^{\Downarrow}(e')$. Straight from the definition we have $\forall n \in \mathbb{N}, (e, e') \in [\![\varnothing \vdash \tau]\!]^{\top\top}(n)$. The empty evaluation context is always related to itself (at any type). This implies

$$\forall n \in \mathbb{N}, \mathfrak{P}^{\Downarrow}_{n}(e) \leq \mathfrak{P}^{\Downarrow}(e')$$

which further implies (since the right-hand side is independent of n) that

$$\sup_{n\in\omega}\left(\mathfrak{P}^{\Downarrow}_{n}(e)\right) \leq \mathfrak{P}^{\Downarrow}(e').$$

Using Proposition 2.3.5 we thus have

$$\mathfrak{P}^{\Downarrow}(e) \leq \sup_{n\in\omega}\left(\mathfrak{P}^{\Downarrow}_{n}(e)\right) \leq \mathfrak{P}^{\Downarrow}(e')$$

concluding the proof. $\mathfrak{QED}$

We now have that the logical relation is adequate and compatible. This does not immediately imply that it is contained in the contextual approximation relation, since we do not know that it is transitive. However we have the following lemma where by transitive closure we mean that for each Δ, Γ and τ we take the transitive closure of the relation $\{(e, e') \mid \Delta \mid \Gamma \vdash e \lesssim^{log} e' : \tau\}$. This is another type-indexed relation.

Lemma 2.4.7. *The transitive closure of $\lesssim^{log}$ is compatible and adequate.* $\Diamond$

Proof. Transitive closure of an adequate relation is adequate. Similarly the transitive closure of a compatible and *reflexive* relation (in the sense of Definition 2.4.1) is again compatible (and reflexive). $\mathfrak{QED}$

To relate the logical relation to contextual and CIU approximations we first have that the composition of logical and CIU approximations is included in the logical approximation relation.

Corollary 2.4.8. *If $\Delta \mid \Gamma \vdash e \lesssim^{log} e' : \tau$ and $\Delta \mid \Gamma \vdash e' \lesssim^{CIU} e'' : \tau$ then $\Delta \mid \Gamma \vdash e \lesssim^{log} e'' : \tau$.* $\Diamond$

This follows directly from the definition of the logical relation using biorthogonality. This corollary in turn implies, together with Proposition 2.4.5 and the fact that all compatible relations are in particular reflexive, that CIU approximation relation is contained in the logical relation.

Corollary 2.4.9. *If $\Delta \mid \Gamma \vdash e \lesssim^{CIU} e' : \tau$ then $\Delta \mid \Gamma \vdash e \lesssim^{log} e' : \tau$.* $\Diamond$

Finally we have the main theorem.

Theorem 2.4.10 (CIU theorem)**.** *The relations* $\lesssim^{log}$, $\lesssim^{CIU}$ *and* $\lesssim^{ctx}$ *coincide.* ◊

Proof. It is standard (e.g. [78]) that $\lesssim^{ctx}$ is included in $\lesssim^{CIU}$. To see that $\lesssim^{log}$ is included in $\lesssim^{ctx}$ we have by Lemma 2.4.7 that the transitive closure of $\lesssim^{log}$ is an adequate precongruence, thus included in $\lesssim^{ctx}$. And $\lesssim^{log}$ is included in the transitive closure of $\lesssim^{log}$. Corollary 2.4.8 completes the cycle of inclusions. $\mathfrak{QED}$

Using the logical relation and Theorem 2.4.10 we can prove some extensionality properties. The proofs are standard and can be found in the Section 2.A in the appendix.

Lemma 2.4.11 (Functional extensionality for values)**.** *Suppose* $\tau, \sigma \in \mathfrak{T}(\Delta)$ *and let* f *and* f' *be two* values *of type* $\tau \to \sigma$ *in context* $\Delta \mid \Gamma$. *If for all* $u \in \mathbf{Val}(\tau)$ *we have*

$$\Delta \mid \Gamma \vdash f\,u \lesssim^{ctx} f'\,u : \sigma$$

then

$$\Delta \mid \Gamma \vdash f \lesssim^{ctx} f' : \tau \to \sigma.$$

◊

The extensionality for *expressions*, as opposed to only *values*, of function type does not hold in general due to the presence of choice reductions. See Remark 2.5.2 for an example. We also have extensionality for *values* of universal types.

Lemma 2.4.12 (Extensionality for the universal type)**.** *Let* $\tau \in \mathfrak{T}(\Delta, \alpha)$ *be a type. Let* f, f' *be two* values *of type* $\forall \alpha.\tau$ *in context* $\Delta \mid \Gamma$. *If for all closed types* σ *we have*

$$\Delta \mid \Gamma \vdash f[\,] \lesssim^{ctx} f'[\,] : \tau[\sigma/\alpha]$$

then

$$\Delta \mid \Gamma \vdash f \lesssim^{ctx} f' : \forall \alpha.\tau.$$

◊

2.5 Examples

We now use our logical relation to prove some example equivalences. We show two examples involving polymorphism. More examples can be found in Section 2.D in the appendix. In particular we show the correctness of von Neumann's procedure for generating a fair sequence of coin tosses from an unfair coin. That example in particular shows how the use of biorthogonality allows us to "externalise" the reasoning to arithmetic manipulations.

We first define `fix` of type

$$\forall \alpha, \beta. ((\alpha \to \beta) \to (\alpha \to \beta)) \to (\alpha \to \beta)$$

be the term

$$\Lambda.\Lambda.\lambda f.\lambda z.\delta_f (\texttt{fold}\,\delta_f)\, z$$

where δ_f is the term

$$\lambda y.\texttt{let}\ y' = \texttt{unfold}\, y\ \texttt{in}\ f\,(\lambda x.y'\,y\,x).$$

This is a call-by-value fixed-point combinator. We also write $e_1 \oplus e_2$ for the term $\texttt{if}_1\ \texttt{rand}\,\underline{2}\ \texttt{then}\ e_1\ \texttt{else}\ e_2$. Note that the choice is made before evaluating the terms e_1 and e_2.

We characterise inhabitants of a polymorphic type and show a free theorem. For the former, we need to know which real numbers can be probabilities of termination of programs. Recall that a real number r is *left-computable* if there exists a *computable* increasing (not necessarily strictly) sequence $\{q_n\}_{n\in\omega}$ of *rational numbers* such that $r = \sup_{n\in\omega} q_n$. To not detract from the main development we prove the following proposition in Section 2.B in the appendix.

Proposition 2.5.1. *For any expression e, $\mathfrak{P}^{\Downarrow}(e)$ is a left-computable real number and for any left-computable real number r in the interval $[0,1]$ there is a closed term e_r of type $\mathbf{1} \to \mathbf{1}$ such that $\mathfrak{P}^{\Downarrow}(e_r\,\langle\rangle) = r$.* ◊

Inhabitants of the type $\forall \alpha.\alpha \to \alpha$

In this section we use further syntactic sugar for sequencing. When $e, e' \in \mathbf{Tm}$ are closed terms we write $e;e'$ for $(\lambda\_.e')e$, i.e. first run e, ignore the result and then run e'. We will need the property that for all terms $e, e' \in \mathbf{Tm}$, $\mathfrak{P}^{\Downarrow}(e;e') = \mathfrak{P}^{\Downarrow}(e) \cdot \mathfrak{P}^{\Downarrow}(e')$. This is proved by Scott induction as Lemma 2.A.4 in the appendix.

Using Proposition 2.5.1 we have for each left-computable real r in the interval $[0,1]$ an inhabitant t_r of the type $\forall\alpha.\alpha \to \alpha$ given by $\Lambda.\lambda x.e_r\,\langle\rangle;x$.

We now show that these are the only inhabitants of $\forall\alpha.\alpha \to \alpha$ of the form $\Lambda.\lambda x.e$. Given such an inhabitant let $r = \mathfrak{P}^{\Downarrow}(e[\langle\rangle/x])$. We know from Proposition 2.5.1 that r is left-computable.

Given a value v of type τ and $n \in \mathbb{N}$ we define relations $R(n) = \{(\langle\rangle, v)\}$ and $S(n) = \{(v, \langle\rangle)\}$. Note that the relations are independent of n, i.e. R and S are constant relations. By reflexivity of the logical relation and the relational actions of types we have

$$\forall n, (e[\langle\rangle/x], e[v/x]) \in R^{\top\top}(n) \qquad \text{and} \qquad \forall n, (e[v/x], e[\langle\rangle/x]) \in S^{\top\top}(n) \qquad (2.4)$$

from which we conclude that $\mathfrak{P}^{\Downarrow}(e[\langle\rangle/x]) = \mathfrak{P}^{\Downarrow}(e[v/x])$. We now show that v and $e[v/x]$ are CIU-equivalent. Let $E \in \mathbf{Stk}(\tau)$ be an evaluation context. Let $q = \mathfrak{P}^{\Downarrow}(E[v])$. Define the evaluation context $E' = -; e_q\langle\rangle$. Then $(E, E') \in S^{\top}(n)$ for all n which then means, using (2.4) and Proposition 2.3.5, that $\mathfrak{P}^{\Downarrow}(E[e[v/x]]) \le \mathfrak{P}^{\Downarrow}(E'[e[\langle\rangle/x]])$. We then have

$$\mathfrak{P}^{\Downarrow}(E'[e[\langle\rangle/x]]) = \mathfrak{P}^{\Downarrow}(e[\langle\rangle/x]) \cdot \mathfrak{P}^{\Downarrow}\left(e_q\langle\rangle\right) = r \cdot \mathfrak{P}^{\Downarrow}(E[v])$$

and so $\mathfrak{P}^{\Downarrow}(E[e[v/x]]) \le r \cdot \mathfrak{P}^{\Downarrow}(E[v])$.

Similarly we have $(E', E) \in R^{\top}(n)$ for all n which implies

$$\mathfrak{P}^{\Downarrow}(E[e[v/x]]) \ge \mathfrak{P}^{\Downarrow}(E'[e[\langle\rangle/x]]).$$

We also have $\mathfrak{P}^{\Downarrow}(E'[e[\langle\rangle/x]]) = r \cdot \mathfrak{P}^{\Downarrow}(E[v])$.

So we have proved $\mathfrak{P}^{\Downarrow}(E[e[v/x]]) = r \cdot \mathfrak{P}^{\Downarrow}(E[v]) = \mathfrak{P}^{\Downarrow}(e[v/x]) \cdot \mathfrak{P}^{\Downarrow}(E[v])$. It is easy to show by Scott induction, that $\mathfrak{P}^{\Downarrow}(E[t_r[]v]) = \mathfrak{P}^{\Downarrow}(e_r\langle\rangle) \cdot \mathfrak{P}^{\Downarrow}(E[v])$. We have thus shown that for any value v, the terms $e[v/x]$ and $\mathfrak{P}^{\Downarrow}(t_r[]v)$ are CIU-equivalent. Using Theorem 2.4.10 and Lemmas 2.4.12 and 2.4.11 we conclude that the terms $\forall\alpha.\lambda x.e$ and t_r are contextually equivalent.

Remark 2.5.2. Unfortunately we cannot so easily characterise general values of the type $\forall\alpha.\alpha \to \alpha$, that is, those not of the form $\Lambda.v$ for a value v. Consider the term $\Lambda.t_{\frac{1}{2}} \oplus t_{\frac{1}{3}}$. It is a straightforward calculation that for any evaluation context E and value v,

$$\mathfrak{P}^{\Downarrow}\left(E\left[\left(t_{\frac{1}{2}} \oplus t_{\frac{1}{3}}\right) v\right]\right) = \frac{5}{12}\mathfrak{P}^{\Downarrow}(E[v]) = \mathfrak{P}^{\Downarrow}\left(E\left[t_{\frac{5}{12}}\, v\right]\right)$$

thus if $\Lambda.t_{\frac{1}{2}} \oplus t_{\frac{1}{3}}$ is equivalent to any $\Lambda.t_r$ it must be $\Lambda.t_{\frac{5}{12}}$.

Let E be the evaluation context

$$E = \mathtt{let}\ f\ = -[]\ \mathtt{in}\ \mathtt{let}\ x\ = f\langle\rangle\ \mathtt{in}\ f\langle\rangle.$$

We compute $\mathfrak{P}^{\Downarrow}\left(E\left[\Lambda.t_{\frac{1}{2}} \oplus t_{\frac{1}{3}}\right]\right) = \frac{13}{72}$ and $\mathfrak{P}^{\Downarrow}\left(E\left[\Lambda.t_{\frac{5}{12}}\right]\right) = \frac{25}{144}$ showing that $\Lambda.t_{\frac{1}{2}} \oplus t_{\frac{1}{3}}$ is *not* equivalent to $\Lambda.t_{\frac{5}{12}}$.

This example also shows that extensionality for *expressions*, as opposed to *values*, of function type does not hold. The reason is, of course, that probabilistic choice is a computational effect and so it matters how many times we evaluate the term and this is what the constructed evaluation context uses to distinguish the terms. ♦

A free theorem for lists

Let τ be a type and α not free in τ. We write $[\tau]$ for the type of lists $\mu\alpha.(\mathbf{1} + \tau \times \alpha)$, nil for the empty list and

$$\mathtt{cons} : \forall\alpha.\alpha \to [\alpha] \to [\alpha]$$
$$\mathtt{cons} = \Lambda.\lambda x.\lambda xs.\mathtt{fold}(\mathtt{inr}\langle x, xs\rangle).$$

for the other constructor. The function $\mathtt{map}$ of type

$$\forall\alpha.\forall\beta.(\alpha \to \beta) \to [\alpha] \to [\beta]$$

is the function applying the given function to all elements of the list in order. Additionally, we define composition of terms $f \circ g$ as the term $\lambda x.f(g(x))$ (for x not free in f and g).

We will now show that if a term m of type $\forall\alpha.\forall\beta.(\alpha \to \beta) \to [\alpha] \to [\beta]$ is equivalent to a term of the form $\Lambda.\Lambda.\lambda x.e$ then it satisfies

$$m[][](f \circ g) =^{ctx} m[][]f \circ \mathtt{map}[][]g$$

for all *values* f and all *deterministic and terminating* g. By this we mean that for each value v in the domain of g, there exists a *value* u in the codomain of g, such that $g\,v =^{ctx} u$. For instance, if g reduces without using choice reductions and is terminating, then g is deterministic. There are other functions that are also deterministic and terminating, though, for instance $\lambda x.\langle\rangle \oplus \langle\rangle$. In the appendix we show that these restrictions are not superfluous.

So let m be a closed term of type $\forall\alpha.\forall\beta.(\alpha \to \beta) \to [\alpha] \to [\beta]$ and suppose further that m is equivalent to a term of the form $\Lambda.\Lambda.\lambda x.e$. Let $\tau, \sigma, \rho \in \mathfrak{T}$ be closed types and $f \in \mathbf{Val}(\sigma \to \rho)$ and $g \in \mathbf{Tm}(\tau \to \sigma)$ be a deterministic and terminating function. Then

$$\varnothing \mid \varnothing \vdash m[][](f \circ g) =^{ctx} m[][]f \circ \mathtt{map}[][]g : [\tau] \to [\rho].$$

We prove two approximations separately, starting with $\lesssim^{ctx}$. We use Theorem 2.4.10 multiple times. We have

$$\alpha, \beta \mid \varnothing \vdash m[][] : (\alpha \to \beta) \to [\alpha] \to [\beta].$$

Let $R = \lambda n.\{(v,u) \mid g\,v =^{ctx} u\}$ be an element of $\mathbf{VRel}(\tau,\sigma)$ and $S \in \mathbf{VRel}(\rho,\rho)$ the constant identity relation on $\mathbf{Val}(\rho)$. Let φ map α to R and β to S. Proposition 2.4.5 gives

$$(m[][], m[][]) \in [\![(\alpha \to \beta) \to [\alpha] \to [\beta]]\!](\varphi)^{\top\top}(n)$$

for all $n \in \mathbb{N}$.

We first claim that $(f \circ g, f) \in [\![\alpha \to \beta]\!](\varphi)(n)$ for all $n \in \mathbb{N}$. Since f is a value and has a type, it must be of the form $\lambda x.e$ for some x and e. Take $j \in \mathbb{N}$, related values $(v,u) \in r(j)$, $k \leq j$ and related evaluation contexts $(E,E') \in S^{\top}(k)$. By Theorem 2.4.10 and the definition of relation R we have

$$\mathfrak{P}^{\Downarrow}(E'[f\,u]) = \mathfrak{P}^{\Downarrow}(E'[f(g\,v)]).$$

Using the results about $\mathfrak{P}^{\Downarrow}_k(\cdot)$ and $\mathfrak{P}^{\Downarrow}(\cdot)$ proved in Section 2.C in the appendix we get

$$\mathfrak{P}^{\Downarrow}_k(E[f(g(v))]) \leq \sum_{\pi : f(g(v)) \rightsquigarrow^* w} \mathfrak{W}(\pi)\,\mathfrak{P}^{\Downarrow}_k(E[w]) \leq \sum_{\pi : f(g(v)) \rightsquigarrow^* w} \mathfrak{W}(\pi)\,\mathfrak{P}^{\Downarrow}(E'[w])$$

and the last term is equal to $\mathfrak{P}^{\Downarrow}(E'[f\,(g\,v)])$ which is equal to $\mathfrak{P}^{\Downarrow}(E'[f\,u])$.

From this we can conclude

$$(m[\,][\,](f \circ g), m[\,][\,]f) \in [\![[\alpha] \to [\beta]]\!](\varphi)^{\top\top}(n)$$

for all $n \in \mathbb{N}$. Note that we have *not yet* used the fact that g is deterministic and terminating. We do so now.

Let xs be a list of elements of type τ. Then induction on the length of xs, using the assumption on g, we can derive that there exists a list ys of elements of type σ, such that $\mathtt{map}[\,][\,]\,g\,xs =^{ctx} ys$ and $(xs, ys) \in [\![[\alpha]]\!](\varphi)(n)$ for all n. This gives us

$$(m[\,][\,](f \circ g)\,xs, m[\,][\,]f\,ys) \in [\![[\beta]]\!](\varphi)^{\top\top}(n)$$

for all $n \in \mathbb{N}$. Since the relation S is the identity relation we have for all evaluation contexts E of a suitable type, $(E, E) \in S^{\top}(n)$ for all n, which gives

$$\begin{aligned} m[\,][\,](f \circ g)\,xs \lesssim^{CIU} m[\,][\,]f\,ys &=^{ctx} m[\,][\,]f\,(\mathtt{map}[\,][\,]\,g\,xs) \\ &=^{ctx} (m[\,][\,]f \circ \mathtt{map}[\,][\,]\,g)\,xs \end{aligned}$$

where the last equality holds because β-reduction is an equivalence.

We now conclude by using the fact that m is (equivalent to) a term of the form $\Lambda.\Lambda.\lambda x.e$ and use Lemma 2.4.11 to conclude

$$m[\,][\,](f \circ g) \lesssim^{ctx} m[\,][\,]f \circ \mathtt{map}[\,][\,]\,g.$$

For the other direction, we proceed analogously. The relation for β remains the identity relation, and the relation for R for α is $\{(v, u) \mid v =^{ctx} g\,u\}$.

2.6 Extension with References

We now sketch the extension of $\mathbf{F}^{\mu,\oplus}$ to include dynamically allocated references. For simplicity we add ground store only, so we do not have to solve a domain equation giving us the space of semantic types and worlds [4]. We show an equivalence using state and probabilistic choice which shows that the addition of references to the language is orthogonal to the addition of probabilistic choice. We conjecture that the extension with *higher-order* dynamically allocated references can be done as in earlier work on step-indexed logical relations [40].

We extend the language by adding the type `refnat` and extend the grammar of terms e with

$$e ::= \ldots \mid \ell \mid \mathtt{ref}\,e \mid e_1 := e_2 \mid !e$$

with ℓ being locations.

To model allocation we need to additionally index the interpretation of types by worlds. To keep things simple a world $w \in \mathcal{W}$ is partial bijection f on locations together with, for each pair of locations $(\ell_1, \ell_2) \in f$, a relation R on numerals. We write $(\ell_1, \ell_2, R) \in w$ when the partial bijection in w relates ℓ_1 and ℓ_2 and R is the relation assigned to the pair (ℓ_1, ℓ_2). Technically, worlds can be encoded as subsets of

$$\mathtt{Loc}^2 \times \mathcal{P}(\{\underline{n} \mid n \in \mathbb{N}\} \times \{\underline{n} \mid n \in \mathbb{N}\})$$

satisfying the conditions described above.

The operational semantics has to be extended to include heaps, which are modelled as finite maps from locations to numerals. A pair of heaps (h_1, h_2) satisfies the world w, written $(h_1, h_2) \in \lfloor w \rfloor$, when

$$\forall (\ell_1, \ell_2, R) \in w, (h_1(\ell_1), h_2(\ell_2)) \in R.$$

The interpretation of types is then extended to include worlds. The denotation of a type is now an element of $\mathcal{W} \overset{mon}{\to} \mathbf{VRel}(\cdot, \cdot)$ where the order on $\mathcal{W}$ is inclusion. Let

$$\mathbf{WRel}(\tau, \tau') = \mathcal{W} \overset{mon}{\to} \mathbf{VRel}(\tau, \tau').$$

We define

$$[\![\Delta \vdash \mathtt{ref\,nat}]\!](\varphi)(w)(n) = \{(\ell_1, \ell_2) \mid (\ell_1, \ell_2, =) \in w\}$$

where $=$ is the equality relation on numerals. Notice that the step-index is not used. This is because we are only considering first-order state.

The rest of the interpretation stays the same, apart from some quantification over "future worlds" in the function case to maintain monotonicity. We also need to change the definition of the $\top\top$-closure to use the world satisfaction relation. For $R \in \mathbf{WRel}(\tau, \tau')$ we define an indexed relation (indexed by worlds) $R^\top$ as

$$R^\top(w)(n) = \left\{ (E, E') \;\middle|\; \begin{array}{c} \forall w' \geq w, \forall k \leq n, \forall (h_1, h_2) \in \lfloor w' \rfloor, \forall (v_1, v_2) \in R(w')(k), \\ \mathfrak{P}^{\Downarrow}_k(\langle h_1, E[v_1] \rangle) \leq \mathfrak{P}^{\Downarrow}(\langle h_2, E[v_2] \rangle) \end{array} \right\}$$

and analogously for $\cdot^\perp$.

We now sketch a proof that two modules, each implementing a counter by using a single internal location, are contextually equivalent. The increment method is special. When called, it chooses, uniformly, whether to increment the counter or not. The two modules differ in the way they increment the counter. One module increments the counter by 1, the other by 2. Concretely, we show that the two counters

$$\mathtt{pack}\,(\lambda - .\mathtt{ref}\,\underline{1}, \lambda x.!x, \lambda x.\langle\rangle \oplus (x := \mathtt{S}\,!x))$$

and

$$\mathtt{pack}\,(\lambda - .\mathtt{ref}\,\underline{2}, \lambda x.!x\;\mathtt{div}\;\underline{2}, \lambda x.\langle\rangle \oplus (x := \mathsf{S}\,(\mathsf{S}\,!x)))$$

are contextually equivalent at type

$$\exists\alpha.(\mathbf{1} \to \alpha) \times (\alpha \to \mathtt{nat}) \times (\alpha \to \mathbf{1}).$$

We have used div for the division function on numerals which can easily be implemented.

The interpretation of existentials $[\![\Delta \vdash \exists\alpha.\tau]\!]\,(\varphi)(w)(n)$ is

$$\left\{ (\mathtt{pack}\,v, \mathtt{pack}\,v') \;\middle|\; \begin{array}{l} \exists\sigma,\sigma' \in \Upsilon, \exists R \in \mathbf{WRel}\,(\sigma,\sigma'), \\ (v,v') \in [\![\Delta,\alpha \vdash \tau]\!]\,(\varphi\,[\alpha \mapsto R])\,(w)(n) \end{array} \right\}.$$

To prove the counters are contextually equivalent we show them directly related in the value relation. We choose the types σ and σ' to be $\mathtt{ref\;nat}$ and the relation R to be

$$R(w)(n) = \{(\ell_1,\ell_2) \mid (\ell_1,\ell_2,G(2\times -)) \in w\}$$

where $G(2\times -)$ is the relation

$$\{(\underline{k}, \underline{2\cdot k}) \;\big|\; k \in \mathbb{N}\}.$$

Notice again that the relation is independent of the step-index n. We now need to check all three functions to be related at the value relation.

First, the allocation functions. We only show one approximation, the other is completely analogous. Concretely, we show that for any $n \in \mathbb{N}$ and any world $w \in \mathcal{W}$ we have

$$(\lambda - .\mathtt{ref}\,\underline{1}, \lambda - .\mathtt{ref}\,\underline{2}) \in [\![\mathbf{1} \to \alpha]\!]\,(R)(w)(n).$$

Let $n \in \mathbb{N}$ and $w \in \mathcal{W}$. Take $w' \geq w$ and related arguments v, v' at type $\mathbf{1}$. We know by construction that $v = v' = \langle\rangle$ so we have to show that

$$(\mathtt{ref}\,\underline{1}, \mathtt{ref}\,\underline{2}) \in [\![\alpha]\!]\,(R)^{\top\top}(w')(n).$$

Let $w'' \geq w'$ and $j \leq n$ and take two related evaluation contexts (E,E') at $[\![\alpha]\!]\,(R)^{\top}(w'')(j)$ and $(h,h') \in \lfloor w'' \rfloor$. Let $\ell \notin \mathrm{dom}\,(h)$ and $\ell' \notin \mathrm{dom}\,(h')$. We have

$$\mathfrak{P}^{\Downarrow}_{j}\,(\langle h, E[\mathtt{ref}\,\underline{1}]\rangle) = \mathfrak{P}^{\Downarrow}_{j}\,(\langle h\,[\ell \mapsto \underline{1}], E[\ell]\rangle)$$

and

$$\mathfrak{P}^{\Downarrow}\,(\langle h', E'[\mathtt{ref}\,\underline{2}]\rangle) = \mathfrak{P}^{\Downarrow}\,(\langle h'\,[\ell' \mapsto \underline{2}], E'[\ell']\rangle).$$

Let w''' be w'' extended with $(\ell, \ell', G(2\times -))$. Then the extended heaps are in $\lfloor w''' \rfloor$ and $w''' \geq w''$. Thus E and E' are also related at w''' by monotonicity.

Similarly we can prove that $(\ell,\ell') \in [\![\alpha]\!](R)(w''')(j)$. This then allows us to conclude

$$\mathfrak{P}^{\Downarrow}_{j}(\langle h[\ell \mapsto \underline{1}], E[\ell]\rangle) \le \mathfrak{P}^{\Downarrow}(\langle h'[\ell' \mapsto \underline{2}], E'[\ell']\rangle)$$

which concludes the proof.

Lookup is simple so we omit it. Update is more interesting. Let $n \in \mathbb{N}$ and $w \in \mathcal{W}$. Let ℓ and ℓ' be related at $[\![\alpha]\!](R)(w)(n)$. We need to show

$$(\langle\rangle \oplus (\ell := \mathsf{S}\,!\ell), \langle\rangle \oplus (\ell' := \mathsf{S}\,(\mathsf{S}\,!\ell'))) \in [\![\mathbf{1}]\!](R)^{\top\top}(w)(n).$$

Take $w' \ge w$, $j \le n$ and $(h,h') \in \lfloor w' \rfloor$. Take related evaluation contexts E and E' at w' and j. We have

$$\begin{aligned}
\mathfrak{P}^{\Downarrow}_{j}(\langle h, E[\langle\rangle \oplus (\ell := \mathsf{S}\,!\ell)]\rangle) &= \tfrac{1}{2}\mathfrak{P}^{\Downarrow}_{j}(\langle h, E[\langle\rangle]\rangle) + \tfrac{1}{2}\mathfrak{P}^{\Downarrow}_{j}(\langle h, E[\ell := \mathsf{S}\,!\ell]\rangle)\\
\mathfrak{P}^{\Downarrow}(\langle h', E'[\langle\rangle \oplus (\ell' := \mathsf{S}\,\mathsf{S}\,!\ell')]\rangle) &= \tfrac{1}{2}\mathfrak{P}^{\Downarrow}(\langle h', E'[\langle\rangle]\rangle) + \tfrac{1}{2}\mathfrak{P}^{\Downarrow}(\langle h', E'[\ell' := \mathsf{S}\,\mathsf{S}\,!\ell']\rangle)
\end{aligned}$$

Since ℓ and ℓ' are related at $[\![\alpha]\!](R)(w)(n)$ and $w' \ge w$ and $(h,h') \in \lfloor w' \rfloor$ we know that $h(\ell) = \underline{m}$ and $h'(\ell') = \underline{2 \cdot m}$ for some $m \in \mathbb{N}$. Thus

$$\mathfrak{P}^{\Downarrow}_{j}(\langle h, E[\ell := \mathsf{S}\,!\ell]\rangle) = \mathfrak{P}^{\Downarrow}_{j}(\langle h_1, E[\langle\rangle]\rangle)$$

where $h_1 = h[\ell \mapsto \underline{m+1}]$. And also

$$\mathfrak{P}^{\Downarrow}(\langle h', E'[\ell' := \mathsf{S}\,\mathsf{S}\,!\ell']\rangle) = \mathfrak{P}^{\Downarrow}(\langle h_2, E'[\langle\rangle]\rangle)$$

where $h_2 = h'\left[\ell' \mapsto \underline{2 \cdot (m+1)}\right]$. The fact that h_1 and h_2 are still related concludes the proof.

The above proof shows that reasoning about examples involving state and choice is possible.

2.7 Conclusion

We have constructed a step-indexed logical relation for a higher-order language with probabilistic choice. In contrast to earlier work, our language also features impredicative polymorphism and recursive types. We also show how to extend our logical relation to a language with dynamically allocated local state. In future work, we will explore whether the step-indexed technique can be used for developing models of program logics for probabilistic computation that support reasoning about more properties than just contextual equivalence. We are also interested in including primitives for continuous probability distributions.

Acknowledgements

We thank Filip Sieczkowski, Kasper Svendsen and Thomas Dinsdale-Young for discussions of various aspects of this work and the reviewers for their comments.

This research was supported in part by the ModuRes Sapere Aude Advanced Grant from The Danish Council for Independent Research for the Natural Sciences (FNU) and in part by Microsoft Research through its PhD Scholarship Programme.

2.A Language Definitions and Properties

$$
\begin{aligned}
\tau &::= \alpha \mid \mathbf{1} \mid \mathtt{nat} \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \to \tau_2 \mid \mu\alpha.\tau \mid \forall\alpha.\tau \mid \exists\alpha.\tau \\
v &::= x \mid \langle\rangle \mid \underline{n} \mid \langle v_1, v_2\rangle \mid \lambda x.e \mid \mathtt{inl}\ v \mid \mathtt{inr}\ v \mid \Lambda.e \mid \mathtt{pack}\, v \\
e &::= x \mid \langle\rangle \mid \underline{n} \mid \langle e_1, e_2\rangle \mid \lambda x.e \mid \mathtt{inl}\ e \mid \mathtt{inr}\ e \mid \Lambda.e \mid \mathtt{pack}\, e \\
&\qquad \mid \mathtt{proj}_i\, e \mid e_1\, e_2 \mid \mathtt{match}(e, x_1.e_1, x_2.e_2) \mid e[] \\
&\qquad \mid \mathtt{unpack}\ e_1\ \mathtt{as}\ x\ \mathtt{in}\ e_2 \mid \mathtt{unfold}\, e \mid \mathtt{fold}\, e \mid \mathtt{rand}\, e \\
&\qquad \mid \mathtt{if}_1\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \mid \mathtt{P}\, e \mid \mathtt{S}\, e \\
E &::= - \mid \langle E, e\rangle \mid \langle v, E\rangle \mid \mathtt{inl}\ E \mid \mathtt{inr}\ E \mid \mathtt{pack}\, E \\
&\qquad \mid \mathtt{proj}_i\, E \mid E\, e \mid v\, E \mid \mathtt{match}(E, x_1.e_1, x_2.e_2) \mid E[] \\
&\qquad \mid \mathtt{unpack}\ E\ \mathtt{as}\ x\ \mathtt{in}\ e \mid \mathtt{unfold}\, E \mid \mathtt{fold}\, E \\
&\qquad \mid \mathtt{if}_1\ E\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \mid \mathtt{rand}\, E \mid \mathtt{P}\, E \mid \mathtt{S}\, E
\end{aligned}
$$

Figure 2.2: Types, terms and evaluation contexts. $\underline{n}$ are numerals of type $\mathtt{nat}$.

$$
\frac{\alpha \in \Delta}{\Delta \vdash \alpha} \qquad \Delta \vdash \mathbf{1} \qquad \Delta \vdash \mathtt{nat} \qquad \frac{\Delta \vdash \tau_1 \qquad \Delta \vdash \tau_2}{\Delta \vdash \tau_1 \times \tau_2} \qquad \frac{\Delta \vdash \tau_1 \qquad \Delta \vdash \tau_2}{\Delta \vdash \tau_1 + \tau_2}
$$

$$
\frac{\Delta \vdash \tau_1 \qquad \Delta \vdash \tau_2}{\Delta \vdash \tau_1 \to \tau_2} \qquad \frac{\Delta, \alpha \vdash \tau}{\Delta \vdash \exists\alpha.\tau} \qquad \frac{\Delta, \alpha \vdash \tau}{\Delta \vdash \forall\alpha.\tau} \qquad \frac{\Delta, \alpha \vdash \tau}{\Delta \vdash \mu\alpha.\tau}
$$

Figure 2.3: Well-formed types. The judgement $\Delta \vdash \tau$ expresses $ftv(\tau) \subseteq \Delta$.

The following lemma uses definitions from Section 2.3.

Lemma 2.A.1. *Φ is monotone and preserves suprema of ω-chains.* ◊

Proof. Since the order in $\mathcal{F}$ is pointwise and multiplication and addition are monotone it is easy to see that Φ is monotone.

To show that it is continuous let $\{f_n\}_{n\in\omega}$ be an ω-chain in $\mathcal{F}$. If e is a value the result is immediate. Otherwise we have

$$
\Phi\left(\sup_{n\in\omega} f_n\right)(e) = \sum_{e \overset{p}{\rightsquigarrow} e'} p \cdot \left(\sup_{n\in\omega} f_n\right)(e')
$$

and since suprema in $\mathcal{F}$ are computed pointwise we have

$$
= \sum_{e \overset{p}{\rightsquigarrow} e'} p \cdot \sup_{n\in\omega}\left(f_n(e')\right)
$$

$$\frac{x{:}\tau \in \Gamma \qquad \Delta \vdash \Gamma}{\Delta \mid \Gamma \vdash x : \tau} \qquad \frac{\Delta \vdash \Gamma}{\Delta \mid \Gamma \vdash \langle\rangle : \mathbf{1}} \qquad \frac{\Delta \mid \Gamma \vdash e_1 : \tau_1 \qquad \Delta \mid \Gamma \vdash e_2 : \tau_2}{\Delta \mid \Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2}$$

$$\frac{\Delta \mid \Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Delta \mid \Gamma \vdash \lambda x.e : \tau_1 \to \tau_2} \qquad \frac{\Delta \mid \Gamma \vdash e : \tau_1 \qquad \Delta \vdash \tau_2}{\Delta \mid \Gamma \vdash \mathtt{inl}\ e : \tau_1 + \tau_2} \qquad \frac{\Delta \mid \Gamma \vdash e : \tau_2 \qquad \Delta \vdash \tau_1}{\Delta \mid \Gamma \vdash \mathtt{inr}\ e : \tau_1 + \tau_2}$$

$$\frac{\Delta \mid \Gamma, x_1{:}\tau_1 \vdash e_1 : \tau \qquad \Delta \mid \Gamma, x_2{:}\tau_2 \vdash e_2 : \tau \qquad \Delta \mid \Gamma \vdash e : \tau_1 + \tau_2}{\Delta \mid \Gamma \vdash \mathtt{match}(e, x_1.e_1, x_2.e_2) : \tau}$$

$$\frac{\Delta, \alpha \mid \Gamma \vdash e : \tau}{\Delta \mid \Gamma \vdash \Lambda.e : \forall \alpha.\tau} \qquad \frac{\Delta \mid \Gamma \vdash e : \tau_1 \times \tau_2}{\Delta \mid \Gamma \vdash \mathtt{proj}_i\ e : \tau_i} \qquad \frac{\Delta \mid \Gamma \vdash e : \tau' \to \tau \qquad \Delta \mid \Gamma \vdash e' : \tau'}{\Delta \mid \Gamma \vdash e\, e' : \tau}$$

$$\frac{\Delta \vdash \tau_1 \qquad \Delta \mid \Gamma \vdash e : \tau[\tau_1/\alpha]}{\Delta \mid \Gamma \vdash \mathtt{pack}\, e : \exists \alpha.\tau}$$

$$\frac{\Delta \mid \Gamma \vdash e : \exists \alpha.\tau_1 \qquad \Delta \vdash \tau \qquad \Delta, \alpha \mid \Gamma, x : \tau_1 \vdash e' : \tau}{\Delta \mid \Gamma \vdash \mathtt{unpack}\ e\ \mathtt{as}\ x\ \mathtt{in}\ e' : \tau}$$

$$\frac{\Delta \mid \Gamma \vdash e : \mu\alpha.\tau}{\Delta \mid \Gamma \vdash \mathtt{unfold}\, e : \tau[\mu\alpha.\tau/\alpha]} \qquad \frac{\Delta \mid \Gamma \vdash e : \tau[\mu\alpha.\tau/\alpha]}{\Delta \mid \Gamma \vdash \mathtt{fold}\, e : \mu\alpha.\tau}$$

$$\frac{\Delta \mid \Gamma \vdash e : \forall \alpha.\tau \qquad \Delta \vdash \tau'}{\Delta \mid \Gamma \vdash e[\,] : \tau[\tau'/\alpha]} \qquad \frac{\Delta \mid \Gamma \vdash e : \mathtt{nat}}{\Delta \mid \Gamma \vdash \mathtt{rand}\ e : \mathtt{nat}}$$

$$\frac{\Delta \mid \Gamma \vdash e : \mathtt{nat} \qquad \Delta \mid \Gamma \vdash e_1 : \tau \qquad \Delta \mid \Gamma \vdash e_2 : \tau}{\Delta \mid \Gamma \vdash \mathtt{if}_1\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : \tau} \qquad \frac{\Delta \mid \Gamma \vdash e : \mathtt{nat}}{\Delta \mid \Gamma \vdash \mathtt{P}\, e : \mathtt{nat}}$$

$$\frac{\Delta \mid \Gamma \vdash e : \mathtt{nat}}{\Delta \mid \Gamma \vdash \mathtt{S}\, e : \mathtt{nat}}$$

Figure 2.4: Typing of terms, where $\Gamma ::= \emptyset \mid \Gamma, x{:}\tau$ and $\Delta ::= \emptyset \mid \Delta, \alpha$.

Using the fact that sum and product are continuous and that the sum in the

Basic reductions $\overset{\cdot}{\longmapsto}$

$$
\begin{aligned}
\mathtt{proj}_i\,\langle v_1, v_2\rangle &\xrightarrow{1} v_i \\
(\lambda x.e)\,v &\xrightarrow{1} e[v/x] \\
\mathtt{unfold}\,(\mathtt{fold}\,v) &\xrightarrow{1} v \\
\mathtt{unpack}\ (\mathtt{pack}\,v)\ \mathtt{as}\ x\ \mathtt{in}\ e &\xrightarrow{1} e[v/x] \\
(\Lambda.e)[\,] &\xrightarrow{1} e \\
\mathtt{rand}\,\underline{n} &\xrightarrow{\frac{1}{n}} \underline{k} \qquad (k \in \{1, 2, \ldots, n\}) \\
\mathtt{match}\,(\mathtt{inl}\,v, x_1.e_1, x_2.e_2) &\xrightarrow{1} e_1[v/x_1] \\
\mathtt{match}\,(\mathtt{inr}\,v, x_1.e_1, x_2.e_2) &\xrightarrow{1} e_2[v/x_2] \\
\mathtt{P}\,\underline{n} &\xrightarrow{1} \underline{\max\{n-1, 1\}} \\
\mathtt{S}\,\underline{n} &\xrightarrow{1} \underline{n+1} \\
\mathtt{if}_1\ \underline{1}\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 &\xrightarrow{1} e_1 \\
\mathtt{if}_1\ \mathtt{S}\,\underline{n}\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 &\xrightarrow{1} e_2
\end{aligned}
$$

One step reduction relation $\overset{\cdot}{\leadsto}$

$$
E[e] \overset{p}{\leadsto} E[e'] \qquad\qquad \text{if } e \xrightarrow{p} e'
$$

Figure 2.5: Operational semantics.

definition of Φ is finite we get

$$
\begin{aligned}
\Phi\left(\sup_{n\in\omega} f_n\right)(e) &= \sup_{n\in\omega}\left(\sum_{e \overset{p}{\leadsto} e'} p \cdot f_n(e')\right) \\
&= \sup_{n\in\omega} \Phi\,(f_n)\,(e) = \left(\sup_{n\in\omega} \Phi(f_n)\right)(e)
\end{aligned}
$$

𝔔𝔈𝔇

Example 2.A.2. Let us compute probabilities of termination of some example programs.

- If $v \in \mathbf{Val}$ then by definition $\mathfrak{P}^{\Downarrow}(v) = 1$.
- If $e \in \mathbf{Tm} \setminus \mathbf{Val}$ is stuck then $\mathfrak{P}^{\Downarrow}(e) = 0$ by definition.

- Suppose there exists a cycle $e \overset{1}{\rightsquigarrow} e_1 \overset{1}{\rightsquigarrow} e_2 \overset{1}{\rightsquigarrow} \cdots \overset{1}{\rightsquigarrow} e_n \overset{1}{\rightsquigarrow} e$. Then $\mathfrak{P}^{\Downarrow}(e) = \mathfrak{P}^{\Downarrow}(e_1) = \cdots = \mathfrak{P}^{\Downarrow}(e_n) = 0$.

  It follows from the assumption that none of e_k are values and since the sum of outgoing weights is at most 1 we have that for each e_k and e all other weights must be 0. We thus get that $\mathfrak{P}^{\Downarrow}(e) = \mathfrak{P}^{\Downarrow}(e_1) = \cdots = \mathfrak{P}^{\Downarrow}(e_n)$ by simply unfolding the fixed point n-times. To show that they are all 0 we use Scott induction. Define

  $$\mathcal{S} = \{f \in \mathcal{F} | f(e) = f(e_1) = f(e_2) = \ldots = f(e_n) = 0\}.$$

  Clearly $\mathcal{S}$ is an admissible subset of $\mathcal{F}$ and $\bot \in \mathcal{S}$. Using the above existence of the cycle of reductions it is easy to show that $\mathcal{S} \subseteq \Phi[\mathcal{S}]$. Hence by the principle of Scott induction we have $\mathfrak{P}^{\Downarrow}(\cdot) \in \mathcal{S}$ and thus $\mathfrak{P}^{\Downarrow}(e) = \mathfrak{P}^{\Downarrow}(e_1) = \ldots = \mathfrak{P}^{\Downarrow}(e_n) = 0$.

♦

This example also shows that we do really want the least fixed point of Φ, since this allows us to use Scott-induction and prove that diverging terms have zero probability of termination.

Remark 2.A.3. It is perhaps instructive to consider the relationship to the termination predicate when we do not have weights on reductions. In such a case we can consider two extremes, may- and must-termination predicates. These can be considered to be maps $\mathbf{Tm} \to 2$ where 2 is the boolean lattice $0 \leq 1$. Let $\mathcal{B} = \mathbf{Tm} \to 2$. Since 2 is a complete lattice so is $\mathcal{B}$. In particular it is a pointed ω-cpo. We can define may-termination as the least fixed point of $\Psi : \mathcal{B} \to \mathcal{B}$ defined as

$$\Psi(f)(e) = \begin{cases} 1 & \text{if } e \in \mathbf{Val} \\ \max\limits_{e \rightsquigarrow e'} f(e') & \text{otherwise} \end{cases}.$$

Observe again that if e is stuck then $\Psi(f)(e) = 0$ since the maximum of an empty set is the least element by definition.

Must-termination is slightly different. We need a special case for stuck terms.

$$\Psi'(f)(e) = \begin{cases} 1 & \text{if } e \in \mathbf{Val} \\ \min\limits_{e \rightsquigarrow e'} f(e') & \exists e' \in \mathbf{Tm}, p \in \mathcal{I}, e \overset{p}{\rightsquigarrow} e' \\ 0 & \text{otherwise} \end{cases}$$

Let $\downarrow$ be the least fixed point of Ψ and $\Downarrow$ the least fixed point of Ψ'. An additional property that holds for $\downarrow$ and $\Downarrow$, because of the fact that 2 is discrete, is that for a given e, if $e\!\downarrow = 1$ then there is a natural number n, such that

$\Psi^n(\bot)(e) = 1$, i.e. if it terminates we can observe this in finite time. This is because if an increasing sequence in 2 has supremum 1, then the sequence must be constant 1 from some point onward.

In contrast, if $\mathfrak{P}^{\Downarrow}(e) = 1$ it is not necessarily the case that there is a natural number n with $\Phi^n(\bot)(e) = 1$ because it might be the case that 1 is only reached in the limit. ♦

The next lemma uses the abbreviation ; defined in Section 2.5.

Lemma 2.A.4. *For all terms $e, e' \in \mathbf{Tm}$, $\mathfrak{P}^{\Downarrow}(e;e') = \mathfrak{P}^{\Downarrow}(e) \cdot \mathfrak{P}^{\Downarrow}(e')$.* ◊

Proof. We prove two approximations separately, both of them by Scott induction.

$\leq$ Consider the set

$$\mathcal{S} = \left\{ f \in \mathcal{F} \;\middle|\; \begin{array}{l} f \leq \mathfrak{P}^{\Downarrow}(\cdot) \wedge \forall e, e' \in \mathbf{Tm}, \\ f(e;e') \leq \mathfrak{P}^{\Downarrow}(e) \cdot \mathfrak{P}^{\Downarrow}(e') \end{array} \right\}.$$

It is easy to see that $\mathcal{S}$ contains $\bot$ and is closed under ω-chains, so we only need to show that it is preserved by Φ. The first condition is trivial to check since $\mathfrak{P}^{\Downarrow}(\cdot)$ is a fixed point of Φ. Let $f \in \mathcal{F}$ and $e, e' \in \mathbf{Tm}$. If $e \in \mathbf{Val}$ then $\Phi(f)(e;e') = f(e')$ on account of one β-reduction. By assumption $f(e') \leq \mathfrak{P}^{\Downarrow}(e')$ and by definition we have $\mathfrak{P}^{\Downarrow}(e) = 1$.

If e is not a value we have $\Phi(f)(e;e') = \sum_{e \overset{p}{\rightsquigarrow} e''} p \cdot f(e'';e') \leq \sum_{e \overset{p}{\rightsquigarrow} e''} p \cdot \mathfrak{P}^{\Downarrow}(e'') \cdot \mathfrak{P}^{\Downarrow}(e') = \mathfrak{P}^{\Downarrow}(e) \cdot \mathfrak{P}^{\Downarrow}(e')$.

Thus we can conclude by Scott induction that $\mathfrak{P}^{\Downarrow}(\cdot) \in \mathcal{S}$.

$\geq$ For this direction we consider the set

$$\mathcal{S} = \left\{ f \in \mathcal{F} \;\middle|\; \begin{array}{l} \forall E \in \mathbf{Stk}, e \in \mathbf{Tm}, v \in \mathbf{Val}, \\ \mathfrak{P}^{\Downarrow}(E[e]) \geq f(e) \cdot \mathfrak{P}^{\Downarrow}(E[v]) \end{array} \right\}.$$

It is easy to see that it is admissible and closed under Φ. Hence $\mathfrak{P}^{\Downarrow}(\cdot) \in \mathcal{S}$. Thus we have, taking $E = -;e'$ and any value v, that $\mathfrak{P}^{\Downarrow}(e) \cdot \mathfrak{P}^{\Downarrow}(v;e') \leq \mathfrak{P}^{\Downarrow}(e;e')$ and it is easy to see that $\mathfrak{P}^{\Downarrow}(v;e') = \mathfrak{P}^{\Downarrow}(e')$.

$\mathfrak{QED}$

Interpretation of types and the logical relation

Lemma 2.A.5. *The interpretation of types in Figure 2.1 is well defined. In particular the interpretation of types is non-expansive.* ◊

The substitution lemma is crucial for proving compatibility of existential and universal types. The proof is by induction on the derivation $\Delta \vdash \tau$.

Lemma 2.A.6 (Substitution)**.** *For any well-formed types* $\Delta, \alpha \vdash \tau$ *and* $\Delta \vdash \sigma$ *and any* φ *we have* $[\![\Delta \vdash \tau[\sigma/\alpha]]\!](\varphi) = [\![\Delta, \alpha \vdash \tau]\!](\varphi[\alpha \mapsto [\![\Delta \vdash \sigma]\!](\varphi)])$. ◊

We state and prove additional context extension lemmas. The other cases are similar.

Lemma 2.A.7. *Let* $n \in \mathbb{N}$. *If*

$$(v, v') \in [\![\Delta \vdash \tau_1 \to \tau_2]\!](\varphi)(n)$$

and

$$(E, E') \in [\![\Delta \vdash \tau_2]\!](\varphi)^{\top}(n)$$

then

$$(E \circ (v\,[\,]), E' \circ (v'\,[\,])) \in [\![\Delta \vdash \tau_1]\!](\varphi)^{\top}(n).$$

◊

This follows directly from the definition of the interpretation of types.

Corollary 2.A.8. *Let* $n \in \mathbb{N}$. *If*

$$(e, e') \in [\![\Delta \vdash \tau_1]\!](\varphi)^{\top\top}(n)$$

and

$$(E, E') \in [\![\Delta \vdash \tau_2]\!](\varphi)^{\top}(n)$$

then

$$(E \circ ([\,]\,e), E' \circ ([\,]\,e')) \in [\![\Delta \vdash \tau_1 \to \tau_2]\!](\varphi)^{\top}(n).$$

◊

Proof. Let $n \in \mathbb{N}$. Take $(v, v') \in [\![\Delta \vdash \tau_1 \to \tau_2]\!](\varphi)(n)$. By Lemma 2.A.7 and monotonicity we have for all $k \leq n$, $(E \circ (v\,[\,]), E' \circ (v'\,[\,])) \in [\![\Delta \vdash \tau_1]\!](\varphi)^{\top}(k)$ and by the assumption that $(e, e') \in [\![\Delta \vdash \tau_1]\!](\varphi)^{\top\top}(n)$ we have

$$\mathfrak{P}_k^{\Downarrow}(E[v\,e]) \leq \mathfrak{P}^{\Downarrow}(E'[v'\,e'])$$

concluding the proof. QED

Lemma 2.A.9. *Let* $n \in \mathbb{N}$. *If* $(E, E') \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)^{\top}(n)$ *then*

$$(E \circ (\texttt{unfold}\,[\,]), E' \circ (\texttt{unfold}\,[\,])) \in [\![\Delta \vdash \mu\alpha.\tau]\!](\varphi)^{\top}(n).$$

◊

Proof. Let $n \in \mathbb{N}$. We consider two cases.

- $n = m+1$
  Take $(\mathtt{fold}\, v, \mathtt{fold}\, v') \in [\![\Delta \vdash \mu\alpha.\tau]\!](\varphi)(n)$. By definition
  $$(v,v') \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)(m).$$
  Let $k \leq n$. If $k = 0$ the condition is trivially true since
  $$\mathfrak{P}^{\Downarrow}_{k}(E[\mathtt{unfold}\, \mathtt{fold}\, v]) = 0$$
  so assume $k = \ell + 1$. Note that crucially $\ell \leq m$. Using Lemma 2.4.3, Lemma 2.3.4 and Lemma 2.3.1 we have
  $$\begin{aligned}\mathfrak{P}^{\Downarrow}_{k}(E[\mathtt{unfold}\,(\mathtt{fold}\, v)]) &= \mathfrak{P}^{\Downarrow}_{\ell}(E[v])\\ &\leq \mathfrak{P}^{\Downarrow}(E'[v'])\\ &= \mathfrak{P}^{\Downarrow}(E'[\mathtt{unfold}\,(\mathtt{fold}\, v')])\end{aligned}$$
  concluding the proof.
- $n = 0$. This case is trivial, since $\mathfrak{P}^{\Downarrow}_{0}(e) = 0$ for any e.

𝔔𝔈𝔇

Lemma 2.A.10. *Let $n \in \mathbb{N}$. If $(E,E') \in [\![\Delta \vdash \mu\alpha.\tau]\!](\varphi)^{\top}(n)$ then*
$$(E \circ (\mathtt{fold}\,[]), E' \circ (\mathtt{fold}\,[])) \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)^{\top}(n).$$

◊

Proof. Easily follows from the fact that if (v,v') are related at the unfolded type then $(\mathtt{fold}\, v, \mathtt{fold}\, v')$ are related at the folded type (using weakening to get to the same stage). 𝔔𝔈𝔇

Lemma 2.A.11 (Functional extensionality for values)**.** *Suppose $\tau,\sigma \in \mathfrak{T}(\Delta)$ and let $\lambda x.e$ and $\lambda x'.e'$ be two values of type $\tau \to \sigma$ in context $\Delta \mid \Gamma$. If for all $u \in$* **Val** (τ) *we have $\Delta \mid \Gamma \vdash (\lambda x.e)\, u \lesssim^{ctx} (\lambda x'.e')\, u : \sigma$ then*
$$\Delta \mid \Gamma \vdash \lambda x.e \lesssim^{ctx} \lambda x'.e' : \tau \to \sigma \quad .$$

◊

Proof. We use Theorem 2.4.10 several times and show $\lambda x.e$ and $\lambda x'.e'$ are logically related. Let $n \in \mathbb{N}$, $\varphi \in$ **VRel** (Δ) and $(\gamma,\gamma') \in [\![\Delta \vdash \Gamma]\!](\varphi)(n)$. Let $v = \lambda x.e\gamma$ and $v' = \lambda x'.e'\gamma'$. We are to show $(v,v') \in [\![\Delta \vdash \tau \to \sigma]\!](\varphi)^{\top\top}(n)$ and to do this we show directly $(v,v') \in [\![\Delta \vdash \tau \to \sigma]\!](\varphi)(n)$.

Let $j \leq n$, $(u,u') \in [\![\tau]\!](\varphi)(n)$, $k \leq j$ and $(E,E') \in [\![\sigma]\!](\varphi)^{\top}(k)$. We have to show $\mathfrak{P}^{\Downarrow}_{k}(E[v\, u]) \leq \mathfrak{P}^{\Downarrow}(E'[v'\, u'])$. From Proposition 2.4.5 we have that $(v,v) \in [\![\tau \to \sigma]\!](\varphi)^{\top\top}(n)$ and so $\mathfrak{P}^{\Downarrow}_{k}(E[v\, u]) \leq \mathfrak{P}^{\Downarrow}(E'[v\, u'])$. From the assumption of the lemma we have that $v\, u' \lesssim^{CIU} v'\, u'$ which concludes the proof. 𝔔𝔈𝔇

Lemma 2.A.12 (Extensionality for the universal type)**.** *Let $\tau \in \mathfrak{T}(\Delta, \alpha)$ be a type. Let $\Lambda.e, \Lambda.e'$ be two terms of type $\forall\alpha.\tau$ in context $\Delta \mid \Gamma$. If for all closed types $\sigma \in \mathfrak{T}$ we have*

$$\Delta \mid \Gamma \vdash e \lesssim^{ctx} e' : \tau[\sigma/\alpha]$$

then $\Delta \mid \Gamma \vdash \Lambda.e \lesssim^{ctx} \Lambda.e' : \forall\alpha.\tau$. $\Diamond$

Proof. We again use Theorem 2.4.10 multiple times. Let $n \in \mathbb{N}$, $\varphi \in \mathbf{VRel}(\Delta)$ and $(\gamma, \gamma') \in \llbracket \Delta \vdash \Gamma \rrbracket(\varphi)(n)$. Let $v = \Lambda.e\gamma$ and $v' = \Lambda.e'\gamma'$. We show directly that $(v, v') \in \llbracket \Delta \vdash \forall\alpha.\tau \rrbracket(\varphi)(n)$.

So take $\sigma, \sigma' \in \mathfrak{T}$ and $r \in \mathbf{VRel}(\sigma, \sigma')$ and we need to show $(e\gamma, e'\gamma') \in \llbracket \Delta, \alpha \rrbracket(\varphi[\alpha \mapsto r])^{\top\top}(n)$. Let $k \leq n$ and (E, E') related at k. We have to show $\mathfrak{P}_k^{\Downarrow}(E[e\gamma]) \leq \mathfrak{P}^{\Downarrow}(E'[e'\gamma'])$. From Proposition 2.4.5 we have

$$(e\gamma, e\gamma') \in \llbracket \Delta, \alpha \rrbracket(\varphi[\alpha \mapsto r])^{\top\top}(n)$$

and so $\mathfrak{P}_k^{\Downarrow}(E[e\gamma]) \leq \mathfrak{P}^{\Downarrow}(E'[e\gamma'])$. Let $\vec{\sigma}$ be the types for the right hand side in φ. Then $E' \in \mathbf{Stk}(\tau[\vec{\sigma}, \sigma'/\Delta, \alpha])$. Using the assumption of the lemma we get that $e\gamma' \lesssim^{CIU} e'\gamma'$ at the type $\tau[\vec{\sigma}, \sigma'/\Delta, \alpha]$ which immediately implies that $\mathfrak{P}^{\Downarrow}(E'[e\gamma']) \leq \mathfrak{P}^{\Downarrow}(E'[e'\gamma'])$ concluding the proof. 𝔔𝔈𝔇

2.B Probability of Termination

We prove the claims from Section 2.5 about the termination probability.

Proposition 2.B.1. *For any expression e, $\mathfrak{P}^{\Downarrow}(e)$ is a left-computable real number.* $\Diamond$

Proof. We first prove by induction that for any n, $\Phi^n(\bot)$ restricts to a map $\mathbf{Tm} \to [0,1] \cap \mathbb{Q}$. The proof is simple since the function $\bot$ clearly maps into rationals and for the inductive step we use the fact that the sums in the definition of Φ are always finite, and the rational numbers are closed under finite sums.

To conclude the proof we have by definition that $\mathfrak{P}^{\Downarrow}(e) = \sup_{n \in \omega} \Phi^n(\bot)(e)$ and we have just shown that all the numbers $\Phi^n(\bot)(e)$ are rational. Moreover the sequence $\{\Phi^n(\bot)(e)\}_{n \in \mathbb{N}}$ is computable, since for a given n we only need to check all the reductions from e of length at most n to determine the value of $\Phi^n(\bot)(e)$ and the reduction relation $\overset{p}{\rightsquigarrow}$ is naturally computable. 𝔔𝔈𝔇

Example 2.B.2. To see that the probability of termination can also be non-computable we describe a program whose probability of termination would allow us to solve the halting problem were it computable.

The program we construct is recursively defined as $T = \mathtt{fix}[][]\,\varphi$ where

$$\varphi = \lambda f.\lambda x.t\,x \oplus (\Omega \oplus f\,(\mathsf{succ}\,x))$$

where $t\,x$ is a program that runs the x-th Turing machine on the empty input and does not use any choice reductions. Thus $\mathfrak{P}^{\Downarrow}(t\,x) \in \{0,1\}$. It is well known that the empty string acceptance problem is undecidable. Note that we put Ω in the program to ensure that every second digit in binary will be 0. It is an easy computation to show that

$$\mathfrak{P}^{\Downarrow}(T\,\underline{1}) = \sum_{n=0}^{\infty} \frac{1}{2^{2n+1}} p_{n+1}$$

where $p_n = 1$ if the n-th Turing machine terminates on the empty input and 0 otherwise. If $\mathfrak{P}^{\Downarrow}(T\,\underline{1})$ were computable we could decide whether a given Turing machine accepts the empty string by computing its index n and then computing the first $2n$ digits of $\mathfrak{P}^{\Downarrow}(T\,\underline{1})$. ♦

We now generalise the last example and show that any left-computable real arises as the probability of termination of a program. Technically, we show that given a term of the language that computes an increasing bounded sequence of rationals (represented as pairs of naturals) we can define a program that terminates with probability the supremum of the sequence. We then use the fact that our language $\mathbf{F}^{\mu,\oplus}$ is Turing complete to claim that any computable sequence of rationals can be represented as such a term of $\mathbf{F}^{\mu,\oplus}$.

Proposition 2.B.3. *For every left-computable real in* $[0,1]$ *there is a program* e_r *of type* $\mathbf{1} \to \mathbf{1}$ *such that* $\mathfrak{P}^{\Downarrow}(e_r\,\langle\rangle) = r$. ◊

Proof. So let $r : \mathtt{nat} \to \mathtt{nat} \times \mathtt{nat}$ compute an increasing sequence of rationals in the interval $[0,1]$. Additionally assume that for all $n \in \mathbb{N}$.

$$r\,\underline{n} \overset{\mathrm{cf}}{\Longrightarrow} \langle \underline{k_n}, \underline{\ell_n} \rangle$$

for some $k_n, \ell_n \in \mathbb{N}$. That is, r does not use choice reductions. This is not an essential limitation, but simplifies the argument which we are about to give.

First we define a recursive function e of type $e : (\mathtt{nat} \to \mathtt{nat} \times \mathtt{nat}) \to \mathbf{1}$ as $e = \mathtt{fix}[][]\,\varphi$ where

$$\begin{aligned} \varphi = \lambda f.\lambda r.&\mathtt{let}\ (\underline{k},\underline{\ell})\ =\ r\,\underline{1}\ \mathtt{in} \\ &\mathtt{let}\ y\ =\ \mathtt{rand}\,\underline{\ell}\ \mathtt{in} \\ &\mathtt{if}\ y \leq \underline{k}\ \mathtt{then}\ \langle\rangle\ \mathtt{else}\ f\,r' \end{aligned}$$

and

$$r' = \lambda z.\frac{r\,(\mathtt{succ}\,z) - (\underline{k},\underline{\ell})}{1 - (\underline{k},\underline{\ell})}$$

and subtraction and division is implemented in the obvious way. Note that the condition in φ ensures that $(\underline{k},\underline{\ell})$ does not represent the rational number

1 and therefore division would make sense. But technically, since we implement rationals with pairs of naturals no exception can occur and we just represent the pair with the second component being $\underline{0}$.

Let f and r be values of the appropriate type. We have

$$\mathfrak{p}^{\Downarrow}_{m+1}(\varphi\, f\, r) \leq \frac{k_1}{\ell_1}\mathfrak{p}^{\Downarrow}_{m}(\langle\rangle) + \frac{\ell_1 - k_1}{\ell_1} \cdot \mathfrak{p}^{\Downarrow}_{m}(f\, r')$$

where $r\,\underline{1} \overset{\text{cf}}{\Longrightarrow} (\underline{k}_1, \underline{\ell}_1)$. The inequality comes from the fact that applying r might take some unfold-fold reductions. Iterating this we get

$$\mathfrak{p}^{\Downarrow}_{m+1+2n}(e\, r) \leq \frac{k_n}{\ell_n} + \frac{\ell_n - k_n}{\ell_n} \cdot \mathfrak{p}^{\Downarrow}_{m+1}\left(e\, r^{(n)}\right)$$

where $r\,\underline{n} \overset{\text{cf}}{\Longrightarrow} (\underline{k}_n, \underline{\ell}_n)$ and

$$r^{(n)} = \lambda z.\frac{r(\mathsf{succ}^n\, z) - (\underline{k}_n, \underline{\ell}_n)}{1 - (\underline{k}_n, \underline{\ell}_n)}$$

is the n-th iteration of the $'$ used on r in φ.

It is easy to see that $\mathfrak{p}^{\Downarrow}_{1}\left(e\, r^{(n)}\right) = 0$ since it takes at least one unfold-fold and one choice reduction to terminate. Thus picking $m = 1$ we have $\mathfrak{p}^{\Downarrow}_{2+2n}(e\, r) = \frac{k_n}{\ell_n}$ and thus

$$\sup_{n \in \omega} \mathfrak{p}^{\Downarrow}_{n}(e\, r) \leq \sup_{n \in \omega} \frac{k_n}{\ell_n}$$

Using the same reasoning as above we also have

$$\mathfrak{p}^{\Downarrow}(e\, r) \geq \frac{k_n}{\ell_n} + \frac{\ell_n - k_n}{\ell_n} \cdot \mathfrak{p}^{\Downarrow}\left(e\, r^{(n)}\right) \geq \frac{k_n}{\ell_n}$$

which shows (using Proposition 2.3.5) that

$$\sup_{n \in \omega} \frac{k_n}{\ell_n} \leq \mathfrak{p}^{\Downarrow}(e\, r) \leq \sup_{n \in \omega} \mathfrak{p}^{\Downarrow}_{n}(e\, r) \leq \sup_{n \in \omega} \frac{k_n}{\ell_n}$$

and so

$$\sup_{n \in \omega} \frac{k_n}{\ell_n} = \mathfrak{p}^{\Downarrow}(e\, r).$$

𝔔𝔈𝔇

2.C Distributions

We now define distributions and prove some of their properties and properties of the probability of termination which are used in some of the examples.

By a distribution we mean a subprobability measure on the discrete space **Val** of values. Let

$$\mathbf{Dist} = \left\{ f : \mathbf{Val} \to [0,1] \middle| \sum_{v \in \mathbf{Val}} f(v) \leq 1 \right\}$$

be the space of subprobability measures on **Val**. To be precise, $f \in \mathbf{Dist}$ are not measures, but given any f we can define a subprobability measure $\mu_f(A) = \sum_{v \in A} f(v)$ and given any subprobability measure μ, we can define $f_\mu \in \mathbf{Dist}$ as the Radon-Nikodym derivative with respect to the counting measure. Or in more prosaic terms $f_\mu(v) = \mu(\{v\})$. It is easy to see that these two operations are mutually inverse and since $f \in \mathbf{Dist}$ are easier to work with we choose this presentation.

Lemma 2.C.1. **Dist** *ordered pointwise is a pointed ω-cpo.* ◊

Proof. The bottom element is the everywhere 0 function. Let $\{f_n\}_{n \in \omega}$ be an ω-chain. Define the limit function f as the pointwise supremum

$$f(v) = \sup_{n \in \omega} f_n(v).$$

Clearly all pointwise suprema exist and f is the least upper bound, provided we can show that $f \in \mathbf{Dist}$. To show this last fact we need to show

$$\sum_{v \in \mathbf{Val}} \sup_{n \in \omega} f_n(v) \leq 1.$$

but this is a simple consequence of Fatou's lemma since from the assumption that $\{f_n\}_{n \in \omega}$ we have $\sup_{n \in \omega} f_n(v) = \lim_{n \to \infty} f_n(v) = \liminf_{n \to \infty} f_n(v)$ and so by Fatou's lemma (relative to the counting measure on **Val**) we have

$$\sum_{v \in \mathbf{Val}} \sup_{n \in \omega} f_n(v) \leq \liminf_{n \to \infty} \left(\sum_{v \in \mathbf{Val}} f_n(v) \right) \leq \liminf_{n \to \infty} 1 = 1.$$

𝔔𝔈𝔇

Now define $\Xi : (\mathbf{Tm} \to \mathbf{Dist}) \to (\mathbf{Tm} \to \mathbf{Dist})$ as follows

$$\Xi(\varphi)(e) = \begin{cases} \delta_e & \text{if } e \in \mathbf{Val} \\ \sum_{e \overset{p}{\leadsto} e'} p \cdot \varphi(e') & \text{otherwise} \end{cases}$$

where δ_e is (the density function of) the Dirac measure at point e. Since **Dist** is an ω-cpo so is $\mathbf{Tm} \to \mathbf{Dist}$ ordered pointwise. It is easy to see that in this ordering Ξ is monotone and continuous and so by Kleene's fixed point theorem it has a least fixed point reached in ω iterations. Let $\mathcal{D} = \sup_{n\in\omega}(\Xi^n(\bot))$ be this fixed point.

Lemma 2.C.2. *Let $e \in \mathbf{Tm}$ and $v \in \mathbf{Val}$. If $\mathcal{D}(e)(v) > 0$ then there exists a path π from e to v, i.e. e steps to v.* $\Diamond$

Proof. We use Scott induction. Define

$$\mathcal{S} = \{f : \mathbf{Tm} \to \mathbf{Dist} \mid \forall e, v, f(e)(v) > 0 \to \exists \pi, \pi : e \rightsquigarrow^* v\}$$

The set $\mathcal{S}$ contains $\bot$. To see that it is closed under ω-chains observe that if $(\sup_{n\in\omega} f_n)(e)(v) > 0$ then there must be $n \in \omega$, such that $f_n(e)(v) > 0$ so we may use the path from e to v that we know exists from the assumption that $f_n \in \mathcal{S}$.

It is similarly easy to see that given $f \in \mathcal{S}$ we have $\Xi(f) \in \mathcal{S}$. Thus we have that $\mathcal{D} \in \mathcal{S}$ concluding the proof. $\mathfrak{QED}$

Lemma 2.C.3. *For any expression $e \in \mathbf{Tm}$ we have*

$$\sum_{v\in\mathbf{Val}} \mathcal{D}(e)(v) = \mathfrak{P}^{\Downarrow}(e)$$

$\Diamond$

Proof. First we show by induction on n that all the finite approximations of $\mathfrak{P}^{\Downarrow}(e)$ and $\mathcal{D}(e)$ agree.

- The base case is trivial since by definition

$$\sum_{v\in\mathbf{Val}} \Xi^0(\bot)(e)(v) = 0 = \Phi^0(\bot)(e)$$

- For the inductive case we consider two cases. If $e \in \mathbf{Val}$ then both sides are 1. In the other case we have

$$\begin{aligned}\sum_{v\in\mathbf{Val}} \Xi^{n+1}(\bot)(e)(v) &= \sum_{v\in\mathbf{Val}} \left(\sum_{e \overset{p}{\rightsquigarrow} e'} p \cdot \Xi^n(e') \right)(v) \\ &= \sum_{v\in\mathbf{Val}} \left(\sum_{e \overset{p}{\rightsquigarrow} e'} p \cdot \Xi^n(e')(v) \right)\end{aligned}$$

by Tonelli's theorem we can we can interchange the sums to get

$$
\begin{aligned}
&= \sum_{e \overset{p}{\rightsquigarrow} e'} \left(p \sum_{v \in \mathbf{Val}} \Xi^n(e')(v) \right) \\
&= \sum_{e \overset{p}{\rightsquigarrow} e'} p \cdot \Phi^n(\bot)(e') = \Phi^{n+1}(\bot)(e)
\end{aligned}
$$

Thus we have that for all n,

$$\sum_{v \in \mathbf{Val}} \Xi^n(\bot)(e)(v) = \Phi^n(\bot)(e)$$

and so

$$\sup_{n \in \omega} \left(\sum_{v \in \mathbf{Val}} \Xi^n(\bot)(e)(v) \right) = \sup_{n \in \omega} (\Phi^n(\bot)(e)) = \mathfrak{P}^{\Downarrow}(e)$$

By the dominated convergence theorem we can exchange the sup (which is the limit) and the sum on the left to get

$$
\begin{aligned}
\sup_{n \in \omega} \left(\sum_{v \in \mathbf{Val}} \Xi^n(\bot)(e)(v) \right) &= \sum_{v \in \mathbf{Val}} \sup_{n \in \omega} (\Xi^n(\bot)(e)(v)) \\
&= \sum_{v \in \mathbf{Val}} \mathcal{D}(e)(v)
\end{aligned}
$$

as required. 𝔔𝔈𝔇

Proposition 2.C.4 (Monadic bind for distributions). *Let $e \in \mathbf{Tm}$ and E an evaluation context of appropriate type.*

$$\mathcal{D}(E[e]) = \sum_{v \in \mathbf{Val}} \mathcal{D}(e)(v) \cdot \mathcal{D}(E[v]).$$

◊

Proof. It is easy to show by induction on ℓ that

$$\forall e \in \mathbf{Tm}, \Xi^{\ell}(\bot)(E[e]) = \sum_{v \in \mathbf{Val}} \sum_{\substack{\pi : e \rightsquigarrow^* v \\ \mathbf{len}(\pi) \leq \ell}} \mathfrak{W}(\pi) \cdot \Xi^{\ell - \mathbf{len}(\pi)}(E[v]) \tag{2.5}$$

(using the fact that the length of the empty path is 0 and its weight 1).

Similarly it is easy to show by induction on ℓ that

$$\forall e \in \mathbf{Tm}, \Xi^{\ell+1}(\bot)(e)(v) = \sum_{\substack{\pi : e \rightsquigarrow^* v \\ \mathbf{len}(\pi) \leq \ell}} \mathfrak{W}(\pi) \tag{2.6}$$

which immediately implies

$$\forall e \in \mathbf{Tm}, \mathcal{D}(e)(v) = \sum_{\pi : e \rightsquigarrow^* v} \mathfrak{W}(\pi) \tag{2.7}$$

Using these we have

$$\mathcal{D}(E[e]) = \sup_{\ell \in \omega} \sum_{v \in \mathbf{Val}} \sum_{\substack{\pi : e \rightsquigarrow^* v \\ \mathbf{len}(\pi) \leq \ell}} \mathfrak{W}(\pi) \cdot \Xi^{\ell - \mathbf{len}(\pi)}(E[v])$$

and since for each v the sequence $\sum_{\substack{\pi : e \rightsquigarrow^* v \\ \mathbf{len}(\pi) \leq \ell}} \mathfrak{W}(\pi) \cdot \Xi^{\ell - \mathbf{len}(\pi)}(E[v])$ is increasing with ℓ we have

$$\begin{aligned}
&= \sum_{v \in \mathbf{Val}} \sup_{\ell \in \omega} \sum_{\substack{\pi : e \rightsquigarrow^* v \\ \mathbf{len}(\pi) \leq \ell}} \mathfrak{W}(\pi) \cdot \Xi^{\ell - \mathbf{len}(\pi)}(E[v]) \\
&= \sum_{v \in \mathbf{Val}} \sum_{\pi : e \rightsquigarrow^* v} \mathfrak{W}(\pi) \cdot \mathcal{D}(E[v]) \\
&= \sum_{v \in \mathbf{Val}} \mathcal{D}(E[v]) \sum_{\pi : e \rightsquigarrow^* v} \mathfrak{W}(\pi) \\
&= \sum_{v \in \mathbf{Val}} \mathcal{D}(e)(v) \cdot \mathcal{D}(E[v])
\end{aligned}$$

$\mathfrak{QED}$

Corollary 2.C.5. *Let $e \in \mathbf{Tm}$ be typeable and E an evaluation context of appropriate type. Then $\mathfrak{P}^{\Downarrow}(E[e]) = \sum_{\pi : e \rightsquigarrow^* v} \mathcal{W}(\pi) \cdot \mathfrak{P}^{\Downarrow}(E[v])$.* $\Diamond$

Corollary 2.C.6. *For any term e and evaluation context E the equality*

$$\mathfrak{P}^{\Downarrow}(E[e]) = \sum_{v \in \mathbf{Val}} \mathcal{D}(e)(v) \cdot \mathfrak{P}^{\Downarrow}(E[v])$$

holds. $\Diamond$

Corollary 2.C.7. *Let $e \in \mathbf{Tm}$ and E an evaluation context. Suppose $\mathcal{D}(e) = p \cdot \delta_v$ for some $v \in \mathbf{Val}$ and $p \in [0,1]$. Then $\mathfrak{P}^{\Downarrow}(E[e]) = p \cdot \mathfrak{P}^{\Downarrow}(E[v])$.* $\Diamond$

Proof. Use Proposition 2.C.4 and Lemma 2.C.3. $\mathfrak{QED}$

Proposition 2.C.8. *For any evaluation context E and term e and any $k \in \mathbb{N}$,*

$$\mathfrak{P}_k^{\Downarrow}(E[e]) \leq \sum_{\pi : e \rightsquigarrow^* v} \mathfrak{W}(\pi) \cdot \mathfrak{P}_k^{\Downarrow}(E[v])$$

$\Diamond$

The proof proceeds by induction on k.

2.D Further Examples

In this section we show further equivalences which did not fit into the paper proper due to space restrictions.

Fair coin from an unfair one

Given an unfair coin, that is, a coin that comes up heads with probability p and tails with probability $1-p$, where $0 < p < 1$ we can derive an infinite sequence of fair coin tosses using the procedure proposed by von Neumann. The procedure follows from the observation that if we toss an unfair coin twice, the likelihood of getting (H, T) is the same as the likelihood of getting (T, H). So the procedure works as follows

- Toss the coin twice
- If the result is (H, T) or (T, H) return the result of the first toss
- Else repeat the process

We only consider rational p in this section (for a computable p we could proceed similarly, but the details would be more involved, since the function which returns 1 with probability p and 0 with probability $1-p$ is a bit more challenging to write).

Let $1 \leq k < n$ be two natural numbers and $p = \frac{k}{n}$. Below we define $e_p : \mathbf{1} \to \mathbf{2}$ to be the term implementing the von Neumann procedure for generating fair coin tosses from an unfair coin t_p which returns `true` with probability p and `false` with probability $1-p$. We will show that e_p is contextually equivalent to $\lambda x.\texttt{true} \oplus \texttt{false}$. We define e_p as

$$e_p = \texttt{fix}[][]\varphi$$

where

$$\begin{aligned}
\mathbf{2} &= \mathbf{1} + \mathbf{1} \\
\texttt{true} &= \texttt{inl}\,\langle\rangle \\
\texttt{false} &= \texttt{inr}\,\langle\rangle \\
e \equiv e' &= \texttt{match}(e, \_.e', \_.\texttt{match}(e', \_.\texttt{false}, \_.\texttt{true})) \\
\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 &= \texttt{match}(e, \_.e_1, \_.e_2) \\
t_p &= \lambda\langle\rangle.\texttt{let}\ y\ = \texttt{rand}\,\underline{n}\ \texttt{in}\ (y \leq \underline{k})
\end{aligned}$$

and

$$\begin{aligned}
\varphi = \lambda f.\lambda\langle\rangle.&\texttt{let}\ x\ =\ t_p\,\langle\rangle\ \texttt{in} \\
&\texttt{let}\ y\ =\ t_p\,\langle\rangle\ \texttt{in} \\
&\texttt{if}\ x \equiv y\ \texttt{then}\ f\,\langle\rangle\ \texttt{else}\ x.
\end{aligned}$$

By a simple calculation using the operational semantics we can see that given any evaluation context E, we have

$$\mathfrak{p}^{\Downarrow}\left(E[t_p\,\langle\rangle]\right) = \frac{k}{n}\mathfrak{p}^{\Downarrow}(E[\mathtt{true}]) + \frac{n-k}{n}\mathfrak{p}^{\Downarrow}(E[\mathtt{false}]).$$

Given any value f of type $(\mathbf{1} \to 2)$ and any evaluation context E with the hole of type 2 we compute that $\mathfrak{p}^{\Downarrow}(E[\varphi\, f\,\langle\rangle])$ is equal to $\frac{k^2+(n-k)^2}{n^2}\mathfrak{p}^{\Downarrow}(E[f\,\langle\rangle]) + 2 \cdot \frac{k\cdot(n-k)}{n^2}\mathfrak{p}^{\Downarrow}(E[\mathtt{true} \oplus \mathtt{false}])$. Finally for e_p and any evaluation context E with hole of type 2 we have

$$\mathfrak{p}^{\Downarrow}\left(E[e_p\,\langle\rangle]\right) = \mathfrak{p}^{\Downarrow}\left(\varphi\, e_p\,\langle\rangle\right) = \frac{k^2+(n-k)^2}{n^2}\mathfrak{p}^{\Downarrow}\left(E[e_p\,\langle\rangle]\right) + 2 \cdot \frac{k\cdot(n-k)}{n^2}\mathfrak{p}^{\Downarrow}(E[\mathtt{true} \oplus \mathtt{false}]).$$

from which we have by simple algebraic manipulation that $\mathfrak{p}^{\Downarrow}\left(E[e_p\,\langle\rangle]\right) = \mathfrak{p}^{\Downarrow}(E[\mathtt{true} \oplus \mathtt{false}])$.

It is now straightforward to show $\varnothing \mid \varnothing \vdash e_p \cong^{log} \lambda\langle\rangle.\mathtt{true} \oplus \mathtt{false} : \mathbf{1} \to 2$ since both e_p and $\lambda\langle\rangle.\mathtt{true} \oplus \mathtt{false}$ are values, so we can show them related in the value relation. The proof uses reflexivity of $\cong^{log}$.

Alternatively, we could have used Theorem 2.4.10 and showed directly that $e_p\,\langle\rangle$ and $\mathtt{true} \oplus \mathtt{false}$ are CIU-equivalent and then used extensionality for values to conclude the proof.

A hesitant identity function

We consider the identity function e that does not return immediately, but instead when applied to a value v flips a coin whether to return v or call itself recursively with the same argument. We show that this function is contextually equivalent to the identity function $\lambda x.x$. The reason for this is, intuitively, that even though e when applied may diverge, the probability of it doing so is 0.

Example 2.D.1. Let $e = \mathtt{fix}[][](\lambda f.\lambda x.(x \oplus f\,x)) : \alpha \to \alpha$. We have

$$\alpha \mid \varnothing \vdash e \lesssim^{log} \lambda x.x : \alpha \to \alpha$$

and

$$\alpha \mid \varnothing \vdash \lambda x.x \lesssim^{log} e : \alpha \to \alpha.$$

♦

Proof. We prove the two approximations separately. Let $\varphi \in \mathbf{VRel}(\alpha)$, $n \in \mathbb{N}$. Since e and $\lambda x.x$ are values we show them directly related in the value relation. In both cases let $\varphi = \lambda f.\lambda x.(x \oplus f\,x)$ and $h = \lambda z.\delta_\varphi\,(\mathtt{fold}\,\delta_\varphi)\,z$.

- By definition of the interpretation of function types we have to show, given $k \leq n$ and $(v,v') \in \varphi_r(\alpha)(k)$, that $(e\,v, (\lambda x.x)\,v') \in \varphi_r(\alpha)^{\top\top}(k)$.

  It is straightforward to see that $e\,v \overset{\mathrm{cf}}{\Longrightarrow} \varphi\,e\,v$ using exactly one unfold-fold reduction.

  Now let (E,E') be related at k. We proceed by induction and show that for every $\ell \leq k$, $\mathfrak{P}^{\Downarrow}_{\ell}(E[e\,v]) \leq \mathfrak{P}^{\Downarrow}(E'[v'])$ which suffices by Lemma 2.3.1. When $\ell = 0$ there is nothing to prove. So let $\ell = \ell' + 1$.

  $$\mathfrak{P}^{\Downarrow}_{\ell}(E[e\,v]) = \mathfrak{P}^{\Downarrow}_{\ell'}(\varphi\,e\,v) = \mathfrak{P}^{\Downarrow}_{\ell'}(E[v \oplus e\,v]).$$

  If $\ell' = 0$ we are trivially done. So suppose $\ell' = \ell'' + 1$ to get using Lemma 2.3.4

  $$\mathfrak{P}^{\Downarrow}_{\ell'}(E[v \oplus e\,v]) = \frac{1}{2}\mathfrak{P}^{\Downarrow}_{\ell''}(E[v]) + \frac{1}{2}\mathfrak{P}^{\Downarrow}_{\ell''}(e\,v)$$

  Using the fact that $\ell'' \leq k$ and monotonicity we have

  $$\mathfrak{P}^{\Downarrow}_{\ell''}(E[v]) \leq \mathfrak{P}^{\Downarrow}(E'[v']).$$

  Using the induction hypothesis we have

  $$\mathfrak{P}^{\Downarrow}_{\ell''}(e\,v) \leq \mathfrak{P}^{\Downarrow}(E'[v'])$$

  which together conclude the proof.

- Again by definition of the interpretation of function types we have to show, given $k \leq n$ and $(v,v') \in \varphi_r(\alpha)(k)$, that $((\lambda x.x)\,v', e\,v) \in \varphi_r(\alpha)^{\top\top}(k)$.

  Again we have that $e\,v' \overset{\mathrm{cf}}{\Longrightarrow} \varphi\,e\,v'$ using exactly one unfold-fold reduction. Let $\ell \leq k$ and (E,E') related at ℓ. Using Lemma 2.3.1 and the fact that $\mathfrak{P}^{\Downarrow}(\cdot)$ is a fixed point of Φ we have

  $$\begin{aligned}\mathfrak{P}^{\Downarrow}(E'[e\,v']) &= \mathfrak{P}^{\Downarrow}(E'[\varphi\,e\,v']) \\ &= \frac{1}{2}\mathfrak{P}^{\Downarrow}(E'[v']) + \frac{1}{2}\mathfrak{P}^{\Downarrow}(E'[e\,v'])\end{aligned}$$

  and from this we get $\frac{1}{2}\mathfrak{P}^{\Downarrow}(E'[e\,v']) = \frac{1}{2}\mathfrak{P}^{\Downarrow}(E'[v'])$ by simple algebraic manipulation and thus $\mathfrak{P}^{\Downarrow}(E'[e\,v']) = \mathfrak{P}^{\Downarrow}(E'[v'])$. Using this property it is a triviality to finish the proof.

$\mathfrak{QED}$

Further simple examples

The following example is a proof of *perfect security* for the one-time pad encryption scheme. Define the following functions

$$
\begin{aligned}
&\texttt{not} : 2 \to 2\\
&\texttt{not} = \lambda x.\texttt{if}\ x\ \texttt{then}\ \texttt{false}\ \texttt{else}\ \texttt{true}\\
&\texttt{xor} : 2 \to 2 \to 2\\
&\texttt{xor} = \lambda x.\lambda y.\texttt{if}\ x\ \texttt{then}\ \texttt{not}\ y\ \texttt{else}\ y\\
&\texttt{gen} : 2\\
&\texttt{gen} = \texttt{true} \oplus \texttt{false}
\end{aligned}
$$

xor is supposed to be the encryption function, with the first argument the plaintext and the second one the encryption key.

We now encode a game with two players. The first player chooses two plaintexts and gives them to the second player, who encrypts one of them (using xor) chosen at random with uniform probability and gives the result back to the first player. The first player should not be able to guess which of the plaintexts was encrypted. This is expressed as contextual equivalence of the following two programs

$$
\begin{aligned}
\texttt{exp} &= \lambda x.\lambda y.\texttt{xor}\ (x \oplus y)\ \texttt{gen}\\
\texttt{rnd} &= \lambda x.\lambda y.\texttt{gen}
\end{aligned}
$$

To show $\texttt{exp} =^{ctx} \texttt{rnd}$ we first use extensionality for values so we only need to show that for all $v, u \in \mathbf{Val}(2)$

$$\texttt{xor}\ (v \oplus u)\ \texttt{gen} =^{ctx} \texttt{gen}$$

and the easiest way to do this is by using CIU equivalence. Given an evaluation context E we have

$$\mathfrak{P}^{\Downarrow}(E[\texttt{xor}\ (v \oplus u)\ \texttt{gen}]) = \frac{1}{4}\left(\begin{array}{l} \mathfrak{P}^{\Downarrow}(E[\texttt{xor}\ v\ \texttt{true}]) + \\ \mathfrak{P}^{\Downarrow}(E[\texttt{xor}\ v\ \texttt{false}]) + \\ \mathfrak{P}^{\Downarrow}(E[\texttt{xor}\ u\ \texttt{true}]) + \\ \mathfrak{P}^{\Downarrow}(E[\texttt{xor}\ u\ \texttt{false}]) \end{array}\right)$$

and by the canonical forms lemma u and v can be either true or false. It is easy to see that the sum evaluates to

$$\frac{1}{4}(2 \cdot \mathfrak{P}^{\Downarrow}(E[\texttt{true}]) + 2 \cdot \mathfrak{P}^{\Downarrow}(E[\texttt{false}]))$$

quickly leading to the desired conclusion.

If we had used the logical relation directly we would not need the canonical forms lemma, but then we would have to take care of step-indexing.

A similar example is when in one instance we choose to encrypt the first plaintext and in the second instance the second one. Since the key is generated uniformly at random, the first player should not be able to distinguish those two instances. Concretely, this is expressed as contextual equivalence of the following two programs

$$\begin{aligned}\mathtt{exp}_1 &= \lambda x.\lambda y.\mathtt{xor}\ x\ \mathtt{gen}\\ \mathtt{exp}_2 &= \lambda x.\lambda y.\mathtt{xor}\ y\ \mathtt{gen}\end{aligned}$$

The proof is basically the same as the one above. Use extensionality and then CIU equivalence.

Restrictions in the free theorem are necessary

We show that the free theorem in Section 2.5 does not hold without some assumptions on the behaviour of functions f and g.

First, if $f = (\lambda x.\underline{1}) \oplus (\lambda x.\underline{2})$, g is the identity function $\lambda x.x$ and xs is the list $[\langle\rangle, \langle\rangle]$ then the term $\mathtt{map}[][](f \circ g)\, xs$ can reduce to the list $[\underline{1}, \underline{2}]$, however the term $((\mathtt{map}[][] f) \circ (\mathtt{map}[][] g))\, xs$ cannot. The reason is that in the first case the reduction of f is performed for each element of the list separately, but in the latter case, f is first reduced to a value and then the same value is applied to all the elements of the list. Technically, the condition we need for f is that there exists a value f', such that $f =^{ctx} f'$, but this version is easily derived from the version stated above by congruence.

Second, if g diverges with a non-zero probability for some value v, we take m to be the constant function returning the empty list and the list xs to be the singleton list containing only the value v. Then, if f is any value, $m[][](f \circ g)\, xs$ reduces to the empty list with probability 1, however $((m[][] f \circ \mathtt{map}[][] g))\, xs$ reduces to the empty list with a probability smaller than 1, since g is still applied, since we are in a *call-by-value* language.

Third, if $g = \lambda x.\underline{1} \oplus \underline{2}$, f is the identity function and xs is the singleton list containing $\langle\rangle$ we take m to be the function that first appends the given list to itself and *then* applies $\mathtt{map}$ to it. We then have that $m[][](f \circ g)\, xs$ *can* reduce to the list $[\underline{1}, \underline{2}]$, but $((m[][] f) \circ (\mathtt{map}[][] g))\, xs$ cannot, since g is only mapped over the singleton list producing lists $[\underline{1}]$ and $[\underline{2}]$, which are then appended to themselves, giving lists $[\underline{1}, \underline{1}]$ and $[\underline{2}, \underline{2}]$.

And last, if m is not equivalent to a term of the form $\Lambda.\Lambda.\lambda x.e$ then the term on the left reduces to two different (not equivalent) values (or even diverges), but the term on the right does not. We can use this to construct a distinguishing evaluation context.

A property of map

The result in Section 2.5 does not allow us to conclude

$$\mathtt{map}[][]\,(f \circ g) =^{ctx} \mathtt{map}[][]\,f \circ \mathtt{map}[][]\,g.$$

for all $f \in \mathbf{Val}\,(\sigma \to \rho)$ and $g \in \mathbf{Val}\,(\tau \to \sigma)$, however we can show, using the definition of map, that this does in fact hold. By using extensionality (Lemma 2.4.11) we need to show for any list xs we have

$$\mathtt{map}[][]\,(f \circ g)\,xs =^{ctx} (\mathtt{map}[][]\,f \circ \mathtt{map}[][]\,g)\;xs.$$

If f and g are values, E an evaluation context and xs a list of length n, it is easy to see that

$$\mathfrak{P}^{\Downarrow}(E[\mathtt{map}\,f\,xs]) = \sum_{us}\left(\prod_{i=1}^{n}\mathcal{D}(f\,x_i)(u_i)\right)\cdot\mathfrak{P}^{\Downarrow}(E[us])$$

where the first sum is over all the lists of length n and x_i and u_i are the i-th elements of lists xs and us, respectively. This then gives us that

$$\mathfrak{P}^{\Downarrow}(E[\mathtt{map}\,f\,(\mathtt{map}\,g\,xs)])$$

is equal to

$$\begin{aligned}
&\sum_{vs}\left(\prod_{i=1}^{n}\mathcal{D}(g\,x_i)(v_i)\right)\cdot\mathfrak{P}^{\Downarrow}(E[\mathtt{map}\,f\,vs])\\
&=\sum_{vs}\left(\prod_{i=1}^{n}\mathcal{D}(g\,x_i)(v_i)\right)\cdot\left(\sum_{us}\left(\prod_{i=1}^{n}\mathcal{D}(f\,v_i)(u_i)\right)\cdot\mathfrak{P}^{\Downarrow}(E[us])\right)\\
&=\sum_{vs}\sum_{us}\left(\prod_{i=1}^{n}\mathcal{D}(g\,x_i)(v_i)\cdot\mathcal{D}(f\,v_i)(u_i)\right)\cdot\mathfrak{P}^{\Downarrow}(E[us]).
\end{aligned}$$

On the other hand, we have that $\mathfrak{P}^{\Downarrow}(E[\mathtt{map}\,(f \circ g)\,xs])$ is equal to

$$\sum_{us}\left(\prod_{i=1}^{n}\mathcal{D}((f \circ g)\,x_i)(u_i)\right)\cdot\mathfrak{P}^{\Downarrow}(E[us])$$

and

$$\mathcal{D}((f \circ g)\,x_i)(u_i) = \sum_{v}\mathcal{D}(g\,x_i)(v)\cdot\mathcal{D}(f\,v)(u_i)$$

together giving us

$$\sum_{us}\left(\prod_{i=1}^{n}\left(\sum_{v}\mathcal{D}(g\,x_i)(v)\cdot\mathcal{D}(f\,v)(u_i)\right)\right)\cdot\mathfrak{P}^{\Downarrow}(E[us])$$

which by Fubini's theorem and the fact that lists of length n correspond to n-tuples, is equal to

$$\sum_{us}\sum_{vs}\left(\prod_{i=1}^{n}(\mathcal{D}(g\,x_i)(v_i)\cdot\mathcal{D}(f\,v_i)(u_i))\right)\cdot\mathfrak{P}^{\Downarrow}(E[us])$$

which is the same as $\mathfrak{P}^{\Downarrow}(E[\mathtt{map}\,f\,(\mathtt{map}\,g\,xs)])$.

If f and g are not equivalent to values, then the above result for map does not hold. Consider, for instance, $f = \lambda x.\underline{1} \oplus \lambda x.\underline{2}$ and g the identity or conversely, when applied to the list $xs = [\langle\rangle, \langle\rangle]$. The expression $\mathtt{map}[][](f \circ g)\,xs$ can reduce to the list $[1, 2]$, whereas the expression $(\mathtt{map}[][]\,f \circ \mathtt{map}[][]\,g)\,xs$ cannot. We can generalise this to show that if f is not equivalent to a value or g is not, then the stated equality does not hold.

$$\frac{x{:}\tau \in \Gamma}{\Delta \mid \Gamma \vdash x \mathrel{\mathcal{R}} x : \tau} \qquad \frac{}{\Delta \mid \Gamma \vdash \langle\rangle \mathrel{\mathcal{R}} \langle\rangle : \mathbf{1}}$$

$$\frac{\Delta \mid \Gamma \vdash e_1 \mathrel{\mathcal{R}} e_1' : \tau_1 \qquad \Delta \mid \Gamma \vdash e_2 \mathrel{\mathcal{R}} e_2' : \tau_2}{\Delta \mid \Gamma \vdash \langle e_1, e_2\rangle \mathrel{\mathcal{R}} \langle e_1', e_2'\rangle : \tau_1 \times \tau_2} \qquad \frac{\Delta \mid \Gamma, x{:}\tau_1 \vdash e \mathrel{\mathcal{R}} e' : \tau_2}{\Delta \mid \Gamma \vdash \lambda x.e \mathrel{\mathcal{R}} \lambda x.e' : \tau_1 \to \tau_2}$$

$$\frac{\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau_1}{\Delta \mid \Gamma \vdash \mathtt{inl}\, e \mathrel{\mathcal{R}} \mathtt{inl}\, e' : \tau_1 + \tau_2} \qquad \frac{\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau_2}{\Delta \mid \Gamma \vdash \mathtt{inr}\, e \mathrel{\mathcal{R}} \mathtt{inr}\, e' : \tau_1 + \tau_2}$$

$$\frac{\Delta \mid \Gamma, x_1{:}\tau_1 \vdash e_1 \mathrel{\mathcal{R}} e_1' : \tau \qquad \Delta \mid \Gamma, x_2{:}\tau_2 \vdash e_2 \mathrel{\mathcal{R}} e_2' : \tau \qquad \Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau_1 + \tau_2}{\Delta \mid \Gamma \vdash \mathtt{match}(e, x_1.e_1, x_2.e_2) \mathrel{\mathcal{R}} \mathtt{match}(e', x_1.e_1', x_2.e_2') : \tau}$$

$$\frac{\Delta, \alpha \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau}{\Delta \mid \Gamma \vdash \Lambda.e \mathrel{\mathcal{R}} \Lambda.e' : \forall \alpha.\tau} \qquad \frac{\Delta \vdash \tau_1 \qquad \Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau[\tau_1/\alpha]}{\Delta \mid \Gamma \vdash (\mathsf{pack}\, e) \mathrel{\mathcal{R}} (\mathsf{pack}\, e') : \exists \alpha.\tau}$$

$$\frac{\Delta \mid \Gamma \vdash e_1 \mathrel{\mathcal{R}} e_1' : \exists \alpha.\tau_1 \qquad \Delta \vdash \tau \qquad \Delta, \alpha \mid \Gamma, x : \tau_1 \vdash e \mathrel{\mathcal{R}} e' : \tau}{\Delta \mid \Gamma \vdash (\mathtt{unpack}\ e_1\ \mathtt{as}\ x\ \mathtt{in}\ e) \mathrel{\mathcal{R}} (\mathtt{unpack}\ e_1'\ \mathtt{as}\ x\ \mathtt{in}\ e') : \tau}$$

$$\frac{\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau_1 \times \tau_2}{\Delta \mid \Gamma \vdash \mathtt{proj}_i\, e \mathrel{\mathcal{R}} \mathtt{proj}_i\, e' : \tau_i} \qquad \frac{\Delta \mid \Gamma \vdash e_1 \mathrel{\mathcal{R}} e_1' : \tau' \to \tau \qquad \Delta \mid \Gamma \vdash e_2 \mathrel{\mathcal{R}} e_2' : \tau'}{\Delta \mid \Gamma \vdash e_1\, e_2 \mathrel{\mathcal{R}} e_1'\, e_2' : \tau}$$

$$\frac{\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \mu\alpha.\tau}{\Delta \mid \Gamma \vdash \mathtt{unfold}\, e \mathrel{\mathcal{R}} \mathtt{unfold}\, e' : \tau[\mu\alpha.\tau/\alpha]} \qquad \frac{\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau[\mu\alpha.\tau/\alpha]}{\Delta \mid \Gamma \vdash \mathtt{fold}\, e \mathrel{\mathcal{R}} \mathtt{fold}\, e' : \mu\alpha.\tau}$$

$$\frac{\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \forall \alpha.\tau}{\Delta \mid \Gamma \vdash e[\,] \mathrel{\mathcal{R}} e'[\,] : \tau[\tau'/\alpha]}\ ftv(\tau') \subseteq \Delta \qquad \frac{\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \mathtt{nat}}{\Delta \mid \Gamma \vdash \mathtt{rand}\, e \mathrel{\mathcal{R}} \mathtt{rand}\, e' : \mathtt{nat}}$$

$$\frac{\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \mathtt{nat}}{\Delta \mid \Gamma \vdash \mathrm{P}\, e \mathrel{\mathcal{R}} \mathrm{P}\, e' : \mathtt{nat}} \qquad \frac{\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \mathtt{nat}}{\Delta \mid \Gamma \vdash \mathrm{S}\, e \mathrel{\mathcal{R}} \mathrm{S}\, e' : \mathtt{nat}}$$

$$\frac{\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e' : \mathtt{nat} \qquad \Delta \mid \Gamma, \vdash e_1 \mathrel{\mathcal{R}} e_1' : \tau \qquad \Delta \mid \Gamma, \vdash e_2 \mathrel{\mathcal{R}} e_2' : \tau}{\Delta \mid \Gamma \vdash \mathtt{if}_1\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mathrel{\mathcal{R}} \mathtt{if}_1\ e'\ \mathsf{then}\ e_1'\ \mathsf{else}\ e_2' : \tau}$$

Figure 2.6: Compatibility properties of type-indexed relations

$$e \oplus e =^{ctx} e \qquad e_1 \oplus e_2 =^{ctx} e_2 \oplus e_1 \qquad e \oplus \Omega \lesssim^{ctx} e$$

$$\text{if } e_1 \overset{\text{cf}}{\Longrightarrow} e_2 \text{ then } e_1 =^{ctx} e_2 \qquad \text{if } e_1 \oplus e_2 =^{ctx} e_1 \text{ then } e_1 =^{ctx} e_2$$

Figure 2.7: Basic properties of $\lesssim^{ctx}$ and $=^{ctx}$. We write Ω for any diverging term (i.e. $\mathfrak{P}^{\Downarrow}(\Omega) = 0$) and $e \oplus e'$ as syntactic sugar for $\texttt{if}_1\ \texttt{rand}\,\underline{2}\ \texttt{then}\ e\ \texttt{else}\ e'$. Note that the choice when evaluating $e \oplus e'$ is made *before* e and e' are evaluated.

Chapter 3

A Model of Countable Nondeterminism in Guarded Type Theory

This chapter is a revised version of

Aleš Bizjak, Lars Birkedal, and Marino Miculan.

A model of countable nondeterminism in guarded type theory.

In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi*, volume 8560 of *Lecture Notes in Computer Science*, pages 108–123. Springer International Publishing, 2014.

ISBN 978-3-319-08917-1

Abstract

We show how to construct a logical relation for countable nondeterminism in a guarded type theory, corresponding to the internal logic of the topos $\mathbf{Sh}(\omega_1)$ of sheaves over ω_1. In contrast to earlier work on abstract step-indexed models, we not only construct the logical relations in the guarded type theory, but also give an internal proof of the adequacy of the model with respect to standard contextual equivalence. To state and prove adequacy of the logical relation, we introduce a new propositional modality. In connection with this modality we show why it is necessary to work in the logic of $\mathbf{Sh}(\omega_1)$.

3.1 Introduction

Countable nondeterminism arises naturally when modelling properties of concurrent systems or systems with user input, etc. Still, semantic models for reasoning about *must-contextual equivalence* of higher-order programming languages with countable nondeterminism are challenging to construct [3, 10, 38, 60–63, 84]. Recently it was shown [23] how step-indexed logical relations, indexed over the first uncountable ordinal ω_1, can be used to give a simple model of a higher-order programming language $\mathbf{F}^{\mu,?}$ with recursive types and countable nondeterminism, allowing one to reason about must-contextual equivalence. Using step-indexed logical relations is arguably substantially simpler than using other models, but still involves some tedious reasoning about indices, as is characteristic of any concrete step-indexed model.

In previous work [22, 39], the guarded type theory corresponding to the internal logic of the topos $\mathbf{Sh}(\omega)$ of sheaves[1] on ω has been proved very useful for developing abstract accounts of step-indexed models indexed over ω. Such abstract accounts eliminate much of the explicit tedious reasoning about indices. We recall that the internal logic of $\mathbf{Sh}(\omega)$ can be thought of as a logic of discrete time, with time corresponding to ordinals and smaller ordinals being the future. In the application to step-indexed logical relations, the link between steps in the operational semantics and the notion of time provided by the internal logic of $\mathbf{Sh}(\omega)$ is made by defining the operational semantics using guarded recursion [22].

In this paper we show how to construct a logical relation for countable nondeterminism in a guarded type theory GTT corresponding to the internal logic of the topos $\mathbf{Sh}(\omega_1)$ of sheaves over ω_1. For space reasons we only consider the case of must-equivalence; the case for may-equivalence is similar. In contrast to earlier work on abstract step-indexed models [22, 39], we not only construct the logical relation in the guarded type theory, but also give an internal proof of the adequacy of the model with respect to must-contextual equivalence. To state and prove adequacy of the logical relation we introduce a new propositional modality $\square$: intuitively, $\square\varphi$ holds if φ holds at all times. Using this modality we give a logical explanation for why it is necessary to work in the logic of $\mathbf{Sh}(\omega_1)$: a certain logical equivalence involving $\square$ holds in the internal logic of $\mathbf{Sh}(\omega_1)$ but not in the internal logic of $\mathbf{Sh}(\omega)$ (see Lemma 3.4.7).

To model *must*-equivalence, we follow [23] and define the logical relation using *biorthogonality*. Typically, biorthogonality relies on a definition of convergence; in our case, it would be must-convergence. In an abstract account of step-indexed models, convergence would need to be defined by guarded recursion (to show the fundamental lemma). However, that is not possible in the logic of $\mathbf{Sh}(\omega_1)$. There are two ways to understand that. If one considers the natural guarded-recursive definition of convergence,[2] using Löb induction one could show that a non-terminating computation would con-

[1] Considered as sheaves on the topological space ω equipped with the Alexandrov topology.

[2] must-converge$(e) \leftrightarrow \forall e', e \rightsquigarrow e' \rightarrow \triangleright(\text{must-converge}(e'))$.

verge! Another way to understand this issue is in terms of the model. The stratified convergence predicate $\Downarrow_\beta$ from [23] is not a well-defined subobject in $\mathbf{Sh}(\omega_1)$. Intuitively, the reason is that all predicates in GTT are closed wrt. the future (smaller ordinals), but if an expression converges to a value in, say, 15 computation steps, then it does not necessarily converge to a value in 14 steps. Instead we observe that the dual of stratified must-convergence, the stratified *may-divergence*, is a subobject of $\mathbf{Sh}(\omega_1)$ and can easily be defined as a predicate in GTT using guarded recursion. Thus we use the stratified may-divergence predicate to define biorthogonality, modifying the definition accordingly.

The remainder of the paper is organised as follows. In Section 3.2 we explain the guarded type theory GTT, which we use to define the operational semantics of the higher-order programming language $\mathbf{F}^{\mu,?}$ with countable nondeterminism (Section 3.3) and to define the adequate logical relation for reasoning about contextual equivalence (Section 3.4). We include an example to demonstrate how reasoning in the resulting model avoids tedious step-indexing. Finally, in Section 3.5 we show that the guarded type theory GTT is consistent by providing a model thereof in $\mathbf{Sh}(\omega_1)$. Thus, most of the paper can be read without understanding the details of the model $\mathbf{Sh}(\omega_1)$. Most proofs have been omitted from the paper. They can be found in the appendices following the paper on page 98.

3.2 The Logic GTT

The logic GTT is the internal logic of $\mathbf{Sh}(\omega_1)$. In this section we explain some of the key features of the logic; in the subsequent development we will also use a couple of additional facts, which will be introduced as needed.

The logic is an extension of a multisorted intuitionistic higher-order logic with two modalities $\triangleright$ and $\Box$, pronounced “later” and “always” respectively. Types (aka sorts) are ranged over by X, Y; we denote the type of propositions by Ω and the function space from X to Y as Y^X. We write $\mathcal{P}(X) = \Omega^X$ for the type of the power set of X. We think of types as *variable* sets (although in the logic we will not deal with indices explicitly). There is a subset of types which we call *constant sets*; given a set a, we denote by $\Delta(a)$ the type which is constantly equal to a. Constant sets are closed under product and function space. For each type X there is a type $\blacktriangleright X$ and a function symbol $\mathrm{next}^X : X \to \blacktriangleright X$. Intuitively $\blacktriangleright X$ is “one time step later” than the type X, so we can only use it *later*, i.e. after one time step and $\mathrm{next}^X(x)$ freezes x for a time step so it is only available later.

We also single out the space of *total* types. Intuitively, these are the types whose elements at each stage have evolved from some elements from previous stages, i.e. they do not appear out of nowhere.

Definition 3.2.1. For a type X we define $\mathrm{Total}(X)$ to mean that next^X is surjective

$$\mathrm{Total}(X) \triangleq \forall x : \blacktriangleright X, \exists x' : X, \mathrm{next}^X(x') = x$$

and say that X is *total* when $\mathrm{Total}(X)$ holds. ♦

Note that for each X, $\mathrm{Total}(X)$ is a formula of the logic, but Total itself is not a predicate of the logic. Constant sets $\Delta(a)$ for an inhabited a are total. For simplicity, we do not formalise how to construct constant sets. In the following, we shall instead just state for some of the types that we use that they are constant; these facts can be shown using the model in Section 3.5.

We will adopt the usual "sequent-in-context" judgement of the form

$$\Gamma \mid \Xi \vdash \varphi$$

for saying that the formula φ is a consequence of formulas in Ξ, under the typing context Γ.

The $\triangleright$ modality on *formulas* is used to express that a formula holds only "later", that is, after a time step. More precisely, there is a function symbol $\triangleright : \Omega \to \Omega$ which we extend to formulas by composition. We require $\triangleright$ to satisfy the following properties (Γ is an arbitrary context).

1. (Monotonicity) $\Gamma \mid \varphi \vdash \triangleright\varphi$

2. (Löb induction rule) $\Gamma \mid (\triangleright\varphi \to \varphi) \vdash \varphi$

3. $\triangleright$ commutes over $\top$, $\wedge$, $\to$ and $\vee$ (but does not preserve $\bot$).

4. For all X, Y and φ we have $\Gamma, x : X \mid \exists y : Y, \triangleright\varphi(x,y) \vdash \triangleright(\exists y : Y, \varphi(x,y))$.

5. For all X, Y and φ we have $\Gamma, x : X \mid \triangleright(\forall y : Y, \varphi(x,y)) \vdash \forall y : Y, \triangleright\varphi(x,y)$. The converse entailment in the last rule holds if Y is total.

Following [22, Definition 2.8] we define a notion of contractiveness which will be used to construct unique fixed points of morphisms on total types.

Definition 3.2.2. We define the predicate Contr on Y^X as

$$\mathrm{Contr}(f) \triangleq \forall x, x' : X, \triangleright(x = x') \to f(x) = f(x')$$

and we say that f is *(internally) contractive* if $\mathrm{Contr}(f)$ holds. ♦

Intuitively, a function f is contractive if $f(x)$ *now* depends only on the value of x later, in the future. The following theorem holds in the logic.

Theorem 3.2.3 (Internal Banach's fixed point theorem)**.** *Internally, any contractive function f on a total object X has a unique fixed point. More precisely, the following formula is valid in the logic of* $\mathbf{Sh}(\omega_1)$*:*

$$\mathrm{Total}(X) \to \forall f : X^X, \mathrm{Contr}(f) \to \exists! x : X, f(x) = x.$$

◊

We will use Theorem 3.2.3 in Section 3.4 on a function of type $\mathcal{P}(X) \to \mathcal{P}(X)$ for a constant set X. We thus additionally assume that $\mathrm{Total}(\mathcal{P}(X))$ holds for any constant set X.

The $\Box$ modality is used to express that a formula holds for all time steps. It is thus analogous to the $\Box$ modality in temporal logic. It is defined as the *right* adjoint to the $\neg\neg$-closure operation on formulas and behaves as an interior operator. More precisely, for a formula φ in context Γ, $\Box\varphi$ is another formula in context Γ. In contrast to the $\triangleright$ modality, $\Box$ on formulas does not arise from a function on Ω and consequently does not commute with substitution, i.e., in general $(\Box\varphi)[t/x]$ is not equivalent to $\Box(\varphi[t/x])$, although $(\Box\varphi)[t/x]$ always implies $\Box(\varphi[t/x])$ which is useful for instantiating universally quantified assumptions. Thus, to be precise, we would have to annotate the $\Box$ with the context in which it is used. However, restricting to contexts consisting of constant types, $\Box$ does commute with substitution and since we will only use it in such contexts we will omit explicit contexts.

The basic rules for the $\Box$ modality are the following. In particular, note the first rule which characterises $\Box$ as the right adjoint to the $\neg\neg$-closure.

$$\begin{array}{c} \Gamma \mid \neg\neg\varphi \vdash \psi \\ \hline\hline \Gamma \mid \varphi \vdash \Box\psi \end{array} \qquad \frac{\Gamma \mid \varphi \vdash \psi}{\Gamma \mid \Box\varphi \vdash \Box\psi} \qquad \frac{}{\Gamma \mid \Box\varphi \vdash \varphi}$$

$$\frac{}{\Gamma \mid \Box\varphi \vdash \Box\Box\varphi} \qquad \frac{}{\Gamma \mid \neg\neg(\Box\varphi) \vdash \Box\varphi} \qquad \frac{}{\Gamma \mid \neg\neg\varphi \vdash \Box(\neg\neg\varphi)}$$

Note that some of the rules can be derived from others. A simple consequence of the rules is that $\neg\neg\varphi \leftrightarrow \Box(\neg\neg\varphi)$ and $\neg\neg(\Box\varphi) \leftrightarrow \Box\varphi$. Thus one way to understand $\Box\varphi$ is as the largest predicate that implies φ and is $\neg\neg$-closed.

Proposition 3.2.4. *Using the rules for $\Box$ stated above we can prove the following sequents in the logic.*

$$\begin{aligned}
&\Gamma \mid \emptyset \vdash \Box\top \leftrightarrow \top \\
&\Gamma \mid \emptyset \vdash \Box\bot \leftrightarrow \bot \\
&\Gamma \mid \emptyset \vdash \Box(\varphi \wedge \psi) \leftrightarrow \Box\varphi \wedge \Box\psi \\
&\Gamma \mid \emptyset \vdash \Box(\forall x : X, \varphi) \leftrightarrow \forall x : X, \Box\varphi \\
&\Gamma \mid \emptyset \vdash \Box(\varphi \to \psi) \to \Box\varphi \to \Box\psi
\end{aligned}$$

◊

$$\begin{aligned}
\tau &::= \alpha \mid \mathbf{1} \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \to \tau_2 \mid \mu\alpha.\tau \mid \forall\alpha.\tau \mid \exists\alpha.\tau \\
e &::= x \mid \langle\rangle \mid \langle e_1, e_2\rangle \mid \texttt{inl}\ e \mid \texttt{inr}\ e \mid \lambda x.e \mid \Lambda.e \mid \texttt{pack}\,e \mid \texttt{unfold}\,e \mid \texttt{fold}\,e \\
&\qquad \mid ? \mid \texttt{proj}_i\, e \mid e_1\, e_2 \mid \texttt{case}(e, x_1.e_1, x_2.e_2) \mid e[] \mid \texttt{unpack}\ e_1\ \texttt{as}\ x\ \texttt{in}\ e_2 \\
E &::= - \mid \langle E, e\rangle \mid \langle v, E\rangle \mid \texttt{inl}\ E \mid \texttt{inr}\ E \mid \texttt{pack}\,E \mid \texttt{proj}_i\, E \mid E\, e \mid v\, E \mid E[] \\
&\qquad \mid \texttt{case}(E, x_1.e_1, x_2.e_2) \mid \texttt{unpack}\ E\ \texttt{as}\ x\ \texttt{in}\ e \mid \texttt{unfold}\,E \mid \texttt{fold}\,E
\end{aligned}$$

Figure 3.1: Syntax of $\mathbf{F}^{\mu,?}$: types τ, terms e and evaluation contexts E. $\texttt{inl}\ e$ and $\texttt{inr}\,e$ introduce terms of sum type. $\texttt{case}(e, x_1.e_1, x_2.e_2)$ is the pattern matching construct that eliminates a term e of the sum type with the left branch being e_1 and right branch e_2. $\texttt{pack}\,e$ and $\texttt{unpack}\ e_1\ \texttt{as}\ x\ \texttt{in}\ e_2$ introduce and eliminate terms of existential types and $\Lambda.e$ and $e[]$ introduce and eliminate terms of universal types.

A useful derived introduction rule for the $\square$ modality is the well-known $\square$-introduction rule for S4. It states that if we can prove φ using only $\square$'ed facts, then we can also conclude $\square\,\varphi$. Formally:

$$\frac{\Gamma \mid \Xi \vdash \varphi}{\Gamma \mid \Xi \vdash \square\,\varphi}\ \Xi = \square\,\varphi_1, \square\,\varphi_2, \ldots, \square\,\varphi_n$$

3.3 The Language $\mathbf{F}^{\mu,?}$

In this section we introduce $\mathbf{F}^{\mu,?}$, a call-by-value functional language akin to System F, i.e., with impredicative polymorphism, existential and general recursive types, extended with a countable choice expression ?. We work informally in the logic outlined above except where explicitly stated.

Syntax We assume disjoint, countably infinite sets of *type variables*, ranged over by α, and *term variables*, ranged over by x. The syntax of types, terms and evaluation contexts is defined in Figure 3.1. Values v and contexts (terms with a hole) C can be defined in the usual way. The free type variables in a type $ftv(\tau)$ and free term variables in a term $fv(e)$, are defined in the usual way. The notation $\sigma[\vec{\tau}/\vec{\alpha}]$ denotes the simultaneous capture-avoiding substitution of types $\vec{\tau}$ for the free type variables $\vec{\alpha}$ in the type σ; similarly, $e[\vec{v}/\vec{x}]$ denotes simultaneous capture-avoiding substitution of values $\vec{v}$ for the free term variables $\vec{x}$ in e. We define the type of natural numbers as $\texttt{nat} = \mu\alpha.1 + \alpha$ and the corresponding numerals as $\underline{0} = \texttt{fold}(\texttt{inl}\ \langle\rangle)$ and $\underline{n+1} = \texttt{fold}(\texttt{inr}\ \underline{n})$ by induction on n.

The judgement $\Delta \vdash \tau$ expresses $ftv(\tau) \subseteq \Delta$. The typing judgement $\Delta \mid \Gamma \vdash e : \tau$ expresses that e has type τ in type variable context Δ and term

$$\begin{array}{rlr}
\mathtt{proj}_i\,\langle v_1, v_2\rangle \longmapsto v_i & & \mathtt{unfold}(\mathtt{fold}\,v) \longmapsto v \\
(\lambda x.e)\,v \longmapsto e[v/x] & & \mathtt{unpack}\;(\mathtt{pack}\,v)\;\mathtt{as}\;x\;\mathtt{in}\;e \longmapsto e[v/x] \\
(\Lambda.e)[] \longmapsto e & & \mathtt{case}(\mathtt{inl}\,v, x_1.e_1, x_2.e_2) \longmapsto e_1[v/x_1] \\
? \longmapsto \underline{n} \quad (n \in \mathbb{N}) & & \mathtt{case}(\mathtt{inr}\,v, x_1.e_1, x_2.e_2) \longmapsto e_2[v/x_2]
\end{array}$$

$$E[e] \leadsto E[e'] \qquad \text{if } e \longmapsto e'$$

Figure 3.2: Operational semantics of $\mathbf{F}^{\mu,?}$: basic reductions $\longmapsto$ and one step reduction $\leadsto$.

variable context Γ. Typing rules are the same as for system F with recursive types, apart from the typing of the ?, which has type `nat` in any well-formed context.

We write **Type** for the set of closed types τ, i.e. types τ satisfying $ftv(\tau) = \emptyset$. We write $\mathbf{Val}(\tau)$ and $\mathbf{Tm}(\tau)$ for the sets of closed values and terms of type τ, respectively. $\mathbf{Stk}(\tau)$ denotes the set of evaluation contexts E with the hole of type τ. The typing of evaluation contexts can be defined as in [23] by an inductively defined relation. We write **Val** and **Tm** for the set of all closed values and closed terms, respectively, and **Stk** for the set of all evaluation contexts.

Using the model in Section 3.5, we can show that the types of terms, values, evaluation contexts and contexts are constant sets. We use this fact in the proof of adequacy in Section 3.4.

Operational semantics The operational semantics of $\mathbf{F}^{\mu,?}$ is given in Figure 3.2 by a one-step reduction relation $e \leadsto e'$. The rules are standard apart from the rule for ? which states that the countable choice expression ? evaluates nondeterministically to any numeral $\underline{n}$ $(n \in \mathbb{N})$. We extend basic reduction $\longmapsto$ to the single step reduction relation $\leadsto$ using evaluation contexts E.

To define the logical relation we need further restricted reduction relations. These will allow us to ignore most reductions in the definition of the logical relation, except the ones needed to prove the fundamental property (Corollary 3.4.5).

Let $\leadsto^*$ be the reflexive transitive closure of $\leadsto$. Following [23] we call *unfold-fold* reductions those of the form $\mathtt{unfold}(\mathtt{fold}\,v) \longmapsto v$, and *choice* reductions those of the form $? \longmapsto \underline{n} \quad (n \in \mathbb{N})$. Choice reductions are important because these are the only ones that do not preserve equivalence. We define

- $e \overset{p}{\leadsto} e'$ if $e \leadsto^* e'$ and *none* of the reductions is a choice reduction;
- $e \overset{0}{\leadsto} e'$ if $e \leadsto^* e'$ and *none* of the reductions is an unfold-fold reduction;

- $e \overset{1}{\rightsquigarrow} e'$ if $e \rightsquigarrow^{*} e'$ and *exactly one* of the reductions is an unfold-fold reduction;
- $e \overset{p,0}{\rightsquigarrow} e'$ if $e \overset{p}{\rightsquigarrow} e'$ and $e \overset{0}{\rightsquigarrow} e'$;
- $e \overset{p,1}{\rightsquigarrow} e'$ if $e \overset{p}{\rightsquigarrow} e'$ and $e \overset{1}{\rightsquigarrow} e'$.

The $\overset{1}{\rightsquigarrow}$ reduction relation will be used in the stratified definition of divergence and the other reduction relations will be used to state additional properties of the logical relation in Lemma 3.3.1. Note that although some of the relations are described informally using negation they can be described constructively in a positive way. For instance, $\overset{p}{\rightsquigarrow}$ can be defined in the same way as the $\rightsquigarrow^{*}$ but using a subset of the one step relation $\rightsquigarrow$.

Divergence relations We define the logical relation using biorthogonality. As we explained in the introduction we use two *may-divergence* predicates, which are, informally, the negations of the two *must-convergence* relations from [23]. Thus we define, in the logic, the *stratified may-divergence predicate* $\underline{\uparrow}$ as the unique fixed point of $\Psi : \mathcal{P}(\mathbf{Tm}) \to \mathcal{P}(\mathbf{Tm})$ given as

$$\Psi(A) = \left\{ e : \mathbf{Tm} \;\middle|\; \exists e' : \mathbf{Tm}, e \overset{1}{\rightsquigarrow} e' \wedge \triangleright(e' \in A) \right\}.$$

Ψ is internally contractive and since $\mathbf{Tm}$ is a constant set $\mathcal{P}(\mathbf{Tm})$ is total. By Theorem 3.2.3, Ψ has a unique fixed point.

We also define the non-stratified may-divergence predicate $\uparrow$ as the *greatest* fixed-point of $\Phi : \mathcal{P}(\mathbf{Tm}) \to \mathcal{P}(\mathbf{Tm})$ given as

$$\Phi(A) = \left\{ e : \mathbf{Tm} \;\middle|\; \exists e' : \mathbf{Tm}, e \rightsquigarrow e' \wedge e' \in A \right\}.$$

Since Φ is monotone and $\mathcal{P}(\mathbf{Tm})$ is a complete lattice, the greatest fixed point exists by Knaster-Tarski's fixed-point theorem, which holds in our logic.[3] Observe that Ψ is almost the same as $\Phi \circ \triangleright$, apart from using a different reduction relation. We write $e\uparrow$ and $e\underline{\uparrow}$ for $e \in \uparrow$ and $e \in \underline{\uparrow}$, respectively.

The predicates $\underline{\uparrow}$ and $\uparrow$ are closed under some, but not all, reductions.

Lemma 3.3.1. *Let $e, e' : \mathbf{Tm}$. The following properties hold in the logic GTT.*

$$\begin{array}{ll} \textit{if } e \overset{p}{\rightsquigarrow} e' \textit{ then } e\uparrow \leftrightarrow e'\uparrow & \qquad \textit{if } e \overset{p,0}{\rightsquigarrow} e' \textit{ then } e\underline{\uparrow} \leftrightarrow e'\underline{\uparrow} \\ \textit{if } e \overset{0}{\rightsquigarrow} e' \textit{ then } e'\underline{\uparrow} \to e\underline{\uparrow} & \qquad \textit{if } e \overset{1}{\rightsquigarrow} e' \textit{ then } \triangleright(e'\underline{\uparrow}) \to e\underline{\uparrow} \end{array}$$

◊

[3] Knaster-Tarski's fixed point theorem holds in the internal language of any topos.

Must-contextual approximation Contexts can be typed as second-order terms, by means of a typing judgement of the form

$$C : (\Delta \mid \Gamma \Rrightarrow \tau) \hookrightarrow (\Delta' \mid \Gamma' \Rrightarrow \sigma),$$

stating that whenever $\Delta \mid \Gamma \vdash e : \tau$ holds, $\Delta' \mid \Gamma' \vdash C[e] : \sigma$ also holds. The typing of contexts can be defined as an inductive relation defined by suitable typing rules, which we omit here due to lack of space; see [6]. We write

$$C : (\Delta \mid \Gamma \Rrightarrow \tau)$$

to mean there exists a type σ, such that

$$C : (\Delta \mid \Gamma \Rrightarrow \tau) \hookrightarrow (\varnothing \mid \varnothing \Rrightarrow \sigma)$$

holds.

We define *contextual must-approximation* using the may-divergence predicate. This is in contrast with the definition in [23] which uses the must-convergence predicate. However externally, in the model, the two definitions coincide.

Definition 3.3.2 (Must-contextual approximation)**.** In GTT, we define must-contextual approximation $\Delta \mid \Gamma \vdash e_1 \lesssim^{ctx}_{\Downarrow} e_2 : \tau$ as

$$\Delta \mid \Gamma \vdash e_1 : \tau \wedge \Delta \mid \Gamma \vdash e_2 : \tau \ \wedge \forall C, (C : (\Delta \mid \Gamma \Rrightarrow \tau)) \wedge C[e_2]\uparrow \to C[e_1]\uparrow.$$

♦

Note the order in the implication: if $C[e_2]$ may-diverges then $C[e_1]$ may-diverges. This is the contrapositive of the definition in [23] which states that if $C[e_1]$ must-converges then $C[e_2]$ must-converges. Must-contextual approximation defined explicitly using contexts can be shown to be the largest compatible adequate and transitive relation, so it coincides with contextual approximation in [23].

Compatible means closed under the rules in Figure 3.8 on page 128.

3.4 Logical Relation

In this section we give an abstract account of the concrete step-indexed logical relation from [23] by defining a logical relation interpretation of types in GTT. The result is a simpler model without a proliferation of step-indices, as we will demonstrate in the example at the end of the section.

Relational interpretation of types Let $\mathbf{Type}(\Delta) = \{\tau \mid \Delta \vdash \tau\}$ be the set of types well-formed in context Δ. Given $\tau, \tau' \in \mathbf{Type}$ let

$$\begin{aligned} \mathbf{VRel}(\tau,\tau') &= \mathcal{P}(\mathbf{Val}(\tau) \times \mathbf{Val}(\tau')) \\ \mathbf{TRel}(\tau,\tau') &= \mathcal{P}(\mathbf{Tm}(\tau) \times \mathbf{Tm}(\tau')) \\ \mathbf{SRel}(\tau,\tau') &= \mathcal{P}(\mathbf{Stk}(\tau) \times \mathbf{Stk}(\tau')). \end{aligned}$$

We implicitly use the inclusion $\mathbf{VRel}(\tau,\tau') \subseteq \mathbf{TRel}(\tau,\tau')$. For a type variable context Δ, we define $\mathbf{VRel}(\Delta)$ to be

$$\{(\varphi_1,\varphi_2,\varphi_r) \mid \varphi_1,\varphi_2 : \Delta \to \mathbf{Type}, \forall \alpha \in \Delta, \varphi_r(\alpha) \in \mathbf{VRel}(\varphi_1(\alpha),\varphi_2(\alpha))\}$$

where the first two components give syntactic types for the left and right hand sides of the relation and the third component is a relation between those types. The interpretation of types, $[\![\cdot \vdash \cdot]\!]$, is shown in Figure 3.3. The definition is by induction on the judgement $\Delta \vdash \tau$. Given a judgement $\Delta \vdash \tau$, and $\varphi \in \mathbf{VRel}(\Delta)$, we have $[\![\Delta \vdash \tau]\!](\varphi) \in \mathbf{VRel}(\varphi_1(\tau),\varphi_2(\tau))$ where φ_1 and φ_2 are the first two components of φ, and $\varphi_i(\tau)$ denotes substitution of types in φ_i for free type variables in τ. Since we are working in the logic GTT, the interpretations of all type constructions are simple and intuitive. For instance, functions are related when they map related values to related results, two values of universal type are related if they respect all value relations. In particular, there are no admissibility requirements on the relations, nor any step-indexing — but just a use of $\triangleright$ in the interpretation of recursive types, to make it well-defined as a consequence of Theorem 3.2.3, using that the type $\mathcal{P}(\mathbf{Tm} \times \mathbf{Tm})$ is total.

The definition of $\top\top$-closure is where we connect operational semantics and the $\triangleright$ modality, using the stratified may-divergence predicate $\underline{\uparrow}$. $\top\top$-closed relations are closed under some reductions. More precisely, the following lemma holds.

Lemma 3.4.1. *Let $\tau,\tau' : \mathbf{Type}$ and $r \in \mathbf{VRel}(\tau,\tau')$.*

- *If $e \overset{p,0}{\rightsquigarrow} e_1$ and $e' \overset{p}{\rightsquigarrow} e_1'$ then $(e,e') \in r^{\top\top} \leftrightarrow (e_1,e_1') \in r^{\top\top}$.*
- *If $e \overset{1}{\rightsquigarrow} e_1$ then for all $e' : \mathbf{Tm}$, if $\triangleright((e_1,e') \in r^{\top\top})$ then $(e,e') \in r^{\top\top}$.*

◊

We use this fact extensively in the proofs of the fundamental property and example equivalences.

In order to define logical relations, we need first to extend the interpretation of types to the interpretation of contexts (note that in particular, related substitutions map into well-typed values):

$$[\![\Delta \vdash \Gamma]\!](\varphi) = \{(\gamma,\gamma') \mid \gamma,\gamma' : \mathbf{Val}^{\mathrm{dom}(\Gamma)}, \\ \forall x \in \mathrm{dom}(\Gamma), (\gamma(x),\gamma'(x)) \in [\![\Delta \vdash \Gamma(x)]\!](\varphi)\}$$

The logical relation and its fundamental property We define the logical relation on open terms by reducing it to relations on closed terms by substitution.

$$
\begin{aligned}
\llbracket \Delta \vdash \alpha \rrbracket(\varphi) &= \varphi_r(\alpha) \\
\llbracket \Delta \vdash \mathbf{1} \rrbracket(\varphi) &= \mathbf{Id_1} \\
\llbracket \Delta \vdash \tau_1 \times \tau_2 \rrbracket(\varphi) &= \left\{ (\langle v, u\rangle, \langle v', u'\rangle) \;\middle|\; \begin{array}{c} (v, v') \in \llbracket \Delta \vdash \tau_1 \rrbracket(\varphi) \\ (u, u') \in \llbracket \Delta \vdash \tau_2 \rrbracket(\varphi) \end{array} \right\} \\
\llbracket \Delta \vdash \tau_1 + \tau_2 \rrbracket(\varphi) &= \{(\mathtt{inl}\, v, \mathtt{inl}\, v') \mid (v, v') \in \llbracket \Delta \vdash \tau_1 \rrbracket(\varphi)\} \cup \\
&\qquad \{(\mathtt{inr}\, u, \mathtt{inr}\, u') \mid (u, u') \in \llbracket \Delta \vdash \tau_2 \rrbracket(\varphi)\} \\
\llbracket \Delta \vdash \tau_1 \to \tau_2 \rrbracket(\varphi) &= \{(\lambda x.e, \lambda y.e') \mid \forall (v, v') \in \llbracket \Delta \vdash \tau_1 \rrbracket(\varphi), \\
&\qquad (e[v/x], e'[v'/y]) \in \llbracket \Delta \vdash \tau_2 \rrbracket(\varphi)^{\top\top}\} \\
\llbracket \Delta \vdash \forall \alpha.\tau \rrbracket(\varphi) &= \{(\Lambda.e, \Lambda.e') \mid \forall \sigma, \sigma' \in \mathbf{Type}, \forall s \in \mathbf{VRel}(\sigma, \sigma'), \\
&\qquad (e, e') \in \llbracket \Delta, \alpha \vdash \tau \rrbracket(\varphi[\alpha \mapsto (\sigma, \sigma', s)])^{\top\top}\} \\
\llbracket \Delta \vdash \exists \alpha.\tau \rrbracket(\varphi) &= \{(\mathtt{pack}\, v, \mathtt{pack}\, v') \mid \exists \sigma, \sigma' \in \mathbf{Type}, \exists s \in \mathbf{VRel}(\sigma, \sigma'), \\
&\qquad (v, v') \in \llbracket \Delta, \alpha \vdash \tau \rrbracket(\varphi[\alpha \mapsto (\sigma, \sigma', s)])\} \\
\llbracket \Delta \vdash \mu\alpha.\tau \rrbracket(\varphi) &= \mathtt{fix}\left(\lambda s. \left\{ \begin{array}{l} (\mathtt{fold}\, v, \mathtt{fold}\, v') \mid \\ \qquad\qquad \triangleright((v, v') \in \llbracket \Delta, \alpha \vdash \tau \rrbracket(\varphi[\alpha \mapsto s])) \end{array} \right\}\right)
\end{aligned}
$$

where the

$$\cdot^{\top\top} : \mathbf{VRel}(\tau, \tau') \to \mathbf{TRel}(\tau, \tau')$$

is defined with the help of

$$\cdot^{\top} : \mathbf{VRel}(\tau, \tau') \to \mathbf{SRel}(\tau, \tau')$$

as follows

$$
\begin{aligned}
r^{\top} &= \{(E, E') \mid \forall (v, v') \in r, E'[v']\uparrow \to E[v]\mathrel{\underline{\uparrow}}\} \\
r^{\top\top} &= \{(e, e') \mid \forall (E, E') \in r^{\top}, E'[e']\uparrow \to E[e]\mathrel{\underline{\uparrow}}\}.
\end{aligned}
$$

Figure 3.3: Interpretation of types. All the relations are on *typeable* terms and contexts.

Definition 3.4.2 (Logical relation). $\Delta \mid \Gamma \vdash e_1 \lesssim^{log}_{\Downarrow} e_2 : \tau$ if

$$\forall \varphi \in \mathbf{VRel}(\Delta), \forall (\gamma, \gamma') \in \llbracket \Delta \vdash \Gamma \rrbracket(\varphi), (e_1\gamma, e_2\gamma') \in \llbracket \Delta \vdash \tau \rrbracket(\varphi)^{\top\top}.$$

♦

To prove the fundamental property of logical relations and connect the logical relation to contextual-must approximation we start with some simple properties relating evaluation contexts and relations. All the lemmas are essentially of the same form: given two related evaluation contexts at a suitable type, the contexts extended with an elimination form are also related at a suitable type. We only state the case for `unfold`, since it shows the interplay between unfold-fold reductions and the stratified may divergence predicate.

Lemma 3.4.3. *If*

$$(E, E') \in \llbracket \Delta \vdash \tau[\mu\alpha.\tau/\alpha] \rrbracket (\varphi)^{\top}$$

then

$$(E \circ (\texttt{unfold}\,[\,]), E' \circ (\texttt{unfold}\,[\,])) \in \llbracket \Delta \vdash \mu\alpha.\tau \rrbracket (\varphi)^{\top}.$$

◊

Proof. Given $(\texttt{fold}\, v, \texttt{fold}\, v') \in \llbracket \Delta \vdash \mu\alpha.\tau \rrbracket (\varphi)$ suppose $E'[\texttt{unfold}(\texttt{fold}\, v')]\uparrow$. By Lemma 3.3.1 we have $E'[v']\uparrow$ and so $\triangleright(E'[v']\uparrow)$. By definition of interpretation of recursive types we have $\triangleright((v, v') \in \llbracket \Delta \vdash \tau[\mu\alpha.\tau/\alpha] \rrbracket (\varphi))$. Thus $\triangleright(E[v]\underline{\uparrow})$ and so by Lemma 3.3.1 we have $E[\texttt{unfold}(\texttt{fold}\, v)]\underline{\uparrow}$. $\mathfrak{QED}$

Note that the proof would not work, were we to use the $\uparrow$ relation in place of $\underline{\uparrow}$ in the definition of the $\top\top$ closure since the last implication would not hold.

Proposition 3.4.4. *The logical approximation relation is compatible with the typing rules (see also Proposition 3.B.21).* ◊

Proof. We only give two cases, to show how to use the context extension lemmas.
Elimination of recursive types: we need to show

$$\frac{\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e' : \mu\alpha.\tau}{\Delta \mid \Gamma \vdash \texttt{unfold}\, e \lesssim_{\Downarrow}^{log} \texttt{unfold}\, e' : \tau[\mu\alpha.\tau/\alpha]}.$$

So take $\varphi \in \mathbf{VRel}(\Delta)$ and $(\gamma, \gamma') \in \llbracket \Delta \vdash \Gamma \rrbracket (\varphi)$. Let $f = e\gamma$ and $f' = e'\gamma'$. We have to show $(\texttt{unfold}\, f, \texttt{unfold}\, f') \in \llbracket \Delta \vdash \tau[\mu\alpha.\tau/\alpha] \rrbracket (\varphi)^{\top\top}$. So take

$$(E, E') \in \llbracket \Delta \vdash \tau[\mu\alpha.\tau/\alpha] \rrbracket (\varphi)^{\top}.$$

By assumption $(f, f') \in \llbracket \Delta \vdash \mu\alpha.\tau \rrbracket (\varphi)^{\top\top}$ so it suffices to show

$$(E \circ (\texttt{unfold}\,[\,]), E' \circ (\texttt{unfold}\,[\,])) \in \llbracket \Delta \vdash \mu\alpha.\tau \rrbracket (\varphi)^{\top}$$

and this is exactly the content of Lemma 3.4.3.
The ? expression: we need to show $\Delta \mid \Gamma \vdash\, ? \lesssim_{\Downarrow}^{log}\, ? : \texttt{nat}$. It is easy to see by induction that for all $n \in \mathbb{N}$, $(\underline{n}, \underline{n}) \in \llbracket \vdash \texttt{nat} \rrbracket$. So take $(E, E') \in \llbracket \vdash \texttt{nat} \rrbracket^{\top}$ and assume $E'[?]\uparrow$. By definition of the $\uparrow$ relation there exists an e', such that $? \rightsquigarrow e'$ and $E'[e']\uparrow$. Inspecting the operational semantics we see that $e' = \underline{n}$ for some $n \in \mathbb{N}$. This implies $E[\underline{n}]\underline{\uparrow}$ and so by Lemma 3.3.1 we have $E[?]\underline{\uparrow}$. $\mathfrak{QED}$

Corollary 3.4.5 (Fundamental property of logical relations)**.** *If* $\Delta \mid \Gamma \vdash e : \tau$ *then* $\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e : \tau$ $\Diamond$

Proof. The proof is by induction on the typing derivation $\Delta \mid \Gamma \vdash e : \tau$ using Proposition 3.4.4. $\mathfrak{QED}$

We need the next corollary to relate the logical approximation relation to must-contextual approximation.

Corollary 3.4.6. *For any expressions e, e' and context C, if*

$$\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e' : \tau$$

and

$$C : (\Delta \mid \Gamma \Rrightarrow \tau) \looparrowright (\Delta' \mid \Gamma' \Rrightarrow \sigma)$$

then

$$\Delta' \mid \Gamma' \vdash C[e] \lesssim_{\Downarrow}^{log} C[e'] : \tau'.$$

$\Diamond$

Proof. By induction on the judgement

$$C : (\Delta \mid \Gamma \Rrightarrow \tau) \looparrowright (\Delta' \mid \Gamma' \Rrightarrow \sigma)$$

using Proposition 3.4.4. $\mathfrak{QED}$

Adequacy We now wish to show soundness of the logical relation with respect to must-contextual approximation. However, the implication

$$\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e' : \tau \to \Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{ctx} e' : \tau$$

does not hold, due to the different divergence relations used in the definition of the logical relation. To see precisely where the proof fails, let us attempt it. Let $\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e' : \tau$ and take a well-typed closing context C with result type σ. Then by Corollary 3.4.6, $\varnothing \mid \varnothing \vdash C[e] \lesssim_{\Downarrow}^{log} C[e'] : \sigma$. Unfolding the definition of the logical relation we get $(C[e], C[e']) \in [\![\varnothing \vdash \sigma]\!]^{\top\top}$. It is easy to see that $(-,-) \in [\![\varnothing \vdash \sigma]\!]^{\top}$ and so we get by definition of $\top\top$ that $C[e']\uparrow \to C[e]\underline{\uparrow}$. However the definition of contextual equivalence requires the implication $C[e']\uparrow \to C[e]\uparrow$, which is not a consequence of the previous one.

The gist of the problem is that $\uparrow$ defines a time-independent predicate, whereas $\underline{\uparrow}$ is time-dependent, since it is defined by guarded recursion. However, in the model in Section 3.5, we can show the validity of a formula expressing a connection between $\uparrow$ and $\underline{\uparrow}$:

Lemma 3.4.7. $e : \mathbf{Tm} \mid \varnothing \vdash \square(e{\mapsup}) \to e{\uparrow}$ *holds in the logic GTT.* ◊

Thus we additionally assume this principle in our logic. Note that this lemma is *not* valid in the logic of the topos of trees [22] and this is the reason we must work in the logic of $\mathbf{Sh}\,(\omega_1)$. We sketch a proof of the lemma at the end of Section 3.5 which shows the role of □ and why the lemma does not hold in the topos of trees. Using Lemma 3.4.7 we are led to the following corrected statement of adequacy using the □ modality.[4]

Theorem 3.4.8 (Adequacy). *If* e *and* e' *are of type* τ *in context* $\Delta \mid \Gamma$ *then* $\square(\Delta \mid \Gamma \vdash e \lesssim^{log}_{\Downarrow} e' : \tau)$ *implies* $\Delta \mid \Gamma \vdash e \lesssim^{ctx}_{\Downarrow} e' : \tau$. ◊

To prove this theorem we first observe that all the lemmas used in the proof of Corollary 3.4.6 are proved in constant contexts, using only other constant facts. Hence, Corollary 3.4.6 can be strengthened, yielding the following restatement.

Proposition 3.4.9.

$$\square\left[\begin{array}{c} \forall \Delta,\Delta',\Gamma,\Gamma',\tau,\sigma,C,e,e',C : (\Delta \mid \Gamma \Rrightarrow \tau) \hookrightarrow (\Delta' \mid \Gamma' \Rrightarrow \sigma) \\ \to \Delta \mid \Gamma \vdash e \lesssim^{log}_{\Downarrow} e' : \tau \to \Delta' \mid \Gamma' \vdash C[e] \lesssim^{log}_{\Downarrow} C[e'] : \tau' \end{array}\right].$$

◊

Note that all the explicit universal quantification in the proposition is over constant types. One additional ingredient we need to complete the proof is the fact that $\uparrow$ is $\neg\neg$-closed, i.e. $e{\uparrow} \leftrightarrow \neg\neg(e{\uparrow})$. We can show this in the logic using the fact that $\uparrow$ is the greatest post-fixed point by showing that $\neg\neg{\uparrow}$ is another one. This fact further means that $\square(e{\uparrow}) \leftrightarrow (e{\uparrow})$ (using the adjoint rule relating $\neg\neg$ and □ in Section 3.2). We are now ready to proceed with the proof of Theorem 3.4.8.

Theorem 3.4.8. Continuing the proof we started above we get, using Proposition 3.2.4, that $\square(C[e']{\uparrow} \to C[e]{\mapsup})$ and thus also $\square(C[e']{\uparrow}) \to \square(C[e]{\mapsup})$. Moreover, $\square(C[e']{\uparrow}) \leftrightarrow C[e']{\uparrow}$ and, by Lemma 3.4.7, $\square(C[e]{\mapsup}) \to C[e]{\uparrow}$. We thus conclude $C[e']{\uparrow} \to C[e]{\uparrow}$, as required. $\mathfrak{QED}$

Thus, if we can prove that e and e' are logically related relying only on constant facts we can use this theorem to conclude that e must-contextually approximates e'. In particular, the fundamental property (Corollary 3.4.5) can be strengthened to a “boxed” statement.

[4] Readers who are familiar with concrete step-indexed models will note that the □ modality captures the universal quantification over all steps used in the the definition of concrete step-indexed logical relations.

Completeness As in [23] we also get completeness with respect to contextual approximation. The proof proceeds as in [23] via the notion of CIU-approximation [23, 67]. This property relies on the fact that we have built the logical relation using biorthogonality and using typeable realisers.

Theorem 3.4.10. *For any* Δ, Γ, e, e' *and* τ,

$$\Delta \mid \Gamma \vdash e \lesssim^{CIU}_{\Downarrow} e' : \tau \leftrightarrow \Delta \mid \Gamma \vdash e \lesssim^{ctx}_{\Downarrow} e' : \tau \leftrightarrow \Box(\Delta \mid \Gamma \vdash e \lesssim^{log}_{\Downarrow} e' : \tau)$$

◊

Applications

We can now use the logical relation to prove contextual equivalences. In Section 3.B on page 135 we provide internal proofs of all the examples done in the concrete step-indexed model in [23]; these proofs are simpler than the ones in [23]. In this section we include just one example, the proof of syntactic minimal invariance for *must*-equivalence. Remarkably, the proof below is just as simple as the proof of the minimal invariance property in the abstract account of a step-indexed model for the *deterministic* language $\mathbf{F}^{\mu}$ [39].

Let

$$\mathtt{fix} : \forall \alpha, \beta.((\alpha \to \beta) \to (\alpha \to \beta)) \to (\alpha \to \beta)$$

be the term

$$\Lambda.\Lambda.\lambda f.\delta_f(\mathtt{fold}\,\delta_f)$$

where δ_f is the term

$$\lambda y.\mathtt{let}\ y' = \mathtt{unfold}\,y\ \mathtt{in}\ f\,(\lambda x.y'\,y\,x).$$

Consider the type $\tau = \mu\alpha.\mathtt{nat} + \alpha \to \alpha$. Let $\mathrm{id} = \lambda x.x$ and consider the term

$$f \;\equiv\; \lambda h, x.\mathtt{case}(\mathtt{unfold}\,x, y.\mathtt{fold}(\mathtt{inl}\,y), g.\mathtt{fold}(\mathtt{inr}\,\lambda y.h(g(h\,y)))).$$

We show that $\mathtt{fix}[][]f \lesssim^{log}_{\Downarrow} \mathrm{id} : \tau \to \tau$. The other direction is essentially the same. Since we prove this in the context of constant facts we can use Theorem 3.4.10 to conclude that the terms are contextually equivalent.

We show by Löb induction that $(\mathtt{fix}[][]f, \mathrm{id}) \in [\![\tau \to \tau]\!]^{\top\top}$. It is easy to see that $\mathtt{fix}[][]f \overset{p,1}{\rightsquigarrow} \lambda x.\mathtt{case}(\mathtt{unfold}\,x, y.\mathtt{fold}(\mathtt{inl}\,y), g.\mathtt{fold}(\mathtt{inr}\,\lambda y.h(g(h\,y))))$ where $h = \lambda x.\delta_f(\mathtt{fold}\,\delta_f)\,x$. Let

$$\varphi = \lambda x.\mathtt{case}(\mathtt{unfold}\,x, y.\mathtt{fold}(\mathtt{inl}\,y), g.\mathtt{fold}(\mathtt{inr}\,\lambda y.h(g(h\,y)))).$$

We now directly show $(\varphi, \mathrm{id}) \in [\![\tau \to \tau]\!]$ which suffices by Lemma 3.4.1.

Let us take $(u, u') \in [\![\tau]\!]$. By the definition of the interpretation of recursive and sum types there are two cases:

- $u = \mathtt{fold}(\mathtt{inl}\,\underline{n})$ and $u' = \mathtt{fold}(\mathtt{inl}\,\underline{n})$ for some $n \in \mathbb{N}$: immediate.
- $u = \mathtt{fold}(\mathtt{inr}\,g)$, $u' = \mathtt{fold}(\mathtt{inr}\,g')$ for some g, g' such that $\triangleright((g,g') \in [\![\tau \to \tau]\!])$. We then have that

  $$\varphi\, u \overset{p,1}{\rightsquigarrow} \mathtt{fold}(\mathtt{inr}\,\lambda y.h(g(h\,y)))$$

  and $\mathrm{id}\, u' \overset{p}{\rightsquigarrow} u'$ and so it suffices to show

  $$\triangleright(\lambda y.(h(g(h\,y)), g') \in [\![\tau \to \tau]\!]).$$

  We again show that these are related as values so take $\triangleright((v,v') \in [\![\tau]\!])$ and we need to show

  $$\triangleright\Big((h(g(h\,v)), g'\,v') \in [\![\tau]\!]^{\top\top}\Big).$$

  Take $\triangleright((E,E') \in [\![\tau]\!]^{\top})$. Löb induction hypothesis gives us that

  $$\triangleright((h', \mathrm{id}) \in [\![\tau \to \tau]\!]^{\top\top}),$$

  where h' is the body of h, i.e $h = \lambda x.h'\,x$. It is easy to see that this implies

  $$\triangleright((h, \mathrm{id}) \in [\![\tau \to \tau]\!]^{\top\top})$$

  and so by extending the contexts three times using lemmas analogous to Lemma 3.4.3 we get

  $$\triangleright\Big((E[h(g(h[\,]))], E'[g'[\,]]) \in [\![\tau]\!]^{\top}\Big).$$

  So, assuming $\triangleright(E'[g'\,v']\!\uparrow)$ we get $\triangleright(E[h(g(h\,v))]\underline{\uparrow})$, concluding the proof.

3.5 The Model for GTT

In this section, we present a model for the logic GTT, where all the properties we have used in the previous sections are justified. The model we consider is the topos of sheaves over the first uncountable ordinal ω_1 (in fact, any ordinal $\alpha \geq \omega_1$ would suffice). We assume some basic familiarity with topos theory, on the level described in [66]. We briefly recall the necessary definitions.

The objects of $\mathbf{Sh}(\omega_1)$ are sheaves over ω_1 considered as a topological space equipped with the Alexandrov topology. Concretely, this means that objects of $\mathbf{Sh}(\omega_1)$ are continuous functors from $(\omega_1 + 1)^{\mathrm{op}}$ to $\mathbf{Set}$. We think of ordinals as time, with smaller ordinals being the future. The restriction maps then describe the evolution of elements through time.

$\mathbf{Sh}(\omega_1)$ is a full subcategory of the category of presheaves $\mathbf{PSh}(\omega_1 + 1)$. The inclusion functor i has a left adjoint $\mathbf{a} : \mathbf{PSh}(\omega_1 + 1) \to \mathbf{Sh}(\omega_1)$ called the *associated sheaf functor*. Limits and exponentials are constructed as in presheaf categories. Colimits *are not* constructed pointwise as in presheaf categories, but they require also the application of the associated sheaf functor.

There is an essential geometric morphism

$$\Pi_1 \dashv \Delta \dashv \Gamma : \mathbf{Sh}(\omega_1) \to \mathbf{Set},$$

with Δ the *constant sheaf* functor, Γ the global sections functor and $\Pi_1(X) = X(1)$ the evaluation at 1 (we consider 0 to be the first ordinal). Given a set a, the constant sheaf $\Delta(a)$ is not the constant presheaf: rather it is equal to the singleton set 1 at stage 0, and to a at all other stages.

For a sheaf X, an element $\xi \in X(\nu)$ and $\beta \leq \nu$ we write $\xi|_\beta$ for the restriction $X(\beta \leq \nu)(\xi)$.

Analogously to the topos of trees [22], there is a "later" modality on *types*, i.e. a functor $\blacktriangleright : \mathbf{Sh}(\omega_1) \to \mathbf{Sh}(\omega_1)$ defined as (we consider 0 a limit ordinal)

$$\blacktriangleright X(\nu + 1) = X(\nu), \qquad \blacktriangleright X(\alpha) = X(\alpha) \text{ for } \alpha \text{ limit ordinal.}$$

There is an obvious natural transformation $\mathrm{next}^X : X \to \blacktriangleright X$.

The subobject classifier Ω is given by $\Omega(\nu) = \{\beta \mid \beta \leq \nu\}$ and its restriction maps are given by minimum. There is a natural transformation $\triangleright : \Omega \to \Omega$ given as $\triangleright_\nu(\beta) = \min\{\beta + 1, \nu\}$.

Kripke-Joyal semantics [59] is a way to translate formulas in the logic to statements about objects and morphisms of $\mathbf{Sh}(\omega_1)$; we refer to [66, Section VI.5] for a detailed introduction and further references. We now briefly explain the Kripke-Joyal semantics of GTT.

Let X be a sheaf and φ, ψ formulas in the internal language with a free variable of type X. Intuitively, for an ordinal ν and an element $\xi \in X(\nu)$, $\nu \Vdash \varphi(\xi)$ means that φ holds for ξ at stage ν. A formula φ is *valid* if it holds for all ξ and at all stages.

Let $\nu \leq \omega_1$ and $\xi \in X(\nu)$. The rules of Kripke-Joyal semantics are the usual ones (see, e.g., [66, Theorem VI.7.1]), specialised for our particular topology:

- $\nu \Vdash \bot$ iff $\nu = 0$;
- $\nu \Vdash \top$ always;
- $\nu \Vdash \varphi(t)(\xi)$ iff $[\![\varphi]\!]_\nu([\![t]\!]_\nu(\xi)) = \nu$, for a predicate symbol φ on X;
- $\nu \Vdash \varphi(\xi) \wedge \psi(\xi)$ iff $\nu \Vdash \varphi(\xi)$ and $\nu \Vdash \psi(\xi)$;
- $\nu \Vdash \varphi(\xi) \vee \psi(\xi)$ iff $\nu \Vdash \varphi(\xi)$ or $\nu \Vdash \psi(\xi)$;
- $\nu \Vdash \varphi(\xi) \to \psi(\xi)$ iff for all $\beta \leq \nu, \beta \Vdash \varphi(\xi|_\beta)$ implies $\beta \Vdash \psi(\xi|_\beta)$;
- $\nu \Vdash \neg\varphi(\xi)$ iff for all $\beta \leq \nu, \beta \Vdash \varphi(\xi|_\beta)$ implies $\beta = 0$.

Note that $0 \Vdash \varphi$ for any φ, as is usual in Kripke-Joyal semantics for sheaves over a space: intuitively, the stage 0 represents the impossible world. Moreover, if φ is a formula with free variables $x : X$ and $y : Y$, $\nu \leq \omega_1$ and $\xi \in X(\nu)$ then:

- For ν a successor ordinal: $\nu \Vdash \exists y : Y, \varphi(\xi, y)$ iff there exists $\xi' \in Y(\nu)$ such that $\nu \Vdash \varphi(\xi, \xi')$;
- For ν a limit ordinal: $\nu \Vdash \exists y : Y, \varphi(\xi, y)$ iff for all $\beta < \nu$ there exists $\xi_\beta \in Y(\beta)$ such that $\beta \Vdash \varphi(\xi|_\beta, \xi_\beta)$;
- $\nu \Vdash \forall y : Y, \varphi(\xi, y)$ iff for all $\beta \leq \nu$ and for all $\xi_\beta \in Y(\beta)$: $\beta \Vdash \varphi(\xi|_\beta, \xi_\beta)$.

The semantics of $\triangleright$ is as follows. Let φ be a predicate on X, then

$$\nu \Vdash \triangleright\varphi(\alpha) \text{ iff for all } \beta < \nu, \beta \Vdash \varphi(\alpha|_\beta).$$

For successor ordinals $\nu = \nu' + 1$ this reduces to

$$\nu + 1 \Vdash \triangleright\varphi(\alpha) \text{ iff } \nu' \Vdash \varphi(\alpha|_{\nu'}).$$

The predicate Total(X) in Definition 3.2.1 internalises the property that all X's restriction maps are surjections which intuitively means that elements at any stage β evolve from elements in the past. Total sheaves are also called *flabby* in homological algebra literature, but we choose to use the term total since it was used in previous work on guarded recursion to describe an analogous property.

The properties of $\triangleright$ stated in Section 3.2 can be proved easily using the Kripke-Joyal semantics. The rules are similar to the rules in Theorem 2.7 of [22], except the case of the existential quantifier in which the converse implication *does not hold*, even if we restrict to total and inhabited types, or even to constant sets. As a consequence, we cannot prove the internal Banach's fixed point theorem in the logic in the same way as in the topos of trees, cf. [22, Lemma 2.10].

In contrast to that in the topos of trees [22, Theorem 2.9], which requires the type X only to be inhabited, the internal Banach's fixed point theorem in $\mathbf{Sh}(\omega_1)$ (Theorem 3.2.3) has stronger assumptions: we require X to be total, which implies that it is inhabited. The additional assumption seems to be necessary and is satisfied in all the instances where we use the theorem. In particular, for a constant X, $\mathcal{P}(X)$ is total.

The operator $\neg\neg : \Omega \to \Omega$ gives rise to a function $\neg\neg_X$ on the lattice of subobjects $\mathbf{Sub}(X)$. In $\mathbf{Sh}(\omega_1)$, $\neg\neg_X$ preserves suprema[5] on each $\mathbf{Sub}(X)$ and therefore has a right adjoint $\Box_X : \mathbf{Sub}(X) \to \mathbf{Sub}(X)$ defined as

$$\Box_X P = \bigvee \{Q \mid \neg\neg Q \leq P\}.$$

[5] Recall that this is *not* the case in every topos.

If $X = \Delta(a)$ then $\square_X P$ has a simpler description:

$$\square_{\Delta(a)}(P)(\nu) = \begin{cases} 1 & \text{if } \nu = 0 \\ \bigcap_{\beta=1}^{\omega_1} P(\beta) & \text{otherwise.} \end{cases}$$

Thus for a predicate P on a constant set $\Delta(a)$, $\square(P)$ contains only those elements for which P holds at all stages.

However, in contrast to $\neg\neg$ which commutes with reindexing, $\square$ does not. There is a general reason for this: in any category with pullbacks, any deflationary operation $\square$ that preserves the top element and *is natural*, i.e. commutes with reindexing, is necessarily the identity [82, Proposition 4.2]. However

$$\Delta(f)^*\left(\square_{\Delta(a)}(P)\right) = \square_{\Delta(b)}(\Delta(f)^*(P))$$

for any $f : a \to b$ in **Set** and since Δ preserves products we do get that $\square$ in the logic commutes with substitution when restricted to constant contexts.

The external interpretation of $\uparrow$ is exactly the negation of the must-convergence predicate $\Downarrow$ from [23]. In particular, $\uparrow$ is a constant predicate. In contrast, $\underline{\uparrow}(\nu)$ is a set of expressions e such that there exists a reduction of length at least ν starting with e. This can easily be seen using the description of Kripke-Joyal semantics above. Thus, $\underline{\uparrow}$ is externally the pointwise complement of the stratified must-convergence predicate $\{\Downarrow_\beta\}_{\beta<\omega_1}$ from [23]. Then, the proof that $\square\underline{\uparrow} \ \to \uparrow$ corresponds to the proof that $\Downarrow \subseteq \bigcup_{\beta<\omega_1} \Downarrow_\beta$ in [23]. Here we technically see the need for indexing over ω_1.

Acknowledgements

This research was supported in part by the ModuRes Sapere Aude Advanced Grant from The Danish Council for Independent Research for the Natural Sciences (FNU) and in part by Microsoft Research through its PhD Scholarship Programme.

3.A The Topos $\mathbf{Sh}(\omega_1)$

We first describe the topos of sheaves over ω_1, the first uncountable ordinal. To be completely precise, we consider ω_1 as a topological space equipped with the Alexandrov topology arising from the usual ordering of ordinals. Thus the topos $\mathbf{Sh}(\omega_1)$ is a full subcategory of the category $\mathbf{PSh}(\omega_1+1)$, since the opens of ω_1 are exactly the downwards closed subsets of ω_1 which in turn correspond precisely to ω_1+1 (by von Neumann's construction of ordinals, they are exactly ω_1+1). We write 0 for the first ordinal. So objects are of the form

$$X(0) \longleftarrow X(1) \longleftarrow \cdots \longleftarrow X(\omega) \longleftarrow X(\omega+1) \longleftarrow \cdots$$

but not all such chains are sheaves. Sheaves are precisely the continuous functors from $(\omega_1+1)^{\mathrm{op}}$ to $\mathbf{Set}$ and morphisms are natural transformations.

$\mathbf{Sh}(\omega_1)$ is a topos. The inclusion functor $i : \mathbf{Sh}(\omega_1) \to \mathbf{PSh}(\omega_1+1)$ has a *left* adjoint $\mathbf{a}$, the associated sheaf functor. Limits and exponentials in $\mathbf{Sh}(\omega_1)$ are computed as in $\mathbf{PSh}(\omega_1+1)$, i.e. limits are pointwise and exponential X^Y is given at stage ν as

$$\mathbf{Hom}_{\mathbf{Sh}(\omega_1)}\Big(\mathbf{Hom}_{\omega_1+1}(\cdot,\nu)\times Y, X\Big).$$

Colimits, however, are not constructed as in presheaves, but are computed first as in presheaves followed by an application of the associated sheaf functor $\mathbf{a}$.

We denote the lattice of subobjects of an object X by $\mathbf{Sub}(X)$ and we denote reindexing along $f : X \to Y$ by $f^* : \mathbf{Sub}(Y) \to \mathbf{Sub}(X)$. Since $\mathbf{Sh}(\omega_1)$ is a topos, each subobject lattice is a complete Heyting algebra. Further, we can show that each subobject lattice is in fact a bi-Heyting algebra, that is, a Heyting algebra with a $\setminus$ operation that is left adjoint to disjunction, i.e., $X \setminus Y \leq Z \leftrightarrow X \leq Y \vee Z$. The existence of $\setminus$ can be shown by using explicit descriptions of operations on the subobject lattice below. This operation is related to the $\Box$ modality and it makes $\mathbf{Sh}(\omega_1)$ a bi-Heyting topos [82].

Subobject classifier The subobject classifier Ω is given by closed sieves, which are exactly the maximal sieves. More precisely the subobject classifier at ν is given by sieves S such that $\bigvee S \in S$. These sieves therefore correspond to ordinals smaller or equal to ν. Explicitly

$$\Omega(\nu) = \{\beta \mid \beta \leq \nu\}$$

(note that von Neumann's construction of ordinals gives $\Omega(\nu) = \nu + 1$).

The restriction maps are given by minimum, i.e.

$$r_\nu^\beta : \Omega(\beta) \to \Omega(\nu)$$
$$r_\nu^\beta(\gamma) = \min\{\beta, \nu\}$$

and the map **true** : $1 \to \Omega$ maps $*$ to the maximal sieve

$$\mathbf{true}_\nu = \nu.$$

Note that this is different from the construction of the subobject classifier for presheaves, where $\Omega(\nu)$ is all the sieves on ν, including the empty sieve.

Given a subobject $m : A \leq X$ the characteristic map $\chi^m : X \to \Omega$ is given by

$$\chi^m_\nu(x) = \bigvee \left\{\beta \leq \nu \;\middle|\; m_\beta^{-1}\left[x|_\beta\right] \neq \emptyset\right\}$$

i.e. (if we assume $A(\nu) \subseteq X(\nu)$, which we are allowed to)

$$\chi^m_\nu(x) = \bigvee \left\{\beta \leq \nu \;\middle|\; x|_\beta \in A(\beta)\right\}.$$

Note that the supremum of an empty set is 0, the first ordinal, which is an element of all $\Omega(\nu)$, so the characteristic map is well-defined.

Some of the properties later will not hold for all the sheaves but only for a certain subset.

Definition 3.A.1. A sheaf X is total if the restriction maps are surjections. ♦

A way to think of totality is by thinking of ordinals as time with smaller ordinals being the future. Then X being total means that elements at any stage ν are only those that have evolved from some previous stages. They did not just suddenly appear.

Being total can also be characterised internally by the property that the function next^X is internally surjective. The equivalence of these two notions of totality can be shown using the Kripke-Joyal semantics.

Relationship to Set

The global sections geometric morphism $\Delta \vdash \Gamma : \mathbf{Sh}(\omega_1) \to \mathbf{Set}$ is an essential geometric morphism. That is, there is an adjoint triple

$$\Pi_1 \dashv \Delta \dashv \Gamma$$

where $\Pi_1, \Gamma : \mathbf{Sh}(\omega_1) \to \mathbf{Set}$ and $\Delta : \mathbf{Set} \to \mathbf{Sh}(\omega_1)$ are given as follows

$$\begin{aligned}
\Pi_1(X) &= X(1) = \operatorname*{colim}_{\nu \leq \omega_1} X(\nu) \\
\Gamma(X) &= \mathbf{Hom}_{\mathbf{Sh}(\omega_1)}(1, X) = \lim_{\nu \leq \omega_1} X(\nu) = X(\omega_1) \\
\Delta(a)(\nu) &= \begin{cases} 1 & \text{if } \nu = 0 \\ a & \text{otherwise} \end{cases}
\end{aligned}$$

Δ is the *constant sheaf* functor. Note that it is not the *constant presheaf* functor.

The adjunction $\Pi_1 \dashv \Delta$ gives rise to an adjunction between subobject lattices. More precisely for any set a, there is an adjunction

$$\Pi_1^a : \mathbf{Sub}_{\mathbf{Sh}(\omega_1)}(\Delta(a)) \to \mathbf{Sub}_{\mathbf{Set}}(a) : \Delta^a$$

where $\Delta^a(b) = \Delta(b)$ and $\Pi_1^a(A) = A(1)$. These adjunctions are natural in the sense that for any function $\alpha : a' \to a$ we have $\alpha^* \circ \Pi_1^a = \Pi_1^{a'} \circ \Delta(\alpha)^*$, which is easy to check directly.

Thus, $\Delta \dashv \Gamma : \mathbf{Sh}(\omega_1) \to \mathbf{Set}$ is an open geometric morphism [66, Definition IX.6.2] which further means that Δ preserves models of first-order logic in the sense of [66, Theorem X.3.1]. In practice, this means that whatever predicate on a constant set we define in the internal logic using only the first-order fragment and other constant relations and predicates will be constant.

Another way to see that $\Delta \dashv \Gamma$ is an open geometric morphism is by the fact that $\mathbf{Sh}(1)$ (sheaves on a one point space) is isomorphic to the category $\mathbf{Set}$. Since the unique map $\omega_1 \to 1$ is *open*, the induced geometric morphism is open. It can easily be seen that the direct image functor induced by this unique morphism is (isomorphic to) Γ. By uniqueness of adjoints the inverse image functor must then be (isomorphic to) Δ.

Moreover, Π_1 is a logical morphism, meaning it is a cartesian closed functor that preserves Ω. This is easy to see manually, by computing. However, there is a more general argument available. It proceeds as follows. The set $\{0\}$ is an open subset of ω_1. Let $i : \{0\} \to \omega_1$ be the inclusion and let $i_* : \mathbf{Sh}(\{0\}) \to \mathbf{Sh}(\omega_1)$ be the direct image functor. Recall that

$$i_*(F)(U) = F\left(i^{-1}[U]\right)$$

and in our particular case we have

$$i_*(F)(\nu) = \begin{cases} F(\emptyset) & \text{if } \nu = 0 \\ F(\{0\}) & \text{if } \nu \neq 0 \end{cases}$$

Recall that $\mathbf{Sh}(\{0\})$ is isomorphic to $\mathbf{Set}$ with the isomorphism $\xi : \mathbf{Set} \to \mathbf{Sh}(\{0\})$ given by $\xi(a)(\emptyset) = 1$, $\xi(a)(\{0\}) = a$ and the obvious restriction. Thus we see that $\Delta = i_* \circ \xi$. The inverse image functor i^* is left adjoint to i_*. Since we also have $\xi \circ \Pi_1 \dashv \Delta \circ \xi^{-1}$ we have that i^* is naturally isomorphic to $\xi \circ \Pi_1$ or equivalently $\Pi_1 \cong \xi^{-1} \circ i^*$.

Since $\{0\}$ is an open subset of $\mathbf{Sh}(\omega_1)$ this makes $\mathbf{Set}$ (equivalent to) an open subtopos of $\mathbf{Sh}(\omega_1)$ [53, Section A4.5].

Moreover, we have by [66, Theorem 6, Corollary 7] that $\mathbf{Set}$ is equivalent to a category of j-sheaves for some universal closure operator j. We will see in Section 3.A that this local operator is exactly the $\neg\neg$-closure. Thus there exists a geometric morphism $e : \mathbf{Set} \to \mathbf{Sh}_j(\mathbf{Sh}(\omega_1))$ such that $e^* \circ a \cong \Pi_1$ and $\iota \circ e_* \cong \Delta$ where ι is the inclusion $\mathbf{Sh}_j(\mathbf{Sh}(\omega_1)) \to \mathbf{Sh}(\omega_1)$.

Putting all of it together we have $\mathbf{Sh}_j(\mathbf{Sh}(\omega_1))$ is an open subtopos of $\mathbf{Sh}(\omega_1)$ since it is equivalent to **Set** which is equivalent to $\mathbf{Sh}(\{0\})$. By [53, Proposition 4.5.1] this means that $a : \mathbf{Sh}(\omega_1) \to \mathbf{Sh}_j(\mathbf{Sh}(\omega_1))$ is a logical functor and since $\Pi_1 \cong e^* \circ a$, with e^* being part of an equivalence, which means that it is a logical functor, we have that Π_1 is logical.

This means that Π_1 preserves validity of formulas in the internal language.

Description of the Heyting algebra structure of the subobject lattices

We give here explicit descriptions of operations on each subobject lattice $\mathbf{Sub}(X)$. Let $X \in \mathbf{Sh}(\omega_1)$, $A, B \in \mathbf{Sub}(X)$ and $\beta \leq \omega_1$. We have

$$\begin{aligned}
\top &= X \\
\bot(\beta) &= \begin{cases} 1 & \text{if } \beta = 0 \\ \emptyset & \text{otherwise} \end{cases} \\
(A \wedge B)(\beta) &= A(\beta) \cap B(\beta) \\
\left(\bigwedge_{i \in I} A_i\right)(\beta) &= \bigcap_{i \in I} A_i(\beta) \\
(A \Rightarrow B)(\beta) &= \left\{ x \in X(\beta) \,\middle|\, \forall \gamma \leq \beta, x|_\gamma \in A(\gamma) \to x|_\gamma \in B(\gamma) \right\} \\
\left(\bigvee_{i \in I} A_i\right)(\beta) &= \left\{ x \in X(\beta) \,\middle|\, \bigvee \left\{ \gamma \leq \beta \,\middle|\, \exists i \in I, x|_\gamma \in A_i(\gamma) \right\} = \beta \right\}
\end{aligned}$$

Let further $Y \in \mathbf{Sh}(\omega_1)$ and $\varphi : X \to Y$. Then $\exists_\varphi, \forall_\varphi : \mathrm{Sub}(X) \to \mathrm{Sub}(Y)$ are given by the following equations

$$\begin{aligned}
\exists_\varphi(A)(\beta) &= \left\{ y \in Y(\beta) \,\middle|\, \bigvee \left\{ \gamma \leq \beta \,\middle|\, \exists a \in A(\gamma), \varphi_\gamma(a) = y|_\gamma \right\} = \beta \right\} \\
\forall_\varphi(A)(\beta) &= \left\{ y \in Y(\beta) \,\middle|\, \forall \gamma \leq \beta, \varphi_\gamma^{-1}\left[y|_\gamma \right] \subseteq A(\gamma) \right\}
\end{aligned}$$

and $\varphi^* : \mathrm{Sub}(Y) \to \mathrm{Sub}(X)$ is given by

$$\varphi^*(C)(\beta) = \left\{ x \in X(\beta) \,\middle|\, \varphi_\beta(x) \in C(\beta) \right\}$$

These are standard results from [66, III.8] specialised for a particular space with a particular topology.

Using these descriptions it is easy to see that the $\neg\neg_X : \mathbf{Sub}(X) \to \mathbf{Sub}(X)$ is given as follows

$$(\neg\neg_X A)(\nu) = \begin{cases} 1 & \text{if } \nu = 0 \\ \{x \in X(\nu) \mid x|_1 \in A(1)\} & \end{cases}$$

(intuitively, this says that something is not impossible if it will eventually happen (smaller indices are the future)). Observe that in fact

$$\neg\neg_X P = U_X \to P$$

where $U_X \leq X$ is given as $U_X(0) = 1$, $U_X(1) = X(1)$ and $U_X(\nu) = \emptyset$ otherwise. This means that $\neg\neg$ is an *open* local operator [53, Section 4.5].

From these explicit descriptions we can see that $\neg\neg_X$ arises from the functor $\Delta \circ \Pi_1$ as follows. Let $\blacklozenge = \Delta \circ \Pi_1$. Note that $\blacklozenge \dashv \Delta \circ \Gamma$ which means that it preserves all colimits. It is easy to see that $\blacklozenge$ preserves all limits since they are constructed pointwise. We will also see in Section 3.A that it has a left adjoint which implies that it preserves all limits. In particular it preserves monomorphisms, thus subobjects. Hence given a subobject $m : A \leq X$ we have a subobject $\blacklozenge m : \blacklozenge A \leq \blacklozenge X$. Further, $\blacklozenge$ is a monad, thus there is a unit $\eta_X : X \to \blacklozenge X$. It is then easy to see that

$$\begin{tikzcd}
\neg\neg A \arrow[r, dashed] \arrow[d, dashed] \arrow[dr, phantom, very near start, "\lrcorner"] & \blacklozenge A \arrow[d, "\blacklozenge m"] \\
X \arrow[r, "\eta_X"] & \blacklozenge X
\end{tikzcd}$$

is a pullback diagram. Another way to state this is that $\neg\neg_X : \mathbf{Sub}(X) \to \mathbf{Sub}(X)$ is given as $\neg\neg_X(A) = \eta_X^*(\blacklozenge A)$.

Using this description we can easily see that $\neg\neg_X$ preserves suprema. Indeed

$$\neg\neg_X\left(\bigvee_i A_i\right) = \eta_X^*\left(\blacklozenge\left(\bigvee_i A_i\right)\right)$$

and since suprema in $\mathbf{Sub}(X)$ are constructed using a coproduct in $\mathbf{Sh}(\omega_1)$ followed by images, which are in turn constructed using limits and colimits of $\mathbf{Sh}(\omega_1)$, which are preserved by $\blacklozenge$, we have

$$= \eta_X^*\left(\bigvee_i \blacklozenge A_i\right)$$

and since $\eta_X^* : \mathbf{Sub}(X) \to \mathbf{Sub}(X)$ has a right adjoint, the generalised forall quantifier $\forall_{\eta_X}$, it preserves suprema, hence

$$= \bigvee_i \eta_X^*(\blacklozenge A_i) = \bigvee_i \neg\neg_X A_i.$$

Note that as in all toposes, $\neg\neg$ is preserved by reindexing functors, i.e. for any $\varphi : X \to Y$, $\varphi^* \circ \neg\neg_Y = \neg\neg_X \circ \varphi^*$.

We can also easily compute manually using explicit descriptions of operations above, that $\neg\neg_X$ does indeed preserve suprema.

The $\Box$ modality

Since $\neg\neg_X$ preserves suprema it has a *right* adjoint [12, Corollary 9.32]. We denote this right adjoint to $\neg\neg_X$ by $\Box_X$. It can be defined as

$$\Box_X(P) = \bigvee \{Q \mid \neg\neg_X Q \leq P\}.$$

$\Box_X$ is an interior operation on the subobject lattice $\mathbf{Sub}(X)$ and $\Box_X P$ it can be characterised as the greatest element smaller than P that is also $\neg\neg_X$ closed. The fact that $\Box_X P$ is $\neg\neg$-closed is proved in Lemma 3.A.2 and the fact that it preserves $\neg\neg$-closed subobjects is proved in Corollary 3.A.4.

Lemma 3.A.2. *For any object X and subobject $P \leq X$, $\neg\neg_X \Box_X(P) = \Box_X(P)$.* ◊

Proof. By definition of $\Box_X$ we have

$$\neg\neg_X(\Box_X(P)) = \neg\neg_X\left(\bigvee \{Q \mid \neg\neg_X Q \leq P\}\right)$$

and since $\neg\neg_X$ preserves suprema

$$= \bigvee \{\neg\neg_X Q \mid \neg\neg_X Q \leq P\} = \bigvee \{Q \mid \neg\neg_X Q \leq P\} = \Box_X(P)$$

The second to last equality holds because for each Q, we have $\neg\neg_X Q \geq Q$ and $\neg\neg_X \neg\neg_X Q = \neg\neg_X Q$. 𝔔𝔈𝔇

Corollary 3.A.3. *For any object X and subobject $P \leq X$, we have $\Box_X P \leq P$.* ◊

Proof. Since $\neg\neg_X \dashv \Box_X$ we have $\Box_X P \leq \Box_X P \leftrightarrow \neg\neg_X \Box_X P \leq P$. Lemma 3.A.2 concludes the proof. 𝔔𝔈𝔇

Corollary 3.A.4. *For any object X and $P \leq X$, $\Box_X(\neg\neg_X P) = \neg\neg_X P$.* ◊

Proof. For any P, $\neg\neg_X \neg\neg_X P = \neg\neg_X P$. Thus $\neg\neg_X \neg\neg_X P \leq \neg\neg_X P$. Since $\Box_X$ is right adjoint to $\neg\neg_X$ we get $\neg\neg_X P \leq \Box_X \neg\neg_X P$. The other direction follows directly from Corollary 3.A.3. 𝔔𝔈𝔇

Corollary 3.A.5. *For any object X and subobject $P \leq X$, $\Box_X(\Box_X P) = \Box_X P$.* ◊

Proof. $\Box_X(\Box_X P) \leq \Box_X P$ follows from Corollary 3.A.3. The other direction follows from the fact that $\Box_X$ is right adjoint to $\neg\neg_X$ and Lemma 3.A.2 since by adjointness we have $\Box_X P \leq \Box_X \Box_X P \leftrightarrow \neg\neg_X \Box_X P \leq \Box_X P$ and the right hand side holds by Lemma 3.A.2. 𝔔𝔈𝔇

We now state and prove how $\Box_X$ commutes with some other operations on $\mathbf{Sub}(X)$. In particular, it commutes with conjunction, top, bottom and universal quantification.

Proposition 3.A.6. *Let X and Y be types, $P, Q \in \mathbf{Sub}(X)$ and $\varphi : X \to Y$ a morphism. The following hold*

1. $\Box_X \top = \top$
2. $\Box_X \bot = \bot$
3. $\Box_X(P \wedge Q) = \Box_X P \wedge \Box_X Q$
4. $\Box_Y\left(\forall_\varphi P\right) = \forall_\varphi(\Box_X P)$.

◊

Proof. Since $\Box_X$ is a right adjoint it preserves limits in $\mathbf{Sub}(X)$. $\top$ and $\wedge$ are limits, therefore $\Box_X$ necessarily preserves them.

Corollary 3.A.3 shows that $\Box_X P \leq P$ for any P. In particular, this holds for $\bot$ and since $\bot$ is the least element of $\mathbf{Sub}(X)$, we get $\Box_X \bot = \bot$.

To see $\Box_Y\left(\forall_\varphi P\right) = \forall_\varphi(\Box_X P)$ we use the fact that $\forall_\varphi$ is right adjoint to φ^* and that φ^* commutes with $\neg\neg_X$. Thus for any $R \in \mathbf{Sub}(Y)$ we have

$$\begin{aligned} R \leq \Box_Y\left(\forall_\varphi P\right) &\leftrightarrow \neg\neg_Y R \leq \forall_\varphi P \\ &\leftrightarrow \varphi^*(\neg\neg_Y R) \leq P \\ &\leftrightarrow \neg\neg_X(\varphi^* R) \leq P \\ &\leftrightarrow \varphi^* R \leq \Box_X P \\ &\leftrightarrow R \leq \forall_\varphi(\Box_X P) \end{aligned}$$

Thus picking R to be $\Box_Y\left(\forall_\varphi P\right)$ or $\forall_\varphi(\Box_X P)$ we get both approximations, and hence the equality

$$\Box_Y\left(\forall_\varphi P\right) = \forall_\varphi(\Box_X P).$$

𝔔𝔈𝔇

□ and substitution In contrast to $\neg\neg_X$ which is preserved by reindexing functors, $\Box_X$ is not, that is, for $\varphi : Y \to X$ it *is not in general the case* that $\Box_Y \circ \varphi^* = \varphi^* \circ \Box_X$. One direction, however, does hold.

Proposition 3.A.7. *For any $\varphi : Y \to X$,*

$$\varphi^* \circ \Box_X \leq \Box_Y \circ \varphi^*.$$

If φ is an isomorphism then the two sides are also equal. ◊

Proof. By definition of $\Box_X$ and the fact that φ^* has a right adjoint we have

$$\begin{aligned}\varphi^*(\Box_X(P)) &= \varphi^*\left(\bigvee\{Q \mid \neg\neg_X Q \le P\}\right)\\ &= \bigvee\{\varphi^* Q \mid \neg\neg_X Q \le P\}\\ &\le \bigvee\{\varphi^* Q \mid \varphi^*(\neg\neg_X Q) \le \varphi^*(P)\}\\ &= \bigvee\{\varphi^* Q \mid \neg\neg_Y(\varphi^* Q) \le \varphi^*(P)\}\\ &\le \bigvee\{R \mid \neg\neg_Y R \le \varphi^*(P)\}\\ &= \Box_Y(\varphi^* P)\end{aligned}$$

If φ were an isomorphism it would preserve and also reflect order and every element of the lattice would be in the image. Thus the chain can be strengthened to show that in this case $\Box_Y(\varphi^* P)$ and $\varphi^*(\Box_X(P))$ are equal. 𝔔𝔈𝔇

The fact that $\Box_X$ is not natural in X implies that there is no morphism $\Box : \Omega \to \Omega$ such that $\Box_X$ would arise from it, as is the case for $\neg\neg_X$. Note that it is not a coincidence that $\Box_X$ is not natural but a fundamental limitation of interior operations. If $\Box_X$ were natural it would have to be the identity. More precisely, any operation on the subobject lattices that preserves $\top$ and is deflationary and natural in X must be the identity (provided the category satisfies some minimal requirements). This is proved in [82, Proposition 4.2] (see also Proposition 4.1 of loc. cit.).

$\Box_X$ does commute with exchange, however, but in general not with contraction and weakening. It does commute with weakening in the special case proved in Corollary 3.A.9 below. For the proof we need the following lemma stating that $\neg\neg$ commutes with $\exists_\pi$ for suitable π.

Lemma 3.A.8. *Let X and Y be types and $\pi : X \times Y \to X$ the projection. Then $\exists_\pi \circ \neg\neg \le \neg\neg \circ \exists_\pi$ always holds. If Y is total the converse also holds.* ◊

Proof. $\exists_\pi$ is the left adjoint to π^*. Thus

$$\begin{aligned}\exists_\pi(\neg\neg Q) \le \neg\neg(\exists_\pi Q) &\leftrightarrow \neg\neg Q \le \pi^*(\neg\neg(\exists_\pi Q))\\ &\leftrightarrow \neg\neg Q \le \neg\neg(\pi^*(\exists_\pi Q))\\ &\leftarrow Q \le \pi^*(\exists_\pi Q)\end{aligned}$$

and the last holds because $\pi^* \circ \exists_\pi$ is a closure.

Now suppose Y is total and let $Q \in \mathbf{Sub}(X \times Y)$. First, let ν be a successor ordinal. Let $x \in \neg\neg(\exists_\pi(Q))(\nu)$. By definition $x|_1 \in \exists_\pi(Q)(1)$, which further implies there exists $y \in Y(1)$, such that $(x|_1, y) \in Q(1)$. Since Y is total there exists a $y' \in Y(\nu)$, such that $y'|_1 = y$. Thus $(x|_1, y'|_1) \in Q(1)$ and so $(x, y') \in \neg\neg Q(\nu)$. This means that $x \in \exists_\pi(\neg\neg Q)(\nu)$.

Since this holds for all successor ordinals, it must also hold for limit ordinals (actually, the same manual proof would also suffice, but it is more complicated to write it down). 𝔔𝔈𝔇

The restriction on total objects Y is necessary. Consider any total X and let Y be an object which at stage at stage 1 is some nonempty set and at greater stages is the empty set. Suppose $P = X \times Y$, i.e. the top element. Then $\exists_\pi(\neg\neg P)(2) = \emptyset$, however $\neg\neg(\exists_\pi(P))(2)$ is not empty (since X is total).

Corollary 3.A.9. *Let $\pi : X \times Y \to X$ be the projection. If Y is total (restrictions are surjections) then $\pi^* \circ \Box_X = \Box_{X\times Y} \circ \pi^*$.* ◊

Proof. In light of Proposition 3.A.7 we only need to show. $\pi^* \circ \Box_X \geq \Box_{X\times Y} \circ \pi^*$. We have

$$\begin{aligned}
\Box_{X\times Y}(\pi^* Q) \leq \pi^*(\Box_X Q) &\leftrightarrow \exists_\pi \Box_{X\times Y}(\pi^* Q) \leq \Box_X Q \\
&\leftrightarrow \neg\neg(\exists_\pi \Box_{X\times Y}(\pi^* Q)) \leq Q \\
&\leftrightarrow \exists_\pi \neg\neg(\Box_{X\times Y}(\pi^* Q)) \leq Q \\
&\leftrightarrow \neg\neg(\Box_{X\times Y}(\pi^* Q)) \leq \pi^* Q \\
&\leftrightarrow \Box_{X\times Y}(\pi^* Q) \leq \Box_{X\times Y}(\pi^* Q)
\end{aligned}$$

QED

Exchange is reindexing by an isomorphism. Therefore by Proposition 3.A.7 it preserves $\Box$.

$\Box$ modality on constant types

If $X = \Delta(a)$ for some set a then $\Box_X$ has a much simpler description.

Lemma 3.A.10. *If $X = \Delta(a)$ and $P \leq X$ then*

$$\Box_X(P)(\nu) = \begin{cases} 1 & \textit{if } \nu = 0 \\ \bigcap_{\nu=1}^{\omega_1} P(\nu) & \textit{otherwise} \end{cases}.$$

◊

Proof. Since adjoints are unique we only need to show that $\Box_X$ is right adjoint to $\neg\neg_X$. On constant objects the definition of $\neg\neg_X$ simplifies to

$$(\neg\neg_X P)(\nu) = \begin{cases} 1 & \text{if } \nu = 0 \\ P(1) & \text{otherwise} \end{cases}.$$

Thus suppose $\neg\neg_X P \leq Q$. In particular, this means that for all $\nu \geq 1$, $P(1) \subseteq Q(\nu)$. Thus $P(1) \leq \bigcap_{\nu=1}^{\omega_1} Q(\nu)$ and since restrictions on X are inclusions, meaning that $P(\nu) \subseteq P(1)$ for any $\nu \geq 1$, we get $P \leq \Box_X Q$.

Conversely, suppose $P \leq \Box_X Q$. Thus $P(\beta) \leq \bigcap_{\nu=1}^{\omega_1} Q(\nu)$ for all $\beta \geq 1$. In particular this means that $P(1) \subseteq \bigcap_{\nu=1}^{\omega_1} Q(\nu)$ and so $P(1) \subseteq Q(\nu)$ for any $\nu \geq 1$, meaning that $\neg\neg_X P \leq Q$. QED

Using this description we can show that $\Box_X$ is preserved by reindexing functors arising from maps between constant sheaves.

Proposition 3.A.11. *Let a, b be sets and $f : a \to b$. Then*

$$\Delta(f)^* \circ \Box_{\Delta(b)} = \Box_{\Delta(a)} \circ \Delta(f)^*.$$

◇

Proof. Let $\nu \geq 1$ (for $\nu = 0$ there is nothing to prove). By a simple calculation we have

$$\left(\Delta(f)^*(\Box_{\Delta(b)}(P))\right)(\nu) = f^{-1}\left[\bigcap_{\nu=1}^{\omega_1} P(\nu)\right]$$

and since preimages preserve intersections we get

$$\begin{aligned} &= \bigcap_{\beta=1}^{\omega_1} f^{-1}\left[P(\beta)\right] \\ &= \bigcap_{\beta=1}^{\omega_1} (\Delta(f)^*(P)(\beta)) \\ &= \left(\Box_{\Delta(a)}\left(\Delta(f)^*(P)\right)\right)(\nu) \end{aligned}$$

$\mathfrak{QED}$

Note that any morphism $\varphi : \Delta(a) \to \Delta(b)$ is of the form $\varphi = \Delta(f)$ for some (unique) $f : a \to b$, i.e. Δ is full and faithful. Thus if we restrict to constant contexts $\Box$ commutes with substitution and so we may work informally with $\Box$ as with $\neg\neg$ or any other logical operation.

The $\neg\neg$ modality

It is easy to see that $\blacklozenge$ preserves all limits. Indeed, Π_1 also has a left adjoint σ_1 defined as follows

$$\begin{aligned} \sigma_1(a)(0) &= 1 \\ \sigma_1(a)(1) &= a \\ \sigma_1(a)(\nu) &= \emptyset \quad \text{for } \nu \geq 1 \end{aligned}$$

which then means that $\blacklozenge$ has a left adjoint $\sigma_1 \circ \Pi_1$ and thus it must preserve all limits. This then implies, using the fact that $\neg\neg_X(A) = \eta_X^*(\blacklozenge A)$ and that limits in subobject lattices are computed from limits in **Sh**(ω_1), that $\neg\neg_X$ preserves all limits. In particular, it preserves all infima, meaning that it also has a left adjoint, which we call $\diamond_X$. It is given simply as

$$\diamond_X(P) = \bigwedge\{Q \mid P \leq \neg\neg Q\}$$

however it can be described in an elementary way as

Lemma 3.A.12.

$$\diamond_X(P)(\nu) = \begin{cases} 1 & \textit{if } \nu = 0 \\ P(1) & \textit{if } \nu = 1 \\ \emptyset & \textit{otherwise} \end{cases}$$

◊

Proof. By uniqueness of adjoints we only need to show that $\diamond_X$ is left adjoint to $\neg\neg_X$.

First suppose $\diamond_X P \leq Q$. We are to show $P \leq \neg\neg Q$. Let $x \in P(\nu)$. Then $x|_1 \in P(1)$, thus by assumption in $Q(1)$. Hence by the explicit description of $\neg\neg$ we have that $x \in \neg\neg Q(\nu)$.

Now suppose $P \leq \neg\neg Q$ and we are to show $\diamond_X P \leq Q$. The only non-trivial inclusion is at stage 1. So take $x \in P(1)$. Then $x \in Q$ by assumption (since $\neg\neg Q(1) = Q(1)$), concluding the proof. 𝔔𝔈𝔇

Note that $\sigma_1 \circ \Pi_1$ is a comonad, its counit is mono and using the descriptions above we have that $\diamond_X(P) = \sigma_1(\Pi_1(m_P)); \varepsilon_X$ where m_P is the mono belonging to P and ε_X is the counit at X.

In contrast to $\Box$, however, $\diamond$ does commute with reindexing. Thus it defines a map $\diamond : \Omega \to \Omega$ which is simply given as

$$\diamond_\nu(\beta) = \begin{cases} 0 & \text{if } \nu = 0 \\ 1 & \text{otherwise} \end{cases}$$

or equivalently as $\diamond_\nu(\beta) = \min\{1, \beta\}$ which then clearly shows that $\diamond$ is natural, i.e. a morphism $\Omega \to \Omega$.

The fact that $\diamond$ is natural has as a consequence the fact that $\neg\neg$ commutes with universal quantification.

Lemma 3.A.13. *Let $\varphi : X \to Y$ be a morphism and $\forall_\varphi$ the right adjoint to φ^*. Then $\neg\neg \circ \forall_\varphi = \forall_\varphi \circ \neg\neg$.* ◊

Proof. We show two inequalities using properties of adjoints. We have for any R

$$\begin{aligned} R \leq \neg\neg\left(\forall_\varphi Q\right) &\leftrightarrow \diamond R \leq \forall_\varphi Q \\ &\leftrightarrow \varphi^*(\diamond R) \leq Q \\ &\leftrightarrow \diamond(\varphi^* R) \leq Q \\ &\leftrightarrow \varphi^* R \leq \neg\neg Q \\ &\leftrightarrow R \leq \forall_\varphi(\neg\neg Q) \end{aligned}$$

Which implies that $\forall_\varphi(\neg\neg Q)$ and $\neg\neg\left(\forall_\varphi Q\right)$ are equal by picking R to be the two subobjects and using reflexivity of $\leq$ the equivalence just proved. 𝔔𝔈𝔇

Note that the lemma states that $\neg\neg$ commutes over all universals, not just the usual ones arising from weakening. This is in contrast to the situation with $\neg\neg$ and existentials. An analogous proof to the above shows that $\diamond$ commutes over existentials. The reason we cannot use the same proof to show that $\neg\neg$ commutes over existentials, even though it has a right adjoint, is that $\square$ does not commute with reindexing.

The fact that $\diamond$ commutes with reindexing is not contrary to [82, Proposition 4.2] since $\diamond$ does not preserve truth. It is however a comonad, i.e. $\diamond P \leq P$ and $\diamond \diamond P = \diamond P$.

Further, we can show that $\neg\neg$ commutes with implication. Indeed, since $\neg\neg$ preserves limits in subobject lattices we immediately have $\neg\neg(P \to Q) \leq \neg\neg P \to \neg\neg Q$. For the other direction we first recall that $P \to Q = \forall_{m_P}\left(m_P^* Q\right)$ where $m_P : P \to X$ is the inclusion of P into X: To that end we use the fact that $P \wedge Q = m_P^*(Q)$. We have

$$P \wedge R \leq Q \leftrightarrow P \wedge R \leq P \wedge Q \leftrightarrow m_P^*(R) \leq m_P^*(Q) \leftrightarrow R \leq \forall_{m_P}(m_P^*(Q))$$

and by uniqueness of adjoints also $P \to Q = \forall_{m_P}\left(m_P^* Q\right)$. Using this, we can prove the following.

Lemma 3.A.14. $\neg\neg(P \to Q) = P \to \neg\neg Q = \neg\neg P \to \neg\neg Q$ ◊

Proof. Since $\neg\neg(P \to Q) \wedge P \leq \neg\neg(P \to Q \wedge P) \leq \neg\neg Q$ we get $\neg\neg(P \to Q) \leq P \to \neg\neg Q$ and also $\neg\neg(P \to Q) \leq \neg\neg P \to \neg\neg Q$.

By the above characterisation and the fact that $\neg\neg$ commutes over reindexing and universals we have.

$$\neg\neg(P \to Q) = \neg\neg\left(\forall_{m_P}(m_P^*(Q))\right) = \forall_{m_P}(m_P^*(\neg\neg Q)) = P \to \neg\neg Q$$

It is easy to see that

$$\neg\neg P \to \neg\neg Q \leq P \to \neg\neg Q,$$

with the same calculation as above (also, we can always weaken the precondition). We thus have $\neg\neg P \to \neg\neg Q \leq P \to \neg\neg Q = \neg\neg(P \to Q)$. This concludes the proof. 𝔔𝔈𝔇

The ▹ and ► modalities

Recall that the first ordinal, 0, is a limit ordinal. It is the limit of the empty sequence. There is a functor $\blacktriangleright : \mathbf{Sh}(\omega_1) \to \mathbf{Sh}(\omega_1)$ defined by

$$\begin{aligned} \blacktriangleright(X)(\nu+1) &= X(\nu) \\ \blacktriangleright(X)(\alpha) &= X(\alpha) && \text{for a limit ordinal } \alpha \end{aligned}$$

and the obvious action on morphisms. There is a natural transformation $\mathrm{next}^X : X \to \blacktriangleright(X)$ defined as

$$\begin{aligned} \mathrm{next}^X_{\nu+1} &= r_\nu \\ \mathrm{next}^X_\alpha &= \mathrm{id}_{X(\alpha)} && \text{for a limit ordinal } \alpha \end{aligned}$$

where $r_\nu = X(\nu \leq \nu+1) : X(\nu+1) \to X(\nu)$ is the restriction map of X.

Using $\blacktriangleright$ we can define a notion of contractiveness of morphisms.

Definition 3.A.15. A morphism $\varphi : X \to Y$ is *(externally) contractive* if there exists a morphism $g : \blacktriangleright X \to Y$, such that $f = \mathrm{next}^X;g$. ♦

A useful property of contractive endomorphisms is that they have unique fixed points.

Proposition 3.A.16. *If $f : X \to X$ is a contractive morphism then there exists a unique $e : 1 \to X$ such that $e;f = e$.* ♢

Proof. Let $f = \mathrm{next}^X;g$. By construction of $\blacktriangleright$ we have that $g_{\nu+1} : X(\nu) \to X(\nu+1)$ and $X(0) = 1$. We thus define a global section by induction as follows[6] (again, α is a limit ordinal)

$$\begin{aligned} e_0(*) &= g_0(*) \\ e_{\nu+1}(*) &= g_{\nu+1}(e_\nu(*)) \\ e_\alpha &= \lim_{\nu<\alpha} e_\nu \end{aligned}$$

where $\lim_{\nu<\alpha} e_\nu$ denotes the unique element of $X(\alpha)$ that restricts to e_ν, $\nu < \alpha$. Now it is obvious that $r_0(e_1(*)) = e_0(*)$. For the successor ordinals we have

$$r_\nu(e_{\nu+1}) = r_\nu(g_{\nu+1}(e_\nu(*))) = f_\nu(e_\nu(*)) = e_\nu(*)$$

and

$$f_{\nu+1}(e_{\nu+1}(*)) = g_{\nu+1}(r_\nu(e_{\nu+1}(*))) = g_{\nu+1}(e_\nu(*)) = e_{\nu+1}(*).$$

and for limit ordinals we have that e_α restricts to previous ones by construction. To show that $f_\alpha(e_\alpha(*)) = e_\alpha(*)$ we show that $f_\alpha(e_\alpha(*))$ restricts to the same elements. Let $\nu < \alpha$.

$$f_\alpha(e_\alpha(*))|_\nu = f_\nu\left(e_\alpha|_\nu\right) = f_\nu(e_\nu) = e_\nu$$

hence $f_\alpha(e_\alpha(*)) = e_\alpha$ by uniqueness of amalgamations.

Thus e defines a natural transformation $1 \to X$ with $e;f = e$. To see that it is unique observe that if $e' : 1 \to X$ is such that $e';f = e'$ we have

$$e'_{\nu+1}(*) = f_{\nu+1}(e'_{\nu+1}(*)) = g_{\nu+1}(r_\nu(e'_{\nu+1}(*))) = g_{\nu+1}(e'_\nu(*))$$

and the values at limit ordinals α are uniquely determined by the sheaf condition. Thus $e' = e$. QED

[6] More precisely, we define at stage ν a triple of an element $e_\nu \in X(\nu)$ and proofs that it is a fixed point of f_ν and that it restricts to the previous ones.

$\triangleright$ **modality** $\blacktriangleright$ is a modality on types. It is easy to see that it preserves all limits, since they are constructed pointwise. In particular, it preserves monos, therefore subobjects. Thus we can define an operation on subobject lattices, which we call $\triangleright$. Given $m : A \leq X$ the subobject $\triangleright m : \triangleright A \leq X$ is defined via pullback along next^X as in the following diagram

$$\begin{tikzcd}
\triangleright_X A \arrow[r, dotted] \arrow[d, dotted, "\triangleright m"] \arrow[dr, phantom, "\lrcorner", very near start] & \blacktriangleright A \arrow[d, "\blacktriangleright m"] \\
X \arrow[r, "\mathrm{next}^X"] & \blacktriangleright X.
\end{tikzcd}$$

Since it is defined via pullback it is easy to see that this operation is natural in X, i.e. for any morphism $\varphi : Y \to X$ we have $\varphi^* \circ \triangleright_X = \triangleright_Y \circ \varphi^*$. By the usual Yoneda argument we thus get a morphism $\triangleright : \Omega \to \Omega$ such that given $A \leq X$ with the characteristic map χ_A, the characteristic map of $\triangleright_X A$ is $\chi_A;\triangleright$.

We can compute $\triangleright : \Omega \to \Omega$ more concretely as

$$\triangleright_\nu(\beta) = \min(\beta + 1, \nu).$$

Kripke-Joyal semantics

Let X be a sheaf and φ, ψ formulas in the internal language with a free variable of type X, i.e. $x : X \vdash \varphi : \Omega$ and $x : X \vdash \psi : \Omega$.

Let $\nu \leq \omega_1$ and $\xi \in X(\nu)$. Then

- $\nu \Vdash \bot$ iff $\nu = 0$;
- $\nu \Vdash \top$ always;
- $\nu \Vdash \varphi(t)(\xi)$ iff $[\![\varphi]\!]_\nu([\![t]\!]_\nu(\xi)) = \nu$, for a predicate symbol φ on X;
- $\nu \Vdash \varphi(\xi) \wedge \psi(\xi)$ iff $\nu \Vdash \varphi(\xi)$ and $\nu \Vdash \psi(\xi)$;
- $\nu \Vdash \varphi(\xi) \vee \psi(\xi)$ iff $\nu \Vdash \varphi(\xi)$ or $\nu \Vdash \psi(\xi)$;
- $\nu \Vdash \varphi(\xi) \to \psi(\xi)$ iff for all $\beta \leq \nu, \beta \Vdash \varphi(\xi|_\beta)$ implies $\beta \Vdash \psi(\xi|_\beta)$;
- $\nu \Vdash \neg\varphi(\xi)$ iff for all $\beta \leq \nu, \beta \Vdash \varphi(\xi|_\beta)$ implies $\beta = 0$.

If further we have $x : X, y : Y \vdash \varphi : \Omega$, $\nu \leq \omega_1$ and $\xi \in X(\nu)$ then

$$\nu \Vdash \exists y, \varphi(\xi, y) \text{ iff there exists } \xi' \in Y(\nu), \nu \Vdash \varphi(\xi, \xi')$$

if ν is a successor ordinal and

$$\nu \Vdash \exists y, \varphi(\xi, y) \text{ iff for all } \beta < \nu \text{ there exists } \xi_\beta \in Y(\beta), \beta \Vdash \varphi(\xi|_\beta, \xi_\beta)$$

if ν is a limit ordinal. Finally for the universal quantification we have the usual

$$\nu \Vdash \forall y, \varphi(\xi, y) \text{ iff for all } \beta \leq \nu, \text{ for all } \xi_\beta \in Y(\beta), \beta \Vdash \varphi(\xi|_\beta, \xi_\beta)$$

These are standard, cf. [66, Theorem VI.7.1]. The case for disjunction is perhaps not immediately clear since using the cited theorem literally would give us two cases

$$\nu \Vdash \varphi(\xi) \vee \psi(\xi) \text{ iff } \nu \Vdash \varphi(\xi) \text{ or } \nu \Vdash \psi(\xi)$$

if ν is a successor ordinal (since there is only one cover in such a case) and

$$\nu \Vdash \varphi(\xi) \vee \psi(\xi) \text{ iff for all } \beta < \nu, \beta \Vdash \varphi(\xi|_\beta) \text{ or } \beta \Vdash \psi(\xi|_\beta)$$

if ν is a limit ordinal. However because the order on ω_1 is linear in the latter case by truth preservation we must have either

$$\nu \Vdash \varphi(\xi) \vee \psi(\xi) \text{ iff for all } \beta < \nu, \beta \Vdash \varphi(\xi|_\beta)$$

or

$$\nu \Vdash \varphi(\xi) \vee \psi(\xi) \text{ iff for all } \beta < \nu, \beta \Vdash \psi(\xi|_\beta).$$

By the local character property (see below) we then have $\nu \Vdash \varphi(\xi)$ and $\nu \Vdash \psi(\xi)$, respectively.

This can be generalised for any finite disjunction, but infinite disjunctions and existentials cannot be so simplified.

The semantics of $\triangleright$ is as follows. Let $\varphi : \Omega^X$. Then

$$\nu \Vdash \triangleright\varphi(\alpha) \text{ iff for all } \beta < \nu, \beta \Vdash \varphi(\alpha|_\beta) \tag{3.1}$$

For successor ordinals $\nu = \nu' + 1$ this reduces to

$$\nu \Vdash \triangleright\varphi(\alpha) \text{ iff } \nu' \Vdash \varphi(\alpha|_{\nu'})$$

which is easy to check from the definition of $\triangleright$ above. For limit ordinals the characterisation in (3.1) follows from the local character property (see below).

Note that $0 \Vdash \varphi$ for any φ and $1 \Vdash \triangleright\varphi$ for any φ.

Local character By the local character property [66, VI.7 (page 316)] we have for any limit ordinal ξ

$$\xi \Vdash \varphi(\alpha) \text{ iff for all } \beta < \xi, \beta \Vdash \varphi(\alpha|_\beta).$$

Therefore to prove validity of a formula at a limit ordinal, it suffices to do so on all strictly smaller ordinals. This is one of the reasons to use sheaves instead of presheaves. To transfer *local* properties to global ones.

Note that from the characterisation in (3.1) it is immediate that $p \rightarrow \triangleright p$ for any p.

Proposition 3.A.17. *$\triangleright$ satisfies the Löb induction rule $\forall p : \Omega, (\triangleright p \to p) \to p$ and also satisfies the following properties.*

- *$\triangleright$ preserves $\wedge$, $\vee$, $\top$ and $\to$,*
- *$\triangleright(\forall x : X, \varphi(x)) \to \forall x : X, \triangleright\varphi(x)$,*
- *$\exists x : X, \triangleright\varphi(x) \to \triangleright(\exists x : X, \varphi(x))$.*

$\Diamond$

Proof. The proof of the Löb induction rule is by Kripke-Joyal semantics. More precisely, by induction on ν we prove that for all $\xi \in \Omega(\nu)$, for all $\beta \le \nu$, $\beta \Vdash (\triangleright p \to p)(\xi|_\beta)$ implies $\xi|_\beta = \beta$.

- If $\nu = 0$ there is nothing to prove.
- If $\nu = \nu' + 1$ we consider two cases.
  - If $\beta \le \nu'$ the result follows directly from the induction hypothesis.
  - If $\beta = \nu$ assume $\beta \Vdash (\triangleright p \to p)(\xi)$. We need to show $\xi = \beta$. By induction hypothesis and monotonicity $\xi|_{\nu'} = \nu'$, or in other words, $\nu' \Vdash \xi|_{\nu'}$. Since by assumption $\beta \Vdash \triangleright\xi$ implies $\beta \Vdash \xi$ we get $\beta = \xi$.

If ν is a limit ordinal the result follows from the local character property.

$\triangleright$ **preserves** $\wedge$ Given two subobjects $A, B \le X$ with characteristic maps χ_A, χ_B, the characteristic map of $A \wedge B$ is given by $\langle \chi_A, \chi_B \rangle ; \wedge$, where $\wedge : \Omega \times \Omega \to \Omega$ is given by $\wedge_\nu(\beta, \beta') = \min\{\beta, \beta'\}$. It is thus easy to see that $\wedge$ commutes with $\triangleright$, i.e. that $\triangleright \times \triangleright ; \wedge = \wedge ; \triangleright$.

$\triangleright$ **preserves** $\to$ Similarly to $\wedge$, $\to$ is given on subobjects by composition with $\to : \Omega \times \Omega \to \Omega$ which is given as

$$\to_\nu (\beta, \beta') = \begin{cases} \nu & \text{if } \beta' \ge \beta \\ \beta' & \text{otherwise} \end{cases}$$

and since $\triangleright$ preserves order, i.e. $\beta' \ge \beta$ implies $\triangleright_\nu(\beta') \ge \triangleright_\nu(\beta)$, it is easy to see that $\to ; \triangleright = \triangleright \times \triangleright ; \to$.

$\triangleright$ **preserves** $\vee$ $\vee$ is given on subobjects by composition with $\vee : \Omega \times \Omega \to \Omega$ which is given by

$$\vee_\nu(\beta, \beta') = \max\{\beta, \beta'\}.$$

Using this it is easy to see that $\triangleright \times \triangleright ; \vee = \vee ; \triangleright$.

Universal quantifier Take $\nu \leq \omega_1$ and assume $\nu \Vdash \triangleright(\forall x : X, \varphi(x))$. Take $\beta \leq \nu$ and $x_\beta \in X(\beta)$. We are to show $\beta \Vdash \triangleright\varphi(x_\beta)$. Let $\beta' < \beta$. We are to show $\beta' \Vdash \varphi(x_\beta|_{\beta'})$. Since $\beta' < \nu$ we have that $\beta' \Vdash \forall x : X, \varphi(x)$. The conclusion follows.

Existential quantifier Take $\nu \leq \omega_1$ and assume $\nu \Vdash \exists x : X, \triangleright\varphi(x)$. We are to show $\nu \Vdash \triangleright(\exists x : X, \varphi(x))$.

Let $\beta < \nu$. We have to show $\beta \Vdash \exists x : X, \varphi(x)$.

Suppose ν is a successor ordinal. Using the assumption we have that there exists $x_\nu \in X(\nu)$, such that $\nu \Vdash \triangleright\varphi(x_\nu)$ which implies in particular that $\beta \Vdash \varphi(x_\nu|_\beta)$. Choosing $x_\nu|_\beta \in X(\beta)$ we have that $\beta \Vdash \exists x : X, \varphi(x)$.

Suppose ν is a limit ordinal. Let $\beta < \nu$. Then $\beta + 1 < \nu$ and by assumption there exists a $x_{\beta+1} \in X(\beta + 1)$ such that $\beta \Vdash \varphi\left(x_{\beta+1}|_\beta\right)$. Thus $\beta \Vdash \exists x : X, \varphi(x)$. $\mathfrak{QED}$

Proposition 3.A.18. *If X is total then $(\forall x : X, \triangleright\varphi(x)) \to \triangleright(\forall x : X, \varphi(x))$ holds.* ◊

Proof. Take $\nu < \omega_1$ and assume $\nu \Vdash \forall x : X, \triangleright\varphi(x)$. Take $\beta < \nu$. We are to show $\beta \Vdash \forall x : X, \varphi(x)$. Take $\xi \leq \beta$ and $x_\xi \in X(\xi)$. Since X is total there is $x_{\xi+1} \in X(\xi + 1)$ that restricts to x_ξ. Since $\xi + 1 \leq \nu$ we have $\xi + 1 \Vdash \triangleright\varphi(x_{\xi+1})$, thus $\xi \Vdash \varphi(x_\xi)$. $\mathfrak{QED}$

Remark 3.A.19. It is *not* the case that if X is total and inhabited then

$$\triangleright(\exists x : X, \varphi(x)) \to \exists x : X, \triangleright\varphi(x)$$

holds, as is the case in the logic of the topos of trees [22]. In fact, even if X is a constant sheaf this does not necessarily hold. ♦

Proof. Let $X = \Delta(\mathbb{N})$. The constant sheaf of natural numbers is

$$\begin{tikzcd}
1 & \mathbb{N} \arrow[l] & \mathbb{N} \arrow[l, "\mathrm{id}"'] & \mathbb{N} \arrow[l, "\mathrm{id}"'] & \mathbb{N} \arrow[l, "\mathrm{id}"'] & \arrow[l, "\mathrm{id}"']
\end{tikzcd}$$

Let $A \leq X$ be the subobject defined as follows

$$\begin{aligned}
A(0) &= 1 \\
A(n) &= \{k \mid n \leq k < \omega\} && \text{for } 0 < n < \omega \\
A(\nu) &= \emptyset && \text{for } \omega \leq \nu < \omega_1
\end{aligned}$$

A is a subsheaf of X. Let φ be the characteristic map of A. If

$$\triangleright(\exists x : X, \varphi(x)) \to \exists x : X, \triangleright\varphi(x)$$

were to hold it would hold at stage $\omega + 1$. But $\omega + 1 \Vdash \triangleright(\exists x : X, \varphi(x))$ since for all $n < \omega$, exists $n \in A(n)$, i.e. we can choose $x_n = n$ and then $n \Vdash \varphi(x_n)$.

But notice that $A(\omega + 1)$ is empty, hence the right hand side of the implication is false. $\mathfrak{QED}$

Remark 3.A.20. The last remark has implications for fixed points. It seems that the internal Banach fixed point theorem does not hold in the same generality as it does in the topos of trees [22], or rather the proof that works for the topos of trees does not carry over, since it uses the fact that later commutes over existentials for total and inhabited objects.

This state of affairs makes intuitive sense, since [22, Lemma 2.10] does not make intuitive sense for $\mathbf{Sh}(\omega_1)$ since it implies that at any stage we can get a fixed point by a finite iteration, something we would not expect to hold in $\mathbf{Sh}(\omega_1)$. ♦

Guarded recursive predicates

We now show that suitably *internally contractive* functions have unique fixed-points, thus establishing the existence of recursively defined predicates and relations.

Definition 3.A.21. Define

$$\begin{aligned} \mathrm{Inhab}(X) &= \exists x : X.\top \\ \mathrm{Total}(X) &= \forall x : \blacktriangleright X, \exists x' : X, \mathrm{next}^X(x') = x \end{aligned}$$

♦

If $\vdash \mathrm{Inhab}(X)$ then each $X(\nu)$ is non-empty, but this *does not mean* that a global section exists. If $\vdash \mathrm{Total}(X)$ then the restriction maps are surjections (which implies that each stage $X(\nu)$ is inhabited, since X is a sheaf, i.e. $X(1) = 1$).

Proposition 3.A.22 (Internal Banach's fixed point theorem)**.** *The following holds in the internal logic of* $\mathbf{Sh}(\omega_1)$.

$$\mathrm{Total}(X) \to \forall f : X^X, \mathrm{Contr}(f) \to \exists! x : X, f(x) = x$$

◊

Proof. We use the Kripke-Joyal forcing semantics and proceed by induction on ν. If $\nu = 0$ there is nothing to prove. Let $\nu = \nu'+1$ and assume $\nu \Vdash \mathrm{Total}(X)$. Let $f \in X^X(\nu)$, $\beta \leq \nu$ and $\beta \Vdash \mathrm{Contr} f$. We are to show $\beta \Vdash \exists! x : X, f(x) = x$. We define a sequence of elements $e_\xi \in X(\xi)$ for $\xi \leq \beta$ as follows.

$$\begin{aligned} e_0 &= f_0(*) \\ e_{\xi+1} &= f_{\xi+1}(r_\xi^{-1}(e_\xi)) \\ e_\alpha &= \lim_{\xi<\alpha} e_\xi \end{aligned}$$

This requires some explanations. First, r are the restriction maps of X. Second, α is again a limit ordinal and the limit means the unique element of

$X(\alpha)$ that exists by the sheaf condition. Third, since $\xi + 1$ in the definition is smaller or equal to ν we can use the assumption that X is total with the element e_ξ to get some element $r_\xi^{-1}(e_\xi) \in X(\xi+1)$ that restricts to e_ξ.

First we observe that the choice of $r_\xi^{-1}(e_\xi) \in X(\xi+1)$ does not matter. This follows from contractiveness of f since if u, v both restrict to e_ξ then $f_{\xi+1}(u) = f_{\xi+1}(v)$.

Further, we have that $f_\xi(e_\xi) = e_\xi$ and that $r_\xi(e_{\xi+1}) = e_\xi$. For $\xi = 0$ this is obvious. For successor ordinals we have

$$f_{\xi+1}(e_{\xi+1}) = f_{\xi+1}(f_{\xi+1}(r_\xi^{-1}(e_\xi)))$$

Since f is contractive it suffices to show $r_\xi(e_{\xi+1}) = r_\xi(f_{\xi+1}(r_\xi^{-1}(e_\xi))))$ and this follows by naturality of f and the induction hypothesis.

And

$$r_\xi(e_{\xi+1}) = r_\xi(f_{\xi+1}(r_\xi^{-1}(e_\xi))) = f_\xi(e_\xi) = e_\xi.$$

For limit ordinals the restrictions are automatic. To see that $f_\alpha(e_\alpha) = e_\alpha$ we show that $f_\alpha(e_\alpha)$ restricts to the same elements and this is immediate by naturality of the family f.

Thus there exist a $x_\beta \in X(\beta)$ such that $f_\beta(x_\beta) = x_\beta$. Uniqueness is shown similarly to uniqueness of external fixed points in Proposition 3.A.16. 𝔔𝔈𝔇

Predicates defined as least and greatest fixed points

Inductively and coinductively defined predicates are constructed as least and greatest fixed points of suitable maps of type $\mathcal{P}(X) \to \mathcal{P}(X)$ for a suitable X, giving a predicate on X. We will show that these predicates are $\neg\neg$ closed provided the defining functions are sufficiently tame.

Greatest fixed points

Proposition 3.A.23. *Suppose φ is a predicate on $\Gamma, x : X$ and suppose that the sequent*

$$\Gamma \mid \emptyset \vdash \forall x : X, \varphi \leftrightarrow F(\varphi)$$

holds where $F(\varphi)$ is a well-formed formula in context $\Gamma, x : X$ consisting of $\forall, \to, \wedge, \vee, \top, \bot$ and existential quantification over total types and using only $\neg\neg$-closed predicate and relation symbols. Then

$$\Gamma \mid \emptyset \vdash \forall x : X, \neg\neg\varphi \leftrightarrow F(\neg\neg\varphi)$$

also holds. In other words, if φ is a fixed point of F then so is $\neg\neg\varphi$. ◊

Proof. From

$$\Gamma \mid \emptyset \vdash \forall x : X, \varphi \leftrightarrow F(\varphi)$$

we immediately get

$$\Gamma \mid \emptyset \vdash \neg\neg(\forall x : X, \varphi \leftrightarrow F(\varphi))$$

and using the fact that $\neg\neg$ commutes with implication, universal quantifiers and conjuction we get

$$\Gamma \mid \emptyset \vdash \forall x : X, \neg\neg\varphi \leftrightarrow \neg\neg F(\varphi)$$

and now the restrictions on F guarantee that $\neg\neg F(\varphi) = F(\neg\neg\varphi)$, concluding the proof. $\mathfrak{QED}$

As a consequence, suppose we define a predicate φ on X coinductively as the greatest fixed point of some $\mathcal{F} : \mathcal{P}(X) \to \mathcal{P}(X)$. Since $\varphi \to \neg\neg\varphi$ and if φ is a fixed point then also $\neg\neg$ is, we have that φ is $\neg\neg$-closed.

Least fixed points

Proposition 3.A.24. *Suppose φ is a predicate on $\Gamma, x : X$ and suppose that the sequent*

$$\Gamma \mid \emptyset \vdash \forall x : X, F(\varphi) \to \varphi$$

holds where $F(\varphi)$ is a well-formed formula in context $\Gamma, x : X$ consisting of $\forall, \to, \wedge, \vee, \top, \bot$ and existential quantification over total types and using only $\neg\neg$-closed predicate and relation symbols where in addition φ only occurs positively, i.e. F preserves implication.

Then

$$\Gamma \mid \emptyset \vdash \forall x : X, F(\Box\varphi) \to \Box\varphi$$

also holds. $\Diamond$

Proof. Since $\Box$ does not behave as well as $\neg\neg$ we have to prove this using the model. Our assumption gives us that $[\![F(\varphi)]\!] \leq [\![\varphi]\!]$ in the fibre over $\Gamma, x : X$ and we wish to show $[\![F(\Box\varphi)]\!] \leq [\![\Box\varphi]\!] = \Box[\![\varphi]\!]$. Since $\Box$ is the right adjoint to $\neg\neg$ we have

$$[\![F(\Box\varphi)]\!] \leq \Box[\![\varphi]\!] \leftrightarrow \neg\neg[\![F(\Box\varphi)]\!] = [\![\neg\neg F(\Box\varphi)]\!] \leq [\![\varphi]\!]$$

and since the assumptions on F guarantee that $\neg\neg F(\varphi) = F(\neg\neg\varphi)$ we have

$$\leftrightarrow [\![F(\neg\neg\Box\varphi)]\!] \leq [\![\varphi]\!]$$

which by properties of $\neg\neg$ and $\Box$ gives us

$$\leftrightarrow [\![F(\Box\varphi)]\!] \leq [\![\varphi]\!]$$

which holds by the fact that $\Box\varphi \to \varphi$ and the assumption on monotonicity of F. $\mathfrak{QED}$

It is an easy fact to show that least fixed points of monotone functions are exactly their least prefixed points. Using Proposition 3.A.24 and the facts that $\Box$ is deflationary and for any φ, $\Box\varphi$ is $\neg\neg$-closed we can conclude that inductively defined predicates are constant, provided they can be defined using a suitable subset of the logic.

Consequence for the external interpretation Since Π_1 is a logical functor it preserves structure of an elementary topos. More precisely, $\Pi_1 : \mathbf{Sh}(\omega_1) \to \mathbf{Set}$ gives rise to the morphism of higher-order fibrations

$$\begin{tikzcd}
\mathbf{Sub}_{\mathbf{Sh}(\omega_1)} \arrow[r] \arrow[d, "\mathbf{cod}"] & \mathbf{Sub}_{\mathbf{Set}} \arrow[d, "\mathbf{cod}"] \\
\mathbf{Sh}(\omega_1) \arrow[r, "\Pi_1"] & \mathbf{Set}
\end{tikzcd}$$

that preserves the structure of a higher-order fibration (i.e. constructs to model higher-order logic). In particular this means it preserves validity of formulas in the internal language that use only the usual connectives of higher-order logic (i.e. everything but $\triangleright$ and $\square$).

Thus given a formula φ of higher-order logic (i.e. no $\square$ and no $\triangleright$) in some context Γ and choosing some interpretations of relational symbols, the denotation $\Pi_1(\llbracket\varphi\rrbracket)$ as a subobject of $\Pi_1(\llbracket\Gamma\rrbracket)$ is the same as the denotation of φ in **Set**, replacing the interpretations of relational symbols and types by their values at stage 1.

For instance if

$$x : \Delta(a), y : \Delta(b) \mid \emptyset \vdash \forall z : \mathcal{P}(\Delta(a)), \exists w : \mathcal{P}(\Delta(b)), P(x,w) \to Q(z,y)$$

holds in the logic of $\mathbf{Sh}(\omega_1)$ with P and Q being interpreted as some $\Delta(p)$ and $\Delta(q)$, respectively, then

$$x : a, y : b \mid \emptyset \vdash \forall z : \mathcal{P}(a), \exists w : \mathcal{P}(b), P(x,w) \to Q(z,y)$$

holds with P being interpreted as p and Q as q (we used the that $\Pi_1 \circ \Delta = \mathrm{id}_{\mathbf{Set}}$).

Combining this with the construction of fixed points we get that least and greatest fixed points constructed internally are mapped to external least and greatest fixed points.

Proposition 3.A.25. *Let φ be a predicate symbol on $\Delta(a)$ and let $F(\varphi)$ be a formula in context $x : \Delta(a)$. Suppose $F(\varphi)$ is monotone in φ and suppose it involves only quantifiers over constant sets and constant predicate symbols. If*

$$\begin{aligned}\emptyset \mid \emptyset \vdash &(\forall x : \Delta(a), \varphi(x) \leftrightarrow F(\varphi)) \\ &\wedge (\forall \psi : \mathcal{P}(\Delta(a)), (\forall x : \Delta(a), \psi(x) \leftrightarrow F(\psi)) \to (\forall x : \Delta(a), \psi(x) \to \varphi(x)))\end{aligned} \tag{3.2}$$

holds in the logic of $\mathbf{Sh}(\omega_1)$ then

$$\begin{aligned}\emptyset \mid \emptyset \vdash &(\forall x : a, \varphi(x) \leftrightarrow \Pi_1(F)(\varphi)) \\ &\wedge (\forall \psi : \mathcal{P}(a), (\forall x : a, \psi(x) \leftrightarrow \Pi_1(F)(\psi)) \to (\forall x : a, \psi(x) \to \varphi(x)))\end{aligned} \tag{3.3}$$

holds in the logic of **Set** *with interpretations of relational symbols being Π_1-images of their chosen interpretations in* $\mathbf{Sh}(\omega_1)$*. Here $\Pi_1(F)$ means replacing occurrences of $\Delta(b)$ in quantifiers with b.* ◊

Remark 3.A.26. Note that if φ is a predicate on $\Delta(a)$ (in the empty context) then $\varphi(x)$ is a formula in context $x : \Delta(a)$. These formulas characterise φ as the greatest predicate (or subset) satisfying F. We can state and prove an analogous property for least fixed points. ♦

From Proposition 3.A.23 we have that if (3.2) holds then $\varphi = \neg\neg \circ \varphi$. Thus $x : \Delta(a) \mid \emptyset \vdash \varphi(x)$ corresponds to a constant subobject of $\Delta(a)$, say $\Delta\left(\underline{\varphi}\right)$. Formula (3.3) state that $\underline{\varphi}$ is the greatest fixed point of "the same" formula.

Transitive closure

We prove that given a $\neg\neg$ closed relation R on a set with decidable equality its reflexive and transitive closure R^* is also $\neg\neg$-closed.

Lemma 3.A.27. *Let $R \subseteq X \times X$ be a relation on a decidable total type X. If R is $\neg\neg$-closed (meaning for all $x, x' : X$, $(x,x') \in R \leftrightarrow \neg\neg((x,x') \in R)$) then the reflexive transitive closure R^* is also $\neg\neg$-closed.* ◊

Proof. By construction $R^* = \bigvee_{n \in \mathbb{N}} R^n$ where the sequence $\{R^n\}_{n \in \mathbb{N}}$ is defined by induction as

$$\begin{aligned} R^0 &= \{(x,x') \mid x = x'\} \\ R^{n+1} &= \{(x,x'') \mid \exists x' : X, (x,x') \in R \wedge (x',x'') \in R^n\} \end{aligned}$$

Thus it suffices to show that all R^n are $\neg\neg$-closed and we do this by induction. R^0 is $\neg\neg$-closed since X is assumed to have decidable equality. Assuming R^n is $\neg\neg$-closed we have

$$\neg\neg((x,x') \in R^{n+1}) \leftrightarrow \neg\neg\exists x' : X, (x,x') \in R \wedge (x',x'') \in R^n$$

and since X is assumed to be total we have by Lemma 3.A.8 and the fact that $\neg\neg$ commutes with conjunction that

$$\leftrightarrow \exists x' : X, \neg\neg((x,x') \in R) \wedge \neg\neg((x',x'') \in R^n)$$

which is by assumption on R and the induction hypothesis equivalent to

$$\leftrightarrow \exists x' : X, (x,x') \in R \wedge (x',x'') \in R^n$$

which is by definition of R^{n+1} equivalent to

$$\leftrightarrow (x,x') \in R^{n+1}.$$

𝔔𝔈𝔇

In particular, all constant sets are total and decidable and so the lemma applies.

3.B The Model of a Language with Countable Choice

In this section we introduce $\mathbf{F}^{\mu,?}$, a call-by-value functional language akin to System F, i.e., with impredicative polymorphism, existential and general recursive types, extended with a countable choice expression ?. We work informally in the internal logic of $\mathbf{Sh}(\omega_1)$ outlined above except where explicitly stated.

Syntax and operational semantics

The syntax of types, terms and values is defined in Figure 3.4. These should be understood as initial algebras of suitable polynomial functors.

$$
\begin{aligned}
\tau &::= \alpha \mid \mathbf{1} \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \to \tau_2 \mid \mu\alpha.\tau \mid \forall\alpha.\tau \mid \exists\alpha.\tau \\
v &::= x \mid \langle\rangle \mid \langle v_1, v_2\rangle \mid \lambda x.e \mid \texttt{inl}\ v \mid \texttt{inr}\ v \mid \Lambda.e \mid \texttt{pack}\, v \\
e &::= x \mid \langle\rangle \mid \langle e_1, e_2\rangle \mid \lambda x.e \mid \texttt{inl}\ e \mid \texttt{inr}\ e \mid \Lambda.e \mid \texttt{pack}\, e \\
&\quad \mid ? \mid \texttt{proj}_i\, e \mid e_1\, e_2 \mid \texttt{case}\,(e, x_1.e_1, x_2.e_2) \mid e[] \mid \texttt{unpack}\ e_1\ \texttt{as}\ x\ \texttt{in}\ e_2 \\
&\quad \mid \texttt{unfold}\, e \mid \texttt{fold}\, e \\
E &::= - \mid \langle E, e\rangle \mid \langle v, E\rangle \mid \texttt{inl}\ E \mid \texttt{inr}\ E \mid \texttt{pack}\, E \\
&\quad \mid \texttt{proj}_i\, E \mid E\, e \mid v\, E \mid \texttt{case}\,(E, x_1.e_1, x_2.e_2) \mid E[] \mid \texttt{unpack}\ E\ \texttt{as}\ x\ \texttt{in}\ e \\
&\quad \mid \texttt{unfold}\, E \mid \texttt{fold}\, E \\
C &::= - \mid \langle C, e\rangle \mid \langle e, C\rangle \mid \lambda x.C \mid \texttt{inl}\ C \mid \texttt{inr}\ C \mid \Lambda.C \mid \texttt{pack}\, C \\
&\quad \mid \texttt{proj}_i\, C \mid C\, e_2 \mid e\, C \mid \texttt{case}\,(C, x_1.e_1, x_2.e_2) \mid \texttt{case}\,(e, x_1.C, x_2.e_2) \\
&\quad \mid \texttt{case}\,(e, x_1.e_1, x_2.C) \mid C[] \mid \texttt{unpack}\ C\ \texttt{as}\ x\ \texttt{in}\ e \mid \texttt{unpack}\ e\ \texttt{as}\ x\ \texttt{in}\ C \\
&\quad \mid \texttt{unfold}\, C \mid \texttt{fold}\, C
\end{aligned}
$$

Figure 3.4: Types, terms and evaluation contexts

The types include polymorphic, existential and recursive types. The corresponding terms are standard, apart from the countable choice expression ?. We assume disjoint, countably infinite sets of *type variables*, ranged over by α, and *term variables*, ranged over by x. The free type variables of types and terms, $ftv(\tau)$ and $ftv(e)$, and free term variables $fv(e)$, are defined in the usual way. The notation $(\cdot)[\vec{\tau}/\vec{\alpha}]$ denotes the simultaneous capture-avoiding substitution of types $\vec{\tau}$ for the free type variables $\vec{\alpha}$ in types and terms; similarly, $e[\vec{v}/\vec{x}]$ denotes simultaneous capture-avoiding substitution of values $\vec{v}$ for the free term variables $\vec{x}$ in e.

Composition of evaluation contexts Evaluation contexts can be composed. Given two evaluation contexts E, E' we define $E \circ E'$ by induction on E as follows

$$\begin{aligned}
[\,] \circ E' &= E' \\
\langle E, e\rangle \circ E' &= \langle E \circ E', e\rangle \\
\langle v, E\rangle \circ E' &= \langle v, E \circ E'\rangle \\
(\texttt{proj}_i E) \circ E' &= \texttt{proj}_i E \circ E' \\
(\texttt{inl}\, E) \circ E' &= \texttt{inl}\,(E \circ E') \\
(\texttt{inr}\, E) \circ E' &= \texttt{inr}\,(E \circ E') \\
(\texttt{fold}\, E) \circ E' &= \texttt{fold}\,(E \circ E') \\
(\texttt{unfold}\, E) \circ E' &= \texttt{unfold}\,(E \circ E') \\
(\texttt{pack}\, E) \circ E' &= \texttt{pack}\,(E \circ E') \\
(\texttt{unpack}\ E\ \texttt{as}\ x\ \texttt{in}\ e) \circ E' &= \texttt{unpack}\ (E \circ E')\ \texttt{as}\ x\ \texttt{in}\ e \\
(\texttt{case}\,(E, x_1.e_1, x_2.e_2)) \circ E' &= \texttt{case}\,(E \circ E', x_1.e_1, x_2.e_2) \\
(\texttt{case}\,(E, x_1.e_1, x_2.e_2)) \circ E' &= \texttt{case}\,(E \circ E', x_1.e_1, x_2.e_2) \\
(E\, e) \circ E' &= (E \circ E')\, e \\
(v\, E) \circ E' &= v\,(E \circ E').
\end{aligned}$$

Lemma 3.B.1. *For any pair of evaluation contexts E, E' and an expression e,*

$$E[E'[e]] = (E \circ E')[e]$$

◊

Types We define the type of natural numbers $\texttt{nat}$ as $\texttt{nat} = \mu\alpha.1+\alpha$ and the corresponding numerals as $\underline{0} = \texttt{fold}(\texttt{inl}\,\langle\rangle)$ and $\underline{n+1} = \texttt{fold}(\texttt{inr}\,\underline{n})$ by induction on n.

The type system is defined in Figure 3.6. The judgement $\Delta \vdash \tau$ is defined in Figure 3.5. The judgements are mostly standard, apart from the typing of ?.

$$\frac{\alpha \in \Delta}{\Delta \vdash \alpha} \qquad \Delta \vdash \mathbf{1}$$

$$\frac{\Delta \vdash \tau_1 \qquad \Delta \vdash \tau_2}{\Delta \vdash \tau_1 \times \tau_2} \qquad \frac{\Delta \vdash \tau_1 \qquad \Delta \vdash \tau_2}{\Delta \vdash \tau_1 + \tau_2} \qquad \frac{\Delta \vdash \tau_1 \qquad \Delta \vdash \tau_2}{\Delta \vdash \tau_1 \to \tau_2}$$

$$\frac{\Delta, \alpha \vdash \tau}{\Delta \vdash \exists\alpha.\tau} \qquad \frac{\Delta, \alpha \vdash \tau}{\Delta \vdash \forall\alpha.\tau} \qquad \frac{\Delta, \alpha \vdash \tau}{\Delta \vdash \mu\alpha.\tau}$$

Figure 3.5: Well-formed types. The judgement $\Delta \vdash \tau$ expresses $\mathit{ftv}(\tau) \subseteq \Delta$.

$$\frac{x{:}\tau \in \Gamma \qquad \Delta \vdash \Gamma}{\Delta;\Gamma \vdash x : \tau} \qquad \frac{\Delta \vdash \Gamma}{\Delta;\Gamma \vdash \langle\rangle : \mathbf{1}} \qquad \frac{\Delta;\Gamma \vdash e_1 : \tau_1 \qquad \Delta;\Gamma \vdash e_2 : \tau_2}{\Delta;\Gamma \vdash \langle e_1, e_2\rangle : \tau_1 \times \tau_2}$$

$$\frac{\Delta;\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Delta;\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2} \qquad \frac{\Delta;\Gamma \vdash e : \tau_1 \qquad \Delta \vdash \tau_2}{\Delta;\Gamma \vdash \mathtt{inl}\, e : \tau_1 + \tau_2} \qquad \frac{\Delta;\Gamma \vdash e : \tau_2 \qquad \Delta \vdash \tau_1}{\Delta;\Gamma \vdash \mathtt{inr}\, e : \tau_1 + \tau_2}$$

$$\frac{\Delta;\Gamma, x_1{:}\tau_1 \vdash e_1 : \tau \qquad \Delta;\Gamma, x_2{:}\tau_2 \vdash e_2 : \tau \qquad \Delta;\Gamma \vdash e : \tau_1 + \tau_2}{\Delta\ \Gamma \vdash \mathtt{case}\,(e, x_1.e_1, x_2.e_2) : \tau}$$

$$\frac{\Delta, \alpha;\Gamma \vdash e : \tau}{\Delta;\Gamma \vdash \Lambda.e : \forall \alpha.\tau} \qquad \frac{\Delta;\Gamma \vdash e : \tau_1 \times \tau_2}{\Delta;\Gamma \vdash \mathtt{proj}_i\, e : \tau_i} \qquad \frac{\Delta;\Gamma \vdash e : \tau' \to \tau \qquad \Delta;\Gamma \vdash e' : \tau'}{\Delta;\Gamma \vdash e\, e' : \tau}$$

$$\frac{\Delta \vdash \tau_1 \qquad \Delta;\Gamma \vdash e : \tau[\tau_1/\alpha]}{\Delta;\Gamma \vdash \mathtt{pack}\, e : \exists \alpha.\tau}$$

$$\frac{\Delta;\Gamma \vdash e : \exists \alpha.\tau_1 \qquad \Delta \vdash \tau \qquad \Delta, \alpha;\Gamma, x : \tau_1 \vdash e' : \tau}{\Delta;\Gamma \vdash \mathtt{unpack}\ e\ \mathtt{as}\ x\ \mathtt{in}\ e' : \tau}$$

$$\frac{\Delta;\Gamma \vdash e : \mu\alpha.\tau}{\Delta;\Gamma \vdash \mathtt{unfold}\, e : \tau[\mu\alpha.\tau/\alpha]} \qquad \frac{\Delta;\Gamma \vdash e : \tau[\mu\alpha.\tau/\alpha]}{\Delta;\Gamma \vdash \mathtt{fold}\, e : \mu\alpha.\tau}$$

$$\frac{\Delta;\Gamma \vdash e : \forall \alpha.\tau \qquad \Delta \vdash \tau'}{\Delta;\Gamma \vdash e[\,] : \tau[\tau'/\alpha]} \qquad \frac{\Delta \vdash \Gamma}{\Delta;\Gamma \vdash ? : \mathtt{nat}}$$

Figure 3.6: Typing of terms, where $\Gamma ::= \emptyset \mid \Gamma, x{:}\tau$ and $\Delta ::= \emptyset \mid \Delta, \alpha$. The notation $\Delta \vdash \tau$ means that $\mathit{ftv}(\tau) \subseteq \Delta$.

We write $\mathbf{Type}(\Delta)$ for the set of types well-formed in context Δ and **Type** for the set of closed types τ, i.e., where $\mathit{ftv}(\tau) = \emptyset$. We write $\mathbf{Val}\,(\tau)$ and $\mathbf{Tm}\,(\tau)$ for the sets of *closed* values and terms of type τ, respectively. We write **Val** and **Tm** for the set of all closed values and closed terms, respectively. $\mathbf{Stk}\,(\tau)$ denotes the set of τ-accepting evaluation contexts, i.e. evaluation contexts E, such that given any closed term e of type τ, $E[e]$ is a typeable term. **Stk** denotes the set of all evaluation contexts.

For a typing context $\Gamma = x_1{:}\tau_1, \ldots, x_n{:}\tau_n$ with $\tau_1, \ldots, \tau_n \in \mathbf{Type}$, let

$$\mathbf{Subst}(\Gamma) = \{\gamma \in \mathbf{Val}^{\vec{x}} \mid \forall 1 \le i \le n.\ \gamma(x_i) \in \mathbf{Val}\,(\tau_i)\}$$

denote the set of type-respecting value substitutions. In particular, if $\Delta;\Gamma \vdash e : \tau$ then $\varnothing;\varnothing \vdash e\gamma : \tau\delta$ for any $\delta \in \mathbf{Type}^{\Delta}$ and $\gamma \in \mathbf{Subst}(\Gamma\delta)$, and the type system satisfies standard properties of progress and preservation and a canonical forms lemma.

Basic reductions $\longmapsto$

$$\begin{array}{rlrl}
\mathtt{proj}_i\,\langle v_1, v_2\rangle \longmapsto v_i & & \mathtt{unfold}(\mathtt{fold}\,v) \longmapsto v \\
(\lambda x.e)\,v \longmapsto e[v/x] & & \mathtt{unpack}\ (\mathtt{pack}\,v)\ \mathtt{as}\ x\ \mathtt{in}\ e \longmapsto e[v/x] \\
(\Lambda.e)[\,] \longmapsto e & & \mathtt{case}(\mathtt{inl}\,v, x_1.e_1, x_2.e_2) \longmapsto e_1[v/x_1] \\
? \longmapsto \underline{n} & (n \in \mathbb{N}) & \mathtt{case}(\mathtt{inr}\,v, x_1.e_1, x_2.e_2) \longmapsto e_2[v/x_2]
\end{array}$$

One step reduction relation $\rightsquigarrow$

$$E[e] \rightsquigarrow E[e'] \qquad\qquad \text{if } e \longmapsto e'$$

Figure 3.7: Operational semantics.

The operational semantics of the language is given in Figure 3.7 by a reduction relation $e \rightsquigarrow e'$. In particular, the choice operator ? evaluates nondeterministically to any numeral $\underline{n}$ ($n \in \mathbb{N}$). We use evaluation contexts E, and write $E[e]$ for the term obtained by plugging e into E.

To define the logical relation we need further reduction relations. These will allow us to ignore most reductions in the definition of logical relation.

Let $\rightsquigarrow^*$ be the reflexive transitive closure of $\rightsquigarrow$. We call reductions of the form $\mathtt{unfold}(\mathtt{fold}\,v) \rightsquigarrow v$ *unfold-fold* reductions and reductions of the form $? \rightsquigarrow \underline{n} \quad (n \in \mathbb{N})$ *choice* reductions. We define

- $e \overset{p}{\rightsquigarrow} e'$ if $e \rightsquigarrow^* e'$ and *none* of the reductions is a choice reduction
- $e \overset{0}{\rightsquigarrow} e'$ if $e \rightsquigarrow^* e'$ and *none* of the reductions is an unfold-fold reduction
- $e \overset{1}{\rightsquigarrow} e'$ if $e \rightsquigarrow^* e'$ and *exactly one* of the reductions is an unfold-fold reduction
- $\overset{p,1}{\rightsquigarrow} = \overset{p}{\rightsquigarrow} \cap \overset{1}{\rightsquigarrow}$
- $\overset{p,0}{\rightsquigarrow} = \overset{p}{\rightsquigarrow} \cap \overset{0}{\rightsquigarrow}$

The $\overset{1}{\rightsquigarrow}$ reduction relation will be used in the stratified definition of divergence, see Section 3.B, and the other reduction relations will be used to state additional properties of the logical relation.

Termination relations

In order to define the $\top\top$ closure we need to define our observations. Recall that previously [23] this was achieved by defining the termination relations $\Downarrow$ and $\downarrow$ and stratified versions of these, $\Downarrow_\beta$ and $\downarrow_n$ for $\beta < \omega_1$ and $n < \omega$. Focusing on $\Downarrow_\beta$, it is defined by induction on β as $e \Downarrow_\beta \leftrightarrow \forall e', e \overset{1}{\rightsquigarrow} e' \rightarrow \exists \nu <$

$\beta, e' \Downarrow_\nu$. It is easy to see that $\Downarrow_\beta \subset \Downarrow_{\beta+1}$ so $\left\{\Downarrow_\beta\right\}_{\beta<\omega_1}$ does not define a subobject of the constant sheaf $\Delta(\mathbf{Tm})$.

But observe that defining $\upmapsto_\beta = \mathbf{Tm} \setminus \Downarrow_\beta$, the predicate $\upmapsto_\beta$ may be defined by induction on β as

$$e \underset{\beta}{\upmapsto} \leftrightarrow \exists e', e \overset{1}{\rightsquigarrow} e' \wedge \forall \nu < \beta, e' \underset{\nu}{\upmapsto}.$$

Using this, we define internally $\upmapsto : \mathcal{P}(\mathbf{Tm})$ as the unique fixed point of $\Psi : \mathcal{P}(\mathbf{Tm}) \to \mathcal{P}(\mathbf{Tm})$ given by

$$\Psi(m) = \left\{ e : \mathbf{Tm} \;\middle|\; \exists e', e \overset{1}{\rightsquigarrow} e' \wedge \triangleright(m(e')) \right\}.$$

This is the stratified definition of *may-divergence* (there is a diverging path). We can also define (internally) non-stratified divergence predicate $\uparrow$ as the greatest fixed-point of $\Phi : \mathcal{P}(\mathbf{Tm}) \to \mathcal{P}(\mathbf{Tm})$ given as

$$\Phi(m) = \left\{ e : \mathbf{Tm} \;\middle|\; \exists e', e \rightsquigarrow e' \wedge m(e') \right\}.$$

Since Φ is monotone the greatest fixed point exists by Knaster-Tarski fixed-point theorem (which holds in any topos). Observe that Ψ is almost the same as $\Phi \circ \triangleright$, apart from using a different reduction relation.

As a consequence of Proposition 3.A.23 we have that may-divergence is a decidable property, i.e., for all terms e we have $e\uparrow \vee \neg(e\uparrow)$.

Lemma 3.B.2. *Let $e, e' \in \mathbf{Tm}$. The following hold in the internal language.*

1. *if $e \overset{p}{\rightsquigarrow} e'$ then $e\uparrow \leftrightarrow e'\uparrow$*

2. *if $e \overset{p,0}{\rightsquigarrow} e'$ then $e\upmapsto \leftrightarrow e'\upmapsto$*

3. *if $e \overset{0}{\rightsquigarrow} e'$ then $(e'\upmapsto) \to e\upmapsto$*

4. *if $e \overset{1}{\rightsquigarrow} e'$ then $\triangleright(e'\upmapsto) \to e\upmapsto$*

◊

Proof. 1. Suppose $e \overset{p}{\rightsquigarrow} e'$. If $e = e'$ there is nothing to do. Otherwise, the crucial property we have is that there exists a unique e'', such that $e \rightsquigarrow e''$ and $e'' \overset{p}{\rightsquigarrow} e'$. Using this property this case follows by induction on the number of steps in $\overset{p}{\rightsquigarrow}$.

2. This property follows from the fact that if $e \overset{p,0}{\rightsquigarrow} e'$ then $e \overset{1}{\rightsquigarrow} e''$ if and only if $e' \overset{1}{\rightsquigarrow} e''$ which is easy to see directly from the definition of these relations.

3. Suppose $e \overset{0}{\leadsto} e'$ and $e' \mapsup$. Then by definition there exists a e'', such that $e' \overset{1}{\leadsto} e''$ and $\triangleright(e'' \mapsup)$. By definition of $\overset{1}{\leadsto}$ we also have $e \overset{1}{\leadsto} e''$ and so $e \mapsup$.

4. This follows directly from the definition of the $\mapsup$ relation.

$\mathfrak{QED}$

Must-contextual and must-CIU preorders

Contexts can be typed as second-order terms, by means of a typing judgement of the form $C : (\Delta \mid \Gamma \Rrightarrow \tau) \hookrightarrow (\Delta' \mid \Gamma' \Rrightarrow \sigma)$, stating that whenever

$$\Delta \mid \Gamma \vdash e : \tau$$

holds, $\Delta' \mid \Gamma' \vdash C[e] : \sigma$ also holds. The typing of contexts can be defined as an inductive relation defined by suitable typing rules, which we omit here since they are standard; see [6]. We write $C : (\Delta \mid \Gamma \Rrightarrow \tau)$ to mean there exists a type σ, such that $C : (\Delta \mid \Gamma \Rrightarrow \tau) \hookrightarrow (\varnothing \mid \varnothing \Rrightarrow \sigma)$ holds.

We define *contextual must-approximation* using the *may-divergence* predicate. This is in contrast with the definition in [23] which uses the must-convergence predicate. However externally, in the model, this definition is precisely the one used in [23].

Definition 3.B.3 (Must-contextual approximation)**.** We define the must-contextual approximation relation as

$$\begin{aligned}\Delta \mid \Gamma \vdash e_1 \lesssim_{\Downarrow}^{ctx} e_2 : \tau \triangleq {} & \Delta \mid \Gamma \vdash e_1 : \tau \wedge \Delta \mid \Gamma \vdash e_2 : \tau \wedge \\ & \forall C, C : (\Delta \mid \Gamma \Rrightarrow \tau) \wedge C[e_2]\uparrow \rightarrow C[e_1]\uparrow\end{aligned}$$

♦

Note the order of terms in the implication: if $C[e_2]$ may-diverges then $C[e_1]$ may-diverges. This is the contrapositive of the usual definition which states that if $C[e_1]$ must-converges then $C[e_2]$ must-converges. Must-contextual approximation defined explicitly using contexts can be shown to be the largest compatible adequate and transitive relation, so it coincides with contextual approximation defined in [23].

In practice it is difficult to work with contextual approximation directly. An alternative characterisation of the contextual approximation and equivalence relations can be given in terms of CIU preorders [67], which we define below. We will use the logical relation to prove that CIU and contextual approximations coincide and that both of them coincide with the logical approximation relation.

The definition of the CIU approximation is the same as in [23], only with changed termination relations. We state it here for completeness and reference.

Definition 3.B.4 (CIU approximation)**.** *Must-CIU approximation*, $\lesssim^{CIU}$, is the type-indexed relation defined as follows: for all e, e' with $\Delta;\Gamma \vdash e : \tau$ and $\Delta;\Gamma \vdash e' : \tau$, we define $\Delta \mid \Gamma \vdash e \lesssim^{CIU}_{\Downarrow} e' : \tau$ if and only if

$$\forall \delta \in \mathbf{Type}(\Delta), \gamma \in \mathbf{Subst}(\Gamma\delta), E \in \mathbf{Stk}(\tau\delta),\ E[\mathbf{e}'\gamma]\uparrow \to E[\mathbf{e}\gamma]\uparrow$$

Note again the order of terms e and e' in the implication. ♦

It is simple to see that must-contextual approximation implies must-CIU approximation. However the converse is not so simple to see. We will prove it by constructing the logical relation and proving that all three relations coincide.

Logical approximation relation

We now define the logical relation, internalising the definition in [23]. The major difference is that instead of using must-termination and its stratified version we use may-divergence and stratified may-divergence predicates.

Relational interpretation of types Given two closed types $\tau, \tau' \in \mathbf{Type}$ let

$$\begin{aligned}\mathbf{VRel}(\tau,\tau') &= \mathcal{P}(\mathbf{Val}(\tau) \times \mathbf{Val}(\tau'))\\ \mathbf{TRel}(\tau,\tau') &= \mathcal{P}(\mathbf{Tm}(\tau) \times \mathbf{Tm}(\tau'))\\ \mathbf{SRel}(\tau,\tau') &= \mathcal{P}(\mathbf{Stk}(\tau) \times \mathbf{Stk}(\tau')).\end{aligned}$$

We implicitly use the inclusion

$$\mathbf{VRel}(\tau,\tau') \subseteq \mathbf{TRel}(\tau,\tau').$$

For a type variable context Δ we define $\mathbf{VRel}(\Delta)$ as the extension of $\mathbf{VRel}(\cdot,\cdot)$. We defined $\mathbf{VRel}(\Delta)$ to be the set

$$\{(\varphi_1,\varphi_2,\varphi_r) \mid \varphi_1,\varphi_2 : \Delta \to \mathbf{Type}, \forall \alpha \in \Delta, \varphi_r(\alpha) \subseteq \mathbf{VRel}(\varphi_1(\alpha),\varphi_2(\alpha))\}$$

where the first two components give syntactic types for the left and right hand sides of the relation and the third component is a relation between those types.

The interpretation of types, $[\![\cdot \vdash \cdot]\!]$ is by induction on the judgement $\Delta \vdash \tau$. Given a judgement $\Delta \vdash \tau$ and $\varphi \in \mathbf{VRel}(\Delta)$ the interpretation satisfies

$$[\![\Delta \vdash \tau]\!](\varphi) \subseteq \mathbf{VRel}(\varphi_1(\tau),\varphi_2(\tau))$$

where the φ_1 and φ_2 are the first two components of φ and $\varphi_1(\tau)$ denotes substitution of types in φ_1 for free type variables in τ. It is defined in Figure 3.3 on page 89. For the sake of readability we omit the typing judgements in each case, but they should be understood to be there.

The following lemmas are simple to prove.

Lemma 3.B.5. $\cdot^{\top\top}$ *is monotone and inflationary, i.e.*

- *For all* r, $r \subseteq r^{\top\top}$
- *For all* r, s, $r \subseteq s \to r^{\top\top} \subseteq s^{\top\top}$

◊

Lemma 3.B.6. *Let* $r \in \mathbf{VRel}(\tau, \tau')$.

- *If* $e \overset{p,0}{\rightsquigarrow} e_1$ *and* $e' \overset{p}{\rightsquigarrow} e_1'$ *then* $(e, e') \in r^{\top\top} \leftrightarrow (e_1, e_1') \in r^{\top\top}$.
- *If* $e \overset{1}{\rightsquigarrow} e_1$ *then for all* $e' \in \mathbf{Tm}(\tau')$, *if* $\triangleright((e_1, e') \in r^{\top\top})$ *then* $(e, e') \in r^{\top\top}$.

◊

A crucial property of the interpretation of types is that it respects substitution and weakening.

Lemma 3.B.7 (Substitution). *Let* $\Delta \vdash \tau$ *and* $\Delta, \alpha \vdash \sigma$. *Then*

$$[\![\Delta \vdash \sigma[\tau/\alpha]]\!](\varphi) = [\![\Delta, \alpha \vdash \sigma]\!](\varphi[\alpha \mapsto [\![\Delta \vdash \tau]\!](\varphi)]).$$

◊

Lemma 3.B.8 (Weakening). *Suppose* $\Delta \vdash \tau$. *Then for all* $\varphi \in \mathbf{VRel}(\Delta)$, $s \in \mathbf{VRel}(\sigma, \sigma')$ *and for all* $\alpha \notin \Delta$,

$$[\![\Delta \vdash \tau]\!](\varphi) = [\![\Delta, \alpha \vdash \tau]\!](\varphi[\alpha \mapsto (\sigma, \sigma', s)]).$$

◊

The actual "logical relation" is defined on open terms by reducing it to the above relations on closed terms by substitution. We first extend the interpretation of types to the interpretation of contexts and define $[\![\Delta \vdash \Gamma]\!](\varphi)$ to be

$$\left\{(\gamma, \gamma') \;\middle|\; \gamma, \gamma' : \mathbf{Val}^{\mathrm{dom}(\Gamma)}, \forall x \in \mathrm{dom}(\Gamma), (\gamma(x), \gamma'(x)) \in [\![\Delta \vdash \Gamma(x)]\!](\varphi)\right\}.$$

Definition 3.B.9 (Logical relation). Two terms are logically related

$$\Delta; \Gamma \vdash e \precsim_{\Downarrow}^{log} e' : \tau$$

if

$$\forall \varphi \in \mathbf{VRel}(\Delta), \forall (\gamma, \gamma') \in [\![\Delta \vdash \Gamma]\!](\varphi), (e\gamma, e'\gamma') \in [\![\Delta \vdash \tau]\!](\varphi)^{\top\top}.$$

♦

Properties of relations

Definition 3.B.10 (Type-indexed relation)**.** A *type-indexed relation* $\mathcal{R}$ is a set of tuples $(\Delta,\Gamma,e,e',\tau)$ such that $\Delta \vdash \Gamma$, $\Delta \vdash \tau$, $\Delta \mid \Gamma \vdash e : \tau$ and $\Delta \mid \Gamma \vdash e' : \tau$. We write $\Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau$ for $(\Delta,\Gamma,e,e',\tau) \in \mathcal{R}$. ♦

Definition 3.B.11 (Precongruence)**.** A type-indexed relation $\mathcal{R}$ is *reflexive* if $\Delta;\Gamma \vdash e : \tau$ implies $\Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e : \tau$. It is *transitive* if $\Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau$ and $\Delta;\Gamma \vdash e' \mathrel{\mathcal{R}} e'' : \tau$ implies $\Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e'' : \tau$. It is compatible if it is closed under the rules in Figure 3.8.

A *precongruence* is a reflexive, transitive and compatible type-indexed relation. ♦

$$\dfrac{}{\Delta;\Gamma \vdash x \mathrel{\mathcal{R}} x : \tau}\; x{:}\tau \in \Gamma \qquad \dfrac{}{\Delta;\Gamma \vdash \langle\rangle \mathrel{\mathcal{R}} \langle\rangle : \mathbf{1}}$$

$$\dfrac{\Delta;\Gamma \vdash e_1 \mathrel{\mathcal{R}} e_1' : \tau_1 \qquad \Delta;\Gamma \vdash e_2 \mathrel{\mathcal{R}} e_2' : \tau_2}{\Delta;\Gamma \vdash \langle e_1, e_2\rangle \mathrel{\mathcal{R}} \langle e_1', e_2'\rangle : \tau_1 \times \tau_2} \qquad \dfrac{\Delta;\Gamma, x{:}\tau_1 \vdash e \mathrel{\mathcal{R}} e' : \tau_2}{\Delta;\Gamma \vdash \lambda x.e \mathrel{\mathcal{R}} \lambda x.e' : \tau_1 \to \tau_2}$$

$$\dfrac{\Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau_1}{\Delta;\Gamma \vdash \mathtt{inl}\, e \mathrel{\mathcal{R}} \mathtt{inl}\, e' : \tau_1 + \tau_2} \qquad \dfrac{\Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau_2}{\Delta;\Gamma \vdash \mathtt{inr}\, e \mathrel{\mathcal{R}} \mathtt{inr}\, e' : \tau_1 + \tau_2}$$

$$\dfrac{\Delta;\Gamma, x_1{:}\tau_1 \vdash e_1 \mathrel{\mathcal{R}} e_1' : \tau \qquad \Delta;\Gamma, x_2{:}\tau_2 \vdash e_2 \mathrel{\mathcal{R}} e_2' : \tau \qquad \Delta;\Gamma \vdash e\; e' : \tau_1 + \tau_2}{\Delta\;\Gamma \vdash \mathtt{case}\,(e, x_1.e_1, x_2.e_2) \mathrel{\mathcal{R}} \mathtt{case}\,(e', x_1.e_1', x_2.e_2') : \tau}$$

$$\dfrac{\Delta, \alpha;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau}{\Delta;\Gamma \vdash \Lambda.e \mathrel{\mathcal{R}} \Lambda.e' : \forall\alpha.\tau} \qquad \dfrac{\Delta \vdash \tau_1 \qquad \Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau[\tau_1/\alpha]}{\Delta;\Gamma \vdash (\mathtt{pack}\, e) \mathrel{\mathcal{R}} (\mathtt{pack}\, e') : \exists\alpha.\tau}$$

$$\dfrac{\Delta;\Gamma \vdash e_1 \mathrel{\mathcal{R}} e_1' : \exists\alpha.\tau_1 \qquad \Delta \vdash \tau \qquad \Delta,\alpha;\Gamma, x : \tau_1 \vdash e \mathrel{\mathcal{R}} e' : \tau}{\Delta;\Gamma \vdash (\mathtt{unpack}\ e_1\ \mathtt{as}\ x\ \mathtt{in}\ e) \mathrel{\mathcal{R}} (\mathtt{unpack}\ e_1'\ \mathtt{as}\ x\ \mathtt{in}\ e') : \tau}$$

$$\dfrac{\Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau_1 \times \tau_2}{\Delta;\Gamma \vdash \mathtt{proj}_i\, e \mathrel{\mathcal{R}} \mathtt{proj}_i\, e' : \tau_i} \qquad \dfrac{\Delta;\Gamma \vdash e_1 \mathrel{\mathcal{R}} e_1' : \tau' \to \tau \qquad \Delta;\Gamma \vdash e_2 \mathrel{\mathcal{R}} e_2' : \tau'}{\Delta;\Gamma \vdash e_1\, e_2 \mathrel{\mathcal{R}} e_1'\, e_2' : \tau}$$

$$\dfrac{\Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \mu\alpha.\tau}{\Delta;\Gamma \vdash \mathtt{unfold}\, e \mathrel{\mathcal{R}} \mathtt{unfold}\, e' : \tau[\mu\alpha.\tau/\alpha]} \qquad \dfrac{\Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau[\mu\alpha.\tau/\alpha]}{\Delta;\Gamma \vdash \mathtt{fold}\, e \mathrel{\mathcal{R}} \mathtt{fold}\, e' : \mu\alpha.\tau}$$

$$\dfrac{\Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \forall\alpha.\tau}{\Delta;\Gamma \vdash e[\,] \mathrel{\mathcal{R}} e'[\,] : \tau[\tau'/\alpha]}\; \mathit{ftv}(\tau') \subseteq \Delta \qquad \dfrac{}{\Delta;\Gamma \vdash ? \mathrel{\mathcal{R}} ? : \mathtt{nat}}$$

Figure 3.8: Compatibility properties of type-indexed relations.

The fundamental property

To prove the fundamental property (reflexivity) we start with some simple properties relating evaluation contexts and relations. The proof of the compatibility properties in most of the cases will be a simple consequence of these lemmas.

The following is a direct consequence of the fact that $p \to \triangleright p$ for any $p : \Omega$, we only state it here for reference.

Lemma 3.B.12. *The interpretations of types satisfy the following monotonicity properties.*

- *If* $(v, v') \in [\![\Delta \vdash \tau]\!](\varphi)$ *then* $\triangleright((v, v') \in [\![\Delta \vdash \tau]\!](\varphi))$.
- *If* $(e, e') \in [\![\Delta \vdash \tau]\!](\varphi)^{\top\top}$ *then* $\triangleright\Big((e, e') \in [\![\Delta \vdash \tau]\!](\varphi)^{\top\top}\Big)$.
- *If* $(E, E') \in [\![\Delta \vdash \tau]\!](\varphi)^{\top}$ *then* $\triangleright\Big((E, E') \in [\![\Delta \vdash \tau]\!](\varphi)^{\top}\Big)$.

$\Diamond$

Lemma 3.B.13. *If* $(v, v') \in [\![\Delta \vdash \tau_1 \to \tau_2]\!](\varphi)$ *and* $(E, E') \in [\![\Delta \vdash \tau_2]\!](\varphi)^{\top}$ *then* $(E \circ (v\,[\,]), E' \circ (v'\,[\,])) \in [\![\Delta \vdash \tau_1]\!](\varphi)^{\top}$. $\Diamond$

This follows directly from the definition of the interpretation of types, Lemma 3.B.2 and Lemma 3.B.1.

Corollary 3.B.14. *If* $(e, e') \in [\![\Delta \vdash \tau_1]\!](\varphi)^{\top\top}$ *and* $(E, E') \in [\![\Delta \vdash \tau_2]\!](\varphi)^{\top}$ *then*

$$(E \circ ([\,]\,e), E' \circ ([\,]\,e')) \in [\![\Delta \vdash \tau_1 \to \tau_2]\!](\varphi)^{\top}.$$

$\Diamond$

Proof. Take $(v, v') \in [\![\Delta \vdash \tau_1 \to \tau_2]\!](\varphi)$. By Lemma 3.B.13 $(E \circ (v\,[\,]), E' \circ (v'\,[\,]) \in [\![\Delta \vdash \tau_1]\!](\varphi)^{\top}$ so using Lemma 3.B.1 we have

$$E'[v'\,e']\!\uparrow \to E[v\,e]\underline{\uparrow}$$

concluding the proof. 𝔔𝔈𝔇

Corollary 3.B.15. *If* $(e, e') \in [\![\Delta \vdash \tau_1 \to \tau_2]\!](\varphi)^{\top\top}$ *and* $(E, E') \in [\![\Delta \vdash \tau_2]\!](\varphi)^{\top}$ *then* $(E \circ ((\lambda x.e\,x)\,[\,]), E' \circ ((\lambda x.e'\,x)\,[\,])) \in [\![\Delta \vdash \tau_1]\!](\varphi)^{\top}$. $\Diamond$

Proof. If $(e, e') \in [\![\Delta \vdash \tau_1 \to \tau_2]\!](\varphi)^{\top\top}$ then $(\lambda x.e, \lambda x.e') \in [\![\Delta \vdash \tau_1 \to \tau_2]\!](\varphi)$. Use Lemma 3.B.13. 𝔔𝔈𝔇

Lemma 3.B.16. *If* $(E, E') \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)^{\top}$ *then*

$$(E \circ (\texttt{unfold}\,[\,]), E' \circ (\texttt{unfold}\,[\,])) \in [\![\Delta \vdash \mu\alpha.\tau]\!](\varphi)^{\top}.$$

$\Diamond$

Proof. Take

$$(\texttt{fold}\,v, \texttt{fold}\,v') \in [\![\Delta \vdash \mu\alpha.\tau]\!](\varphi)$$

with

$$\triangleright((v,v') \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)).$$

Using Lemma 3.B.1, Lemma 3.B.5 and Lemma 3.B.6 we have

$$E'[\texttt{unfold}(\texttt{fold}\,v')]\uparrow \leftrightarrow E'[v']\uparrow$$

which implies $\triangleright(E'[v'])$ which further implies $\triangleright(E[v]\underline{\uparrow})$ which finally implies $E[\texttt{unfold}(\texttt{fold}\,v)]\underline{\uparrow}$ by definition of stratified may-divergence, concluding the proof. $\mathfrak{QED}$

Lemma 3.B.17. *If* $(E,E') \in [\![\Delta \vdash \mu\alpha.\tau]\!](\varphi)^{\top}$ *then*

$$(E \circ (\texttt{fold}\,[]), E' \circ (\texttt{fold}\,[])) \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)^{\top}.$$

◊

Proof. Easily follows from the fact that if (v,v') are related at the unfolded type then $(\texttt{fold}\,v, \texttt{fold}\,v')$ are related at the folded type. $\mathfrak{QED}$

Lemma 3.B.18. *If* $(E,E') \in [\![\Delta \vdash \exists\alpha.\tau]\!](\varphi)^{\top}$ *then for all* $\Delta \vdash \tau_1$

$$(E \circ (\texttt{pack}\,[]), E' \circ (\texttt{pack}\,[])) \in [\![\Delta \vdash \tau[\tau_1/\alpha]]\!](\varphi)^{\top}.$$

◊

Proof. Take

$$(v,v') \in [\![\Delta \vdash \tau[\tau_1/\alpha]]\!](\varphi).$$

Lemma 3.B.7 implies

$$(v,v') \in [\![\Delta, \alpha \vdash \tau]\!](\varphi[\alpha \mapsto [\![\Delta \vdash \tau]\!](\varphi)])$$

which further means that $(\texttt{pack}\,v, \texttt{pack}\,v') \in [\![\Delta \vdash \exists\alpha.\tau]\!](\varphi)$ which is easily seen to imply the conclusion. $\mathfrak{QED}$

Lemma 3.B.19. *Let* $\Delta \vdash \tau$, $(E,E') \in [\![\Delta \vdash \tau]\!](\varphi)^{\top}$. *If for all* $\sigma, \sigma' \in$ **Type** *and* $s \in$ **VRel**(σ,σ') *and for all* $(v,v') \in [\![\Delta, \alpha \vdash \tau_1]\!](\varphi[\alpha \mapsto s])$,

$$(e[v/x], e'[v'/x]) \in [\![\Delta \vdash \tau]\!](\varphi)^{\top\top}$$

then

$$(E \circ (\texttt{unpack}\;[]\;\texttt{as}\;x\;\texttt{in}\;e), E' \circ (\texttt{unpack}\;[]\;\texttt{as}\;x\;\texttt{in}\;e')) \in [\![\Delta \vdash \exists\alpha.\tau_1]\!](\varphi)^{\top}.$$

◊

Proof. Take $(\mathtt{pack}\,v, \mathtt{pack}\,v') \in [\![\Delta \vdash \exists\alpha.\tau_1]\!](\varphi)$. This implies there exist $\sigma, \sigma' \in$ **Type** and an $s \in \mathbf{VRel}(\sigma, \sigma)$, such that $(v, v') \in [\![\Delta, \alpha \vdash \tau_1]\!](\varphi[\alpha \mapsto s])$. An assumption of this lemma further implies $(e[v/x], e'[v'/x]) \in [\![\Delta \vdash \tau]\!](\varphi)^{\top\top}$. It is now easy to conclude the proof using Lemma 3.B.6. 𝔔𝔈𝔇

The other lemmas concerning context composition are proved similarly. The next lemma will be used to prove compatibility of ?.

Lemma 3.B.20. *For all* $n \in \mathbb{N}$, $(\underline{n}, \underline{n}) \in [\![\vdash \mathtt{nat}]\!]$. ◊

Proof. By induction on n.

$n = 0$ Then $\underline{n} = \mathtt{fold}\,\mathtt{inl}\,\langle\rangle$. It is easy to see that $(\mathtt{inl}\,\langle\rangle, \mathtt{inl}\,\langle\rangle) \in [\![\vdash 1 + \mathtt{nat}]\!]$ and so the result follows by the definition of interpretation of recursive types and Lemma 3.B.12.

$n = m + 1$ Then $\underline{n} = \mathtt{fold}\,\mathtt{inr}\,\underline{m}$. By assumption

$$(\underline{m}, \underline{m}) \in [\![\vdash \mathtt{nat}]\!]$$

and so $(\mathtt{inr}\,\underline{m}, \mathtt{inr}\,\underline{m}) \in [\![\vdash 1 + \mathtt{nat}]\!]$. The result easily follows by the definition of interpretation of recursive types and Lemma 3.B.12. 𝔔𝔈𝔇

We are now ready to prove that the logical approximation relation is compatible.

Proposition 3.B.21. *The relation* $\lesssim_{\Downarrow}^{log}$ *is closed under all the rules in Figure 3.8, i.e. it is compatible.* ◊

Proof. We only show some cases. The general rule is that compatibility rules are proved either by directly showing two values are related at the value relation or relying on the above lemmas and extending the evaluation contexts.

- Introduction of recursive types.

$$\frac{\Delta;\Gamma \vdash e \lesssim_{\Downarrow}^{log} e' : \tau[\mu\alpha.\tau/\alpha]}{\Delta;\Gamma \vdash \mathtt{fold}\,e \lesssim_{\Downarrow}^{log} \mathtt{fold}\,e' : \mu\alpha.\tau}$$

Take $\varphi \in \mathbf{VRel}(\Delta)$ and $(\gamma, \gamma') \in [\![\Delta \vdash \Gamma]\!](\varphi)$. Let $f = e\gamma$ and $f' = e'\gamma'$. We have to show $(\mathtt{fold}\,f, \mathtt{fold}\,f') \in [\![\Delta \vdash \mu\alpha.\tau]\!]\varphi^{\top\top}$. So take $(E, E') \in [\![\Delta \vdash \mu\alpha.\tau]\!]\varphi^{\top}$. By assumption $(f, f') \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!]\varphi^{\top\top}$ so it suffices to show $(E \circ (\mathtt{fold}\,[]), E' \circ \mathtt{fold}\,[]) \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!]\varphi^{\top}$, but this is exactly the content of Lemma 3.B.17.

- Elimination of recursive types.

$$\frac{\Delta;\Gamma \vdash e \lesssim^{log}_{\Downarrow} e' : \mu\alpha.\tau}{\Delta;\Gamma \vdash \texttt{unfold}\, e \lesssim^{log}_{\Downarrow} \texttt{unfold}\, e' : \tau[\mu\alpha.\tau/\alpha]}$$

  Exactly the same reasoning as in the previous case, only this time we use Lemma 3.B.16 in place of Lemma 3.B.17.

- The ? expression.

$$\frac{}{\Delta;\Gamma \vdash ? \lesssim^{log}_{\Downarrow} ? : \texttt{nat}}$$

  By Lemma 3.B.20 we have $\forall n, (\underline{n}, \underline{n}) \in [\![\vdash \texttt{nat}]\!]$.

  Take $(E, E') \in [\![\vdash \texttt{nat}]\!]^{\top}$ and assume $E'[?]\uparrow$. By definition of the $\uparrow$ relation there exists an e', $? \rightsquigarrow e'$ and $E[e']\uparrow$. Inspecting the operational semantics we see that $e' = \underline{n}$ for some $n \in \mathbb{N}$. This implies $E[\underline{n}]\underline{\uparrow}$ which further implies by Lemma 3.B.2 that $E[?]\underline{\uparrow}$.

- Introduction of existential type.

$$\frac{\Delta \vdash \tau_1 \qquad \Delta;\Gamma \vdash e \lesssim^{log}_{\Downarrow} e' : \tau[\tau_1/\alpha]}{\Delta;\Gamma \vdash \texttt{pack}\, e \lesssim^{log}_{\Downarrow} \texttt{pack}\, e' : \exists\alpha.\tau}$$

  Take $\varphi \in \mathbf{VRel}(\Delta)$ and $(\gamma, \gamma') \in [\![\Delta \vdash \Gamma]\!](\varphi)$. Let $f = e\gamma$ and $f' = e'\gamma'$. We need to show

$$(\texttt{pack}\, f, \texttt{pack}\, f') \in [\![\Delta \vdash \exists\alpha.\tau]\!](\varphi)^{\top\top}$$

  Again by assumption $(f, f') \in [\![\Delta \vdash \tau[\tau_1/\alpha]]\!](\varphi)^{\top\top}$. We can now use Lemma 3.B.18 to finish the proof, as we did above for the case of introduction of recursive types.

- Elimination of existential types.

$$\frac{\Delta;\Gamma \vdash e \lesssim^{log}_{\Downarrow} e' : \exists\alpha.\tau_1 \qquad \Delta \vdash \tau \qquad \Delta,\alpha;\Gamma, x : \tau_1 \vdash e_1 \lesssim^{log}_{\Downarrow} e_1' : \tau}{\Delta;\Gamma \vdash (\texttt{unpack}\, e\, \texttt{as}\, x\, \texttt{in}\, e_1) \lesssim^{log}_{\Downarrow} (\texttt{unpack}\, e'\, \texttt{as}\, x\, \texttt{in}\, e_1') : \tau}$$

  Take $\varphi \in \mathbf{VRel}(\Delta)$ and $(\gamma, \gamma') \in [\![\Delta \vdash \Gamma]\!](\varphi)$. Let $f = e\gamma$ and $f' = e'\gamma'$, $f_1 = e_1\gamma$, $f_1' = e_1'\gamma$. We need to show

$$(\texttt{unpack}\, f\, \texttt{as}\, x\, \texttt{in}\, f_1, \texttt{unpack}\, f'\, \texttt{as}\, x\, \texttt{in}\, f_1') \in [\![\Delta \vdash \tau]\!](\varphi)^{\top\top}.$$

  The premise of this case shows that for all types $\sigma, \sigma' \in \mathbf{Type}$ and relations $s \in \mathbf{VRel}(\sigma, \sigma')$, for all $(v, v') \in [\![\Delta, \alpha \vdash \tau_1]\!](\varphi[\alpha \mapsto s])$, we have

$$(f_1[v/x], f_1'[v'/x]) \in [\![\Delta, \alpha \vdash \tau]\!](\varphi[\alpha \mapsto s])^{\top\top}$$

and by Lemma 3.B.8 this is the same as for all $\sigma, \sigma' \in \mathbf{Type}$, for all $s \in \mathbf{VRel}(\sigma, \sigma')$, for all $(v, v') \in [\![\Delta, \alpha \vdash \tau_1]\!](\varphi[\alpha \mapsto s])$,

$$(f_1[v/x], f_1'[v'/x]) \in [\![\Delta \vdash \tau]\!](\varphi)^{\top\top}.$$

We now use Lemma 3.B.19 to conclude the proof. $\mathfrak{QED}$

Corollary 3.B.22 (Fundamental property of logical relations)**.** *If* $\Delta; \Gamma \vdash e : \tau$ *then* $\Delta; \Gamma \vdash e \lesssim_{\Downarrow}^{log} e : \tau$ $\Diamond$

Proof. Every compatible relation is reflexive. This can be shown by an easy induction on the typing derivation. Proposition 3.B.21 shows that the logical relation is compatible, hence it is reflexive. $\mathfrak{QED}$

We need the next corollary to relate the logical approximation relation to must-contextual approximation.

Corollary 3.B.23. *For any expressions* e, e' *and context* C*, if* $\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e' : \tau$ *and* $C : (\Delta \mid \Gamma \Rrightarrow \tau) \hookrightarrow (\Delta' \mid \Gamma' \Rrightarrow \sigma)$ *then* $\Delta' \mid \Gamma' \vdash C[e] \lesssim_{\Downarrow}^{log} C[e'] : \tau'$*.* $\Diamond$

Proof. By induction on the judgement

$$C : (\Delta \mid \Gamma \Rrightarrow \tau) \hookrightarrow (\Delta' \mid \Gamma' \Rrightarrow \sigma),$$

using Proposition 3.B.21. $\mathfrak{QED}$

All three approximation relations coincide

We have already mentioned that it is easy to see that must-contextual approximation implies must-CIU approximation. We now show that must-CIU approximation implies logical approximation.

We start with a lemma showing that the logical approximation relation is closed under postcomposition with the must-CIU approximation relation.

Lemma 3.B.24. *For any terms* e*,* e' *and* e'' *of type* τ *in context* $\Delta \mid \Gamma$*. If* $\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e' :$ *and* $\Delta \mid \Gamma \vdash e' \lesssim_{\Downarrow}^{CIU} e'' : \tau$ *then* $\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e'' :.$ $\Diamond$

Proof. Take $\varphi \in \mathbf{VRel}(\Delta)$ and $(\gamma, \gamma') \in [\![\Delta \vdash \Gamma]\!](\varphi)$. Take $(E, E') \in [\![\tau]\!](\varphi)^{\top}$ and assume $E'[e''\gamma']\uparrow$. Since $\Delta \mid \Gamma \vdash e' \lesssim_{\Downarrow}^{CIU} e'' : \tau$ we have $E'[e'\gamma']\uparrow$ and since $\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e' :$ we further have $E[e\gamma]\mapsup$, concluding the proof. $\mathfrak{QED}$

Corollary 3.B.25. *For any terms* e *and* e' *of type* τ *in context* $\Delta \mid \Gamma$*. If* $\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{CIU} e' : \tau$ *then* $\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e' :.$ $\Diamond$

Proof. By Corollary 3.B.22 we have that $\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e :$. The previous lemma concludes the proof. $\mathfrak{QED}$

The only missing link in the chain of inclusions is the implication from the logical relation to contextual approximation. This, however, requires some more work.

Adequacy

We wish to show soundness of the logical relation with respect to must-contextual approximation. However, the implication

$$\Delta \mid \Gamma \vdash e \precsim^{log}_{\Downarrow} e' : \tau \to \Delta \mid \Gamma \vdash e \precsim^{ctx}_{\Downarrow} e' : \tau$$

does not hold internally, due to the different divergence relations used in the definition of the logical relation. To see precisely where the proof fails let us attempt it. Let $\Delta \mid \Gamma \vdash e \precsim^{log}_{\Downarrow} e' : \tau$ and take a well-typed closing context C with result type σ. Then by Corollary 3.B.23, $\varnothing \mid \varnothing \vdash C[e] \precsim^{log}_{\Downarrow} C[e'] : \sigma$. Unfolding the definition of the logical relation we get $(C[e], C[e']) \in [\![\varnothing \vdash \sigma]\!]^{\top\top}$. It is easy to see that $(-,-) \in [\![\varnothing \vdash \sigma]\!]^{\top}$ and so we get by definition of $\top\top$ that $C[e']{\uparrow} \to C[e]{\underline{\uparrow}}$. However the definition of contextual equivalence requires the implication $C[e']{\uparrow} \to C[e]{\uparrow}$, which is not a consequence of the previous one.

Intuitively, the gist of the problem is that $\uparrow$ defines a time-independent predicate, whereas $\underline{\uparrow}$ depends on the time, as explained in the introduction. However, in the model in we can show the following rule is admissible.

Lemma 3.B.26. *$e : \mathbf{Tm} \mid \varnothing \vdash \square(e{\underline{\uparrow}}) \to e{\uparrow}$ holds in the logic.* $\Diamond$

Thus we additionally assume this principle in our logic. Using this, we are led to the following corrected statement of adequacy using the $\square$ modality.

Theorem 3.B.27 (Adequacy)**.** *If e and e' are of type τ in context $\Delta \mid \Gamma$ then $\square(\Delta \mid \Gamma \vdash e \precsim^{log}_{\Downarrow} e' : \tau)$ implies $\Delta \mid \Gamma \vdash e \precsim^{ctx}_{\Downarrow} e' : \tau$.* $\Diamond$

To prove this theorem we first observe that all the lemmas used in the proof of Corollary 3.B.23 are proved in constant contexts using only other constant facts and so Corollary 3.B.23 can be strengthened. More precisely, we can prove the following restatement.

Proposition 3.B.28. $\square[\forall \Delta, \Delta', \Gamma, \Gamma', \tau, \sigma, C, e, e', C : (\Delta \mid \Gamma \Rrightarrow \tau) \hookrightarrow (\Delta' \mid \Gamma' \Rrightarrow \sigma)$ $\to \Delta \mid \Gamma \vdash e \precsim^{log}_{\Downarrow} e' : \tau \to \Delta' \mid \Gamma' \vdash C[e] \precsim^{log}_{\Downarrow} C[e'] : \tau']$. $\Diamond$

Note that all the explicit universal quantification in the proposition is over constant types. One additional ingredient we need to complete the proof is the fact that $\uparrow$ is $\neg\neg$-closed, i.e. $e{\uparrow} \leftrightarrow \neg\neg(e{\uparrow})$. We can show this in the logic using the fact that $\uparrow$ is the greatest post-fixed point by showing that $\neg\neg{\uparrow}$ is another one. This fact further means that $\square(e{\uparrow}) \leftrightarrow (e{\uparrow})$. We are now ready to proceed with the proof of Theorem 3.B.27.

Theorem 3.B.27. Continuing the proof we started above we get, using Proposition 3.A.6, that $\square(C[e']\uparrow \to C[e]\underline{\uparrow})$ and thus also

$$\square(C[e']\uparrow) \to \square(C[e]\underline{\uparrow}).$$

We have $\square(C[e']\uparrow) \leftrightarrow C[e']\uparrow$ and by Lemma 3.B.26 $\square(C[e]\underline{\uparrow}) \leftrightarrow C[e]\uparrow$. We can thus conclude $C[e']\uparrow \to C[e]\uparrow$, as required. 𝔔𝔈𝔇

Using Proposition 3.A.25 and Proposition 3.A.24 we can see that for each Δ, Γ, e, e' and τ,

$$\Delta \mid \Gamma \vdash e \lesssim^{ctx}_{\Downarrow} e' : \tau \leftrightarrow \neg\neg(\Delta \mid \Gamma \vdash e \lesssim^{ctx}_{\Downarrow} e' : \tau)$$

and similarly

$$\Delta \mid \Gamma \vdash e \lesssim^{CIU}_{\Downarrow} e' : \tau \leftrightarrow \neg\neg(\Delta \mid \Gamma \vdash e \lesssim^{CIU}_{\Downarrow} e' : \tau).$$

Combining this observation with the above we have

Theorem 3.B.29. *For any Δ, Γ, e, e' and τ,*

$$\Delta \mid \Gamma \vdash e \lesssim^{CIU}_{\Downarrow} e' : \tau \leftrightarrow \Delta \mid \Gamma \vdash e \lesssim^{ctx}_{\Downarrow} e' : \tau \leftrightarrow \square(\Delta \mid \Gamma \vdash e \lesssim^{log}_{\Downarrow} e' : \tau)$$

◊

Proof. The only missing link is the implication from CIU approximation to "boxed" logical approximation. However using the previous observation that CIU approximation is $\neg\neg$-closed with Corollary 3.B.25 and the fact that $\neg\neg$ is the left adjoint to $\square$, we get the desired implication. 𝔔𝔈𝔇

Examples of the use of logical relation

We show the syntactic minimal invariance example. We start with two simple lemmas.

Lemma 3.B.30. *Let $\tau, \sigma \in \mathbf{Type}$, let $e \in \mathbf{Tm}\,(\tau \to \sigma)$, $v \in \mathbf{Val}\,(\tau \to \sigma)$. Then*

$$(e, v) \in [\![\tau \to \sigma]\!]^{\top\top} \to (\lambda x.e\,x, v) \in [\![\tau \to \sigma]\!].$$

◊

Proof. By assumption $v = \lambda x.e'$ for some x and e'. Take $(u, u') \in [\![\tau]\!]$ and we're supposed to show $(e\,u, e'[u'/x]) \in [\![\sigma]\!]^{\top\top}$. By Lemma 3.B.6 it suffices to show $(e\,u, v\,u') \in [\![\sigma]\!]^{\top\top}$ and this is a simple consequence of Corollary 3.B.14. 𝔔𝔈𝔇

Lemma 3.B.31. *Let $\tau, \sigma \in \mathbf{Type}$, let $e \in \mathbf{Tm}\,(\tau \to \sigma)$, $v \in \mathbf{Val}\,(\tau \to \sigma)$. Then*

$$(v, e) \in [\![\tau \to \sigma]\!]^{\top\top} \to (v, \lambda x.e\,x) \in [\![\tau \to \sigma]\!].$$

◊

Syntactic minimal invariance

Let $\mathtt{fix} : \forall\alpha,\beta.((\alpha\to\beta)\to(\alpha\to\beta))\to(\alpha\to\beta)$ be the term $\Lambda.\Lambda.\lambda f.\delta_f(\mathtt{fold}\,\delta_f)$ where δ_f is the term $\lambda y.\mathtt{let}\ y' = \mathtt{unfold}\,y\ \mathtt{in}\ f\,(\lambda x.y'\,y\,x)$.

Consider the type $\tau = \mu\alpha.\mathtt{nat} + \alpha \to \alpha$. Let $\mathrm{id} = \lambda x.x$ and consider the term

$$f \equiv \lambda h,x.\mathtt{case}\,(\mathtt{unfold}\,x, y.\mathtt{fold}\,(\mathtt{inl}\,y), g.\mathtt{fold}\,(\mathtt{inr}\,\lambda y.h(g(h\,y)))).$$

We show that $\mathtt{fix}[][]f \cong_{\Downarrow} \mathrm{id} : \tau\to\tau$. First we show that they either logically approximates the other and then use the fact that we have proved this in the context of only constant facts to conclude that the statement always holds. Thus we use Theorem 3.B.29 to conclude that the terms are contextually equivalent.

$\Rightarrow$ We first show by Löb induction that $(\mathtt{fix}[][]f, \mathrm{id}) \in [\![\tau\to\tau]\!]^{\top\top}$. It is easy to see that

$$\mathtt{fix}[][]f \overset{p,1}{\rightsquigarrow} \lambda x.\mathtt{case}\,(\mathtt{unfold}\,x, y.\mathtt{fold}\,(\mathtt{inl}\,y), g.\mathtt{fold}\,(\mathtt{inr}\,\lambda y.h(g(h\,y)))).$$

where $h = \lambda x.\delta_f(\mathtt{fold}\,\delta_f)\,x$. Let

$$\varphi = \lambda x.\mathtt{case}\,(\mathtt{unfold}\,x, y.\mathtt{fold}\,(\mathtt{inl}\,y), g.\mathtt{fold}\,(\mathtt{inr}\,\lambda y.h(g(h\,y)))).$$

We directly show that $(\varphi, \mathrm{id}) \in [\![\tau\to\tau]\!]$ which suffices by Lemma 3.B.12 and Lemma 3.B.6.

So take $(u,u') \in [\![\tau]\!]$. By the definition of the interpretation of recursive types there are two cases

- $u = \mathtt{fold}\,(\mathtt{inl}\,\underline{n})$ and $u' = \mathtt{fold}\,(\mathtt{inl}\,\underline{n})$ for some $n \in \mathbb{N}$. This case is immediate.
- $u = \mathtt{fold}\,(\mathtt{inr}\,g)$, $u' = \mathtt{fold}\,(\mathtt{inr}\,g')$ and $\triangleright((g,g') \in [\![\tau\to\tau]\!])$. We then have that $\varphi\,u \overset{p,1}{\rightsquigarrow} \mathtt{fold}\,(\mathtt{inr}\,\lambda y.h(g(h\,y)))$ and $\mathrm{id}\,u' \overset{p}{\rightsquigarrow} u'$ and so it suffices to show

$$\triangleright(\lambda y.(h(g(h\,y)), g') \in [\![\tau\to\tau]\!]).$$

We again show that these are related as values so take $\triangleright((v,v') \in [\![\tau]\!])$ and we need to show $\triangleright\Big((h(g(h\,v)), g'\,v') \in [\![\tau]\!]^{\top\top}\Big)$. Take

$$\triangleright((E,E') \in [\![\tau]\!]^{\top}).$$

Löb induction hypothesis gives us

$$\triangleright((h',\mathrm{id}) \in [\![\tau\to\tau]\!]^{\top\top}),$$

where h' is the body of h, i.e $h = \lambda x.h'\,x$. By Lemma 3.B.30

$$\triangleright((h, \mathrm{id}) \in [\![\tau \to \tau]\!]^{\top\top})$$

and so by using Lemma 3.B.13 three times we get

$$\triangleright\Big((E[h\,(g\,(h\,[\,]))], E'[g'\,[\,]]) \in [\![\tau]\!]^{\top}\Big).$$

So assuming $\triangleright(E'[g'\,v']\uparrow)$ we get $\triangleright(E[h\,(g\,(h\,v))]\underline{\uparrow})$, concluding the proof.

$\Leftarrow$ This direction is essentially the same, only using Lemma 3.B.31 in place of Lemma 3.B.30.

Least prefixed point

We now prove the following recursion induction principle for the fixed-point combinator. The rule is the same as in [23] and the proof is morally the same, except that we replace induction on ordinals by Löb induction, thus removing a lot of unnecessary bookkeeping. More precisely, we prove

$$\frac{\Delta \mid \Gamma \vdash v \lesssim^{ctx}_{\Downarrow} v' : \tau_1 \to \tau_2}{\Delta \mid \Gamma \vdash \mathtt{fix}[\,][\,]\,v \lesssim^{ctx}_{\Downarrow} v' : \tau_1 \to \tau_2}$$

We do this in a few stages. For simplicity we only consider the case where Δ and Γ are empty. The general case is proved in the same way.

It is easy to see that $\mathtt{fix}[\,][\,]\,v \overset{1}{\rightsquigarrow} v\,h$ where $h = \lambda x.\delta_f\,(\mathtt{fold}\,\delta_f)\,x$. We first show $(h, v') \in [\![\tau_1 \to \tau_2]\!]$ by Löb induction. So assume $\triangleright((h, v') \in [\![\tau_1 \to \tau_2]\!])$. Since v' is a value of the function type $v' = \lambda y.e$ for some y and typeable e. Take $(u, u') \in [\![\tau_1]\!]$ and $(E, E') \in [\![\tau_2]\!]^{\top}$. Then using Corollary 3.B.14, then Corollary 3.B.22 applied to v and then Corollary 3.B.15 for v we have

$$(E[((\lambda x.v\,x)\,-)\,u], E[((\lambda x.v\,x)\,-)\,u]) \in [\![\tau_1 \to \tau_2]\!]^{\top}.$$

Using the assumption that $v\,v'$ approximates v' and Theorem 3.B.29 we get

$$E'[v'\,u']\uparrow \rightarrow E'[(v\,v')\,u']\uparrow$$

and thus

$$\triangleright(E'[(v\,v')\,u']\uparrow)$$

and now using the induction hypothesis $\triangleright((h, v') \in [\![\tau_1 \to \tau_2]\!])$ and the fact that $(E[((\lambda x.v\,x)\,-)\,u], E[((\lambda x.v\,x)\,-)\,u]) \in [\![\tau_1 \to \tau_2]\!]^{\top}$ we get

$$\triangleright(E[(v\,h)\,u]\underline{\uparrow})$$

and since $h\,u \overset{1}{\rightsquigarrow} (v\,h)\,u$ we can use Lemma 3.B.2 to get

$$E[h\,u]\Uparrow$$

concluding the proof.

Since the proof relies only on constant facts we have, using Theorem 3.B.29, that h contextually approximates v'. Thus

$$\mathtt{fix}[][]\,v \lesssim^{ctx}_{\Downarrow} v\,h \lesssim^{ctx}_{\Downarrow} v\,v' \lesssim^{ctx}_{\Downarrow} v'.$$

Parametricity

We now characterise the values of type $\forall\alpha.\alpha\times\alpha\to\alpha$. We start by proving some expected properties of values of the polymorphic and function types.

Lemma 3.B.32. *Let $\alpha\vdash\tau$ and $v\in\mathbf{Val}(\forall\alpha.\tau)$. Then for all types σ,σ' and relations $s\in\mathbf{VRel}(\sigma,\sigma')$ we have*

$$(v[],v[])\in[\![\alpha\vdash\tau]\!](\varphi)^{\top\top}$$

where φ maps α to (σ,σ',s). ◊

Let $\mathbf{\Omega}=\forall\alpha.\mathtt{fix}[](\lambda f.f)\langle\rangle$ be the term of type $\forall\alpha.\alpha$ that deterministically diverges when instantiated (applied).

Lemma 3.B.33. *Let $v\in\mathbf{Val}(\forall\alpha.\alpha\times\alpha\to\alpha)$. If $v[]$ may-diverges then*

$$\emptyset\mid\emptyset\vdash v =^{ctx}_{\Downarrow} \forall\alpha.\mathbf{\Omega}[] : \forall\alpha.\alpha\times\alpha\to\alpha.$$

◊

Proof. This is a simple consequence of Lemma 3.B.39. 𝔔𝔈𝔇

Lemma 3.B.34. *Let $v\in\mathbf{Val}(\forall\alpha.\alpha\times\alpha\to\alpha)$. If $v[]$ does not may-diverge and there exist a type τ and a value $u\in\mathbf{Val}(\tau\times\tau)$ such that $v[]\,u$ may-diverges then for all types σ and for all values $w\in\mathbf{Val}(\sigma\times\sigma)$, $v[]\,w$ may-diverges and*

$$\emptyset\mid\emptyset\vdash v[]\,w =^{ctx}_{\Downarrow} \mathbf{\Omega}[] : \sigma.$$

◊

Proof. It is obvious that $\mathbf{\Omega}[]$ approximates $v[]\,w$ for any w. For the other approximation we use Theorem 3.B.29.

By the canonical forms lemma $u=\langle u_1,u_2\rangle$ for some $u_1,u_2\in\mathbf{Val}(\tau)$. Let $w\in\mathbf{Val}(\sigma\times\sigma)$. Again by the canonical forms lemma $w=\langle w_1,w_2\rangle$ for some $w_1,w_2\in\mathbf{Val}(\sigma)$. Now let $s=\{(w_1,u_1),(w_2,u_2)\}\in\mathbf{VRel}(\sigma,\tau)$. It is obvious that $(w, u) \in s$. Using Lemma 3.B.32 and the compatibility lemma for the application we have $(v[\,]w, v[\,]u) \in [\![\alpha \vdash \alpha]\!](s)^{\top\top}$.[7] Since the empty context is always related to itself and $v[\,]u\uparrow$ we have that $v[\,]w\underline{\uparrow}$. Since Lemma 3.B.32 is easily strengthened to a boxed one we thus have $v[\,]w\uparrow$ using Lemma 3.B.26.

We have thus established that $v[\,]w\uparrow$ for arbitrary w which implies that it CIU-approximates any other term and thus using Theorem 3.B.29 we conclude the proof. $\mathfrak{QED}$

For the rest of the cases we need some properties of the $\uparrow$ relation which we now prove.

Lemma 3.B.35. *Let $e \in \mathbf{Tm}(\tau)$. If $\neg(e\uparrow)$ then there exists a $v\mathbf{Val}(\tau)$, such that $e \rightsquigarrow^* v$.* $\Diamond$

Proof. We first prove by coinduction that the set

$$\mathcal{N} = \{e' \in \mathbf{Val}(\tau) \,\big|\, \forall v \in \mathbf{Val}(\tau), \neg(e' \rightsquigarrow^* v)\}$$

is included in $\uparrow$ using the universal property of $\uparrow$, i.e. that it is the greatest post-fixed point.

Given $e' \in \mathcal{N}$ we have to show there exists e'' such that $e' \rightsquigarrow e''$ and $e'' \in \mathcal{N}$. By the progress lemma e' is either a value or there exists an expression e'' that e' reduces to. Since $e' \in \mathcal{N}$ it cannot be a value. Thus there exists an expression e'' such that $e' \rightsquigarrow e''$. We have to show $e'' \in \mathcal{N}$. Suppose $v \in \mathbf{Val}(\tau)$ and $e'' \rightsquigarrow^* v$. Then $e' \rightsquigarrow^* v$. A contradiction.

Thus $\mathcal{N} \subseteq \uparrow$ and thus $\neg\uparrow \subseteq \neg\mathcal{N}$. But $e \in \mathcal{N} \leftrightarrow \forall v \in \mathbf{Val}(\tau), \neg(e' \rightsquigarrow^* v) \leftrightarrow \neg(\exists v \in \mathbf{Val}(\tau), e \rightsquigarrow^* v)$ and since we can show using properties of $\neg\neg$ that $\exists v \in \mathbf{Val}(\tau), e \rightsquigarrow^* v$ is $\neg\neg$-closed we have proved the lemma. $\mathfrak{QED}$

In the rest of this section we define $\mathbf{2} = \mathbf{1} + \mathbf{1}$ to be the type of booleans. We then write **true** $= \mathtt{inl}\,\langle\rangle$ and **false** $= \mathtt{inr}\,\langle\rangle$. By the canonical forms lemma these are the only two closed values of this type.

Lemma 3.B.36. *Let $E \in \mathbf{Stk}(\mathbf{2})$ and $e \in \mathbf{Tm}(\mathbf{2})$. Assume that $E[\mathbf{false}]\uparrow$ and $\neg(E[\mathbf{true}]\uparrow)$. In this case if $\neg(e \rightsquigarrow^* \mathbf{false})$ and $E[e]\uparrow$ then $e\uparrow$.* $\Diamond$

Proof. We prove this by coinduction. Let

$$\mathcal{N} = \{e \in \mathbf{Tm}(\mathbf{2}) \mid \neg(e \rightsquigarrow^* \mathbf{false}) \wedge E[e]\uparrow\}$$

and we wish to show that $\mathcal{N} \subseteq \uparrow$. Suppose $e \in \mathcal{N}$. We need to exhibit an e', such that $e \rightsquigarrow e'$ and $e' \in \mathcal{N}$. By the progress lemma e is either a value or steps to some e'. First we observe that e cannot be a value since the only two values are **true** and **false**. If $e = \mathbf{false}$ then $e \rightsquigarrow^* \mathbf{false}$ and if $e = \mathbf{true}$ then $E[e]\uparrow$ does not hold by assumption on E.

[7] We abused the notation by writing s instead of a function that maps φ to (σ, τ, s), but the meaning is clear.

So e is not a value. By assumption $E[e]\uparrow$ and so there exists e'', such that $E[e] \rightsquigarrow e''$ and $e''\underline{\uparrow}$. Since e is not a value we have $e'' = E[e']$ for some e' such that $e \rightsquigarrow e'$. For the first condition, suppose $e' \rightsquigarrow^* \textbf{false}$. Then clearly $e \rightsquigarrow^* \textbf{false}$, a contradiction.

We have thus exhibited an e', such that $e \rightsquigarrow e'$ and $e' \in \mathcal{N}$, thus concluding the proof. $\mathfrak{QED}$

Naturally we can exchange the roles of **true** and **false** in the last lemma. We record the next lemma for reference. The proof is by simple coinduction.

Lemma 3.B.37. *If* $e'\uparrow$ *and* $e \rightsquigarrow^* e'$ *then* $e\uparrow$. $\Diamond$

We now prove the converse of Lemma 3.B.26 for well-typed expressions.

Lemma 3.B.38. *Let* $\tau \in \textbf{Type}$ *and* $e \in \textbf{Tm}(\tau)$*. Then* $e\uparrow \to \square(e\underline{\uparrow})$. $\Diamond$

Proof. Using the fundamental theorem and the fact that $\square((-,-) \in [\![\emptyset \vdash \tau]\!]^{\top})$ holds we have that $\square(e\uparrow \to e\underline{\uparrow})$ which implies the lemma since $\square$ distributes over implication in the correct direction. $\mathfrak{QED}$

Note that we cannot directly use Löb induction to prove that $e\uparrow \to e\underline{\uparrow}$ since we use two different step relations in the definitions of $\uparrow$ and $\underline{\uparrow}$.

We record the functional extensionality property for values of the function type. The proof is the same as in [23] so we omit it.

Lemma 3.B.39. *Let* $\tau, \sigma \in \textbf{Type}, f, g \in \textbf{Val}(()\,\tau \to \sigma)$ *and assume*

$$\forall u \in \textbf{Val}(\tau), \emptyset \mid \emptyset \vdash f\, u =^{ctx}_{\Downarrow} g\, u : \sigma.$$

Then

$$\emptyset \mid \emptyset \vdash f =^{ctx}_{\Downarrow} g : \tau \to \sigma.$$

$\Diamond$

We now have all the ingredients to prove the last case in the characterisation of the values of type $\forall \alpha.\alpha \times \alpha \to \alpha$.

Lemma 3.B.40. *Let* $v \in \textbf{Val}(\forall \alpha.\alpha \times \alpha \to \alpha)$*. Suppose that for all* τ *and for all* $u \in \textbf{Val}(\tau \times \tau)$, $\neg(v[]\,u\uparrow)$*. Then one of the following three cases holds.*

1. $\forall \tau \in \textbf{Type}, \forall x, y \in \textbf{Val}(\tau), \emptyset \mid \emptyset \vdash v[]\langle x, y\rangle =^{ctx}_{\Downarrow} y : \tau$

2. $\forall \tau \in \textbf{Type}, \forall x, y \in \textbf{Val}(\tau), \emptyset \mid \emptyset \vdash v[]\langle x, y\rangle =^{ctx}_{\Downarrow} y : \tau$

3. $\forall \tau \in \textbf{Type}, \forall x, y \in \textbf{Val}(\tau), \emptyset \mid \emptyset \vdash v[]\langle x, y\rangle =^{ctx}_{\Downarrow} x \;\mathtt{or}\; y : \tau$

where $x \;\mathtt{or}\; y$ *is the binary choice expression* $\mathtt{case}(\mathtt{unfold?}, \_.x, \_.y)$. $\Diamond$

Proof. By Lemma 3.B.32 and the compatibility for application we have that for any $\tau \in \mathbf{Type}$,

$$\forall r \in \mathbf{VRel}(2,\tau), \forall (b,w) \in r \times r, (v[\,]\, b, v[\,]\, w) \in r^{\top\top} \tag{3.4}$$

and

$$\forall s \in \mathbf{VRel}(\tau,2), \forall (w,b) \in s \times s, (v[\,]\, w, v[\,]\, b) \in s^{\top\top} \tag{3.5}$$

where we write $r \times r$ and $s \times s$ for the construction on value relations used to interpret product types.

We consider 4 cases. In all cases let $x, y \in \mathbf{Val}(\tau)$ and let

$$s = \{(x, \mathbf{true}), (y, \mathbf{false})\} \in \mathbf{VRel}(\tau, 2)$$

and

$$r = \{(\mathbf{true}, x), (\mathbf{false}, y)\} \in \mathbf{VRel}(2, \tau).$$

- Suppose $v[\,]\langle \mathbf{true}, \mathbf{false}\rangle \rightsquigarrow^* \mathbf{true}$ and $v[\,]\langle \mathbf{true}, \mathbf{false}\rangle \rightsquigarrow^* \mathbf{false}$. We will show that in this case $\forall \tau \in \mathbf{Type}, \forall x, y \in \mathbf{Val}(\tau), \emptyset \mid \emptyset \vdash v[\,]\langle x, y\rangle =^{ctx}_{\Downarrow} x \texttt{ or } y : \tau$ and we do this by establishing CIU-equivalence. We prove two approximations.
  - Take a well-typed evaluation context E and assume $E[x \texttt{ or } y]\uparrow$. We need to show that $E[v[\,]\langle x, y\rangle]\uparrow$. $E[x \texttt{ or } y]\uparrow$ implies that at least one of $E[x]\uparrow$, $E[y]\uparrow$ holds. Without loss of generality suppose that $E[x]\uparrow$. Lemma 3.B.38 implies that $(E, (\lambda x.\texttt{if}\ z\ \texttt{then}\ \mathbf{\Omega}[\,]\ \texttt{else}\ z)\ -) \in [\![s]\!]^{\top}$. Using Lemma 3.B.37 and the assumption that

    $$v[\,]\langle \mathbf{true}, \mathbf{false}\rangle \rightsquigarrow^* \mathbf{true}$$

    we have that

    $$(\lambda x.\texttt{if}\ z\ \texttt{then}\ \mathbf{\Omega}[\,]\ \texttt{else}\ z)(v[\,]\langle \mathbf{true}, \mathbf{false}\rangle)\uparrow.$$

    Hence we have from (3.5) that $E[v[\,]\langle x, y\rangle]\underline{\uparrow}$ which we can improve to $E[v[\,]\langle x, y\rangle]\uparrow$ using Lemma 3.B.26 and the fact that we only used constant properties to prove it (in particular, s and r are $\neg\neg$-closed relations).
  - Take a well-typed evaluation context E and assume $E[v[\,]\langle x, y\rangle]\uparrow$. We need to show $E[x \texttt{ or } y]\uparrow$ and using Lemma 3.B.37 it suffices to show that either $E[x]\uparrow$ or $E[y]\uparrow$. Assume for the sake of contradiction that the negation holds. Since in intuitionistic logic $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$ holds we have that neither of $E[x]$ and $E[y]$ may-diverges. This together with the assumption $E[v[\,]\langle x, y\rangle]\uparrow$ means that $(-, E) \in r^{\top}$. Using (3.4) and Lemma 3.B.26 this implies that $v[\,]\langle \mathbf{true}, \mathbf{false}\rangle\uparrow$, contradicting the assumption of the lemma. Thus we have proved $\neg\neg E[x \texttt{ or } y]\uparrow$ and since may-divergence is $\neg\neg$-closed also $E[x \texttt{ or } y]\uparrow$.

- Suppose $v[]\langle\mathbf{true},\mathbf{false}\rangle \rightsquigarrow^* \mathbf{true}$ and not $v[]\langle\mathbf{true},\mathbf{false}\rangle \rightsquigarrow^* \mathbf{false}$. We will show that in this case $\forall\tau \in \mathbf{Type}, \forall x,y \in \mathbf{Val}(\tau), \emptyset \mid \emptyset \vdash v[]\langle x,y\rangle =^{ctx}_{\Downarrow} x : \tau$ and we again do this by establishing CIU-equivalence using two approximations.
  - Take a well-typed evaluation context E and assume that $E[x]\uparrow$. We are to show $E[v[]\langle x,y\rangle]\uparrow$. $E[x]\uparrow$ implies, using Lemma 3.B.38, that $(E, (\lambda z.\mathtt{if}\ z\ \mathtt{then}\ \mathbf{\Omega}[]\ \mathtt{else}\ z)\ -) \in s^{\top}$. Using (3.5) and Lemma 3.B.37 we thus have that $E[v[]\langle x,y\rangle]\uparrow$. Note that in this direction we have not used the assumption that $v[]\langle\mathbf{true},\mathbf{false}\rangle$ does not evaluate to **false**. We shall need it in the other direction, however.
  - Take a well-typed evaluation context E and assume $E[v[]\langle x,y\rangle]\uparrow$. We are to show $E[x]\uparrow$. Assume the converse for the sake of contradiction. This then means that $((\lambda z.\mathtt{if}\ z\ \mathtt{then}\ z\ \mathtt{else}\ \mathbf{\Omega}[])\ -, E) \in r^{\top}$. Using this and (3.4) we have that

    $$(\lambda z.\mathtt{if}\ z\ \mathtt{then}\ z\ \mathtt{else}\ \mathbf{\Omega}[])\, v[]\langle\mathbf{true},\mathbf{false}\rangle\uparrow.$$

    We now use Lemma 3.B.36 to conclude that $v[]\langle\mathbf{true},\mathbf{false}\rangle\uparrow$ (note that here is the place where we used the assumption that

    $$v[]\langle\mathbf{true},\mathbf{false}\rangle$$

    does not reduce to **false**). However this contradicts the assumption that $v[]\langle\mathbf{true},\mathbf{false}\rangle$ does not may-diverge. We have thus established $\neg\neg(E[x]\uparrow)$ and so $E[x]\uparrow$.
- Suppose $v[]\langle\mathbf{true},\mathbf{false}\rangle \rightsquigarrow^* \mathbf{false}$ and not $v[]\langle\mathbf{true},\mathbf{false}\rangle \rightsquigarrow^* \mathbf{true}$. In this case $\forall\tau \in \mathbf{Type}, \forall x,y \in \mathbf{Val}(\tau), \emptyset \mid \emptyset \vdash v[]\langle x,y\rangle =^{ctx}_{\Downarrow} y : \tau$. The proof is completely analogous to the previous case so we omit the details.
- Suppose $v[]\langle\mathbf{true},\mathbf{false}\rangle$ evaluates to neither **true** nor **false**. We claim that this case is impossible. Indeed, by assumption $v[]\langle\mathbf{true},\mathbf{false}\rangle$ does not may-diverge. By Lemma 3.B.35 there is a value $z \in \mathbf{Val}(2)$ such that $v[]\langle\mathbf{true},\mathbf{false}\rangle \rightsquigarrow^* z$. However by the canonical forms lemma we z must be either **true** or **false**. A contradiction.

We claim that the four cases we have considered cover everything. As a consequence of Lemma 3.A.27 we have for any two expressions e and e', that either $e \rightsquigarrow^* e'$ or $\neg(e \rightsquigarrow^* e')$. In particular we have $v[]\langle\mathbf{true},\mathbf{false}\rangle \rightsquigarrow^* \mathbf{false}$ or not and $v[]\langle\mathbf{true},\mathbf{false}\rangle \rightsquigarrow^* \mathbf{true}$ or not, which give exactly the four cases we have considered. 𝔔𝔈𝔇

3.C View From the Outside

We now sketch the interpretation of the types and relations defined in the internal language of $\mathbf{Sh}(\omega_1)$ in the category **Set**.

Interpretation of the model

Types The set of terms, types and evaluation contexts can be constructed as initial algebras of polynomial functors, hence are preserved by Δ, so $\mathbf{Val} = \Delta(\underline{\mathbf{Val}})$, $\mathbf{Tm} = \Delta(\underline{\mathbf{Tm}})$ and $\Delta(\underline{\mathbf{Stk}})$ (here $\underline{\mathbf{Tm}}$, $\underline{\mathbf{Stk}}$ and $\underline{\mathbf{Val}}$ are sets of expressions, evaluation contexts and values defined in **Set**). Moreover, since the initial algebra structure is preserved, in particular catamorphisms are preserved. Thus, functions from constant sets to constant sets "defined by induction" on, say **Stk** are the ones coming from **Set**.

Predicates The basic evaluation relation $\longmapsto$ can be defined by a simple case analysis on the set of closed expressions. More precisely it can be defined in the geometric fragment of first-order logic as a predicate on constant sets, since the type of closed terms is constant and Δ preserves products. Thus if $\underline{\longmapsto}$ is the basic one-step relation defined in **Set**, then $\longmapsto = \Delta(\underline{\longmapsto})$. The one-step reduction relation $\rightsquigarrow$ can be defined using $\longmapsto$ using a function from evaluation contexts and closed expressions to closed expressions, which "plugs the hole". This function can be defined by induction on the structure of the evaluation context, technically as a function from **Stk** to $\mathbf{Tm}^{\mathbf{Tm}}$ and since Δ also preserves exponentials, $\mathbf{Tm}^{\mathbf{Tm}}$ is constant. Thus this function arises from the analogous function in **Set**.

Thus $\rightsquigarrow$ is a constant predicate. The transitive closure of $\rightsquigarrow$, the relation $\rightsquigarrow^*$ is also constant by Lemma 3.A.27. Similarly, the relations $\overset{1}{\rightsquigarrow}$, $\overset{p}{\rightsquigarrow}$ ..., can be defined *positively* by starting with a smaller $\longmapsto$ relation and by relational composition. It is easy to see that composing two constant relations gives a constant relation. Thus, all the step relations are constant and equivalent to the inclusion by Δ of analogous relations defined in sets.

Interpretation of $\uparrow$ $\uparrow$ is defined internally as the greatest fixed point of Φ given as $\Phi(m) = \{e : \mathbf{Tm} \mid \exists e', e \rightsquigarrow e' \wedge m(e')\}$. Thus it satisfies

$$\forall e : \mathbf{Tm}, e\uparrow \leftrightarrow \exists e', e \rightsquigarrow e' \wedge e'\uparrow$$

and is the largest predicate that satisfies this formula. We will now show that $\uparrow = \Delta\underline{\uparrow}$, where $\underline{\uparrow}$ is the may-divergence relation defined in **Set**. We use Kripke-Joyal forcing semantics.

Suppose ν is a successor ordinal. Let $e \in \mathbf{Tm}(\nu)$. Thus $e \in \underline{\mathbf{Tm}}$ and by Kripke-Joyal we have

$$\nu \Vdash e\uparrow \text{ iff } \exists e' \in \underline{\mathbf{Tm}}, \nu \Vdash e \rightsquigarrow e' \text{ and } \nu \Vdash e'\uparrow$$

As we described above, $\nu \Vdash e \rightsquigarrow e'$ if and only if $e \underline{\rightsquigarrow} e'$ this implies that at each successor ordinal[8] ν, $\uparrow(\nu)$ is a fixed point of

$$\Phi'(S) = \left\{e : \underline{\mathbf{Tm}} \,\middle|\, \exists e', e \rightsquigarrow e' \wedge e' \in S\right\}$$

[8] To see the need for assuming that ν is a successor ordinal see the Kripke-Joyal semantics of existentials for limit ordinals.

defined in **Set**. Since $\underline{\uparrow}$ is defined as the greatest fixed point of Φ' we have that for all successor ordinals ν, $\uparrow(\nu) \subseteq \underline{\uparrow}$, thus $\uparrow \leq \Delta\left(\underline{\uparrow}\right)$. It is easy to see (same sequence of steps we did just now) that $\Delta\left(\underline{\uparrow}\right)$ is a fixed point of Φ. Hence, $\uparrow \geq \Delta\left(\underline{\uparrow}\right)$ and so $\uparrow = \Delta\left(\underline{\uparrow}\right)$.

It is easy to see that $\underline{\uparrow}$ is exactly the complement of the must-termination predicate $\Downarrow$ defined in [23].

Interpretation of the stratified may-divergence predicate The predicate ↥ is defined internally as the unique fixed point of Ψ given as

$$\Psi(m) = \left\{ e : \mathbf{Tm} \,\middle|\, \exists e', e \overset{1}{\rightsquigarrow} e' \wedge m(e') \right\}.$$

For a successor ordinal ν we thus have that

$$\nu \Vdash e\text{↥} \text{ iff } \exists e' \in \underline{\mathbf{Tm}}, \nu \Vdash e \overset{1}{\rightsquigarrow} e' \text{ and } \text{ for all } \beta < \nu, \beta \Vdash e'\text{↥}$$

Since for $\nu \geq 1$ $\nu \Vdash e \overset{1}{\rightsquigarrow} e'$ means that externally $e\underline{\overset{1}{\rightsquigarrow}}e'$ this is exactly the same as the definition of ↥ given in Section 3.B.

Thus,

$$\text{↥}(\nu) = \left\{ e \in \underline{\mathbf{Tm}} \,\middle|\, \exists e' \in \underline{\mathbf{Tm}}, e\underline{\overset{1}{\rightsquigarrow}}e' \wedge \forall \beta < \nu, e' \in \text{↥}(\beta) \right\}$$

which is exactly the pointwise negation of the stratified must-termination predicate $\{\Downarrow_\beta\}$ defined in [23], i.e. $\Downarrow^c_\beta = \text{↥}(\beta)$.

Proposition 3.C.1.

$$\bigcap_{\nu=1}^{\omega_1} \text{↥}(\beta) \subseteq \underline{\uparrow}$$

◊

Proof. Since $\Downarrow^c_\beta = \text{↥}(\beta)$ and $\underline{\uparrow} = \Downarrow^c$ we have

$$\bigcap_{\nu=1}^{\omega_1} \text{↥}(\beta) \subseteq \underline{\uparrow} \;\leftrightarrow\; \bigcap_{\nu=1}^{\omega_1} \Downarrow^c_\beta \subseteq \Downarrow^c \;\leftrightarrow\; \left(\bigcup_{\nu=1}^{\omega_1} \Downarrow_\beta \right)^c \subseteq \Downarrow^c \;\leftrightarrow\; \Downarrow \subseteq (\bigcup_{\nu=1}^{\omega_1} \Downarrow_\beta)$$

and the last inclusion holds by [23, Lemma 5.2] (to be completely precise we cannot immediately apply the same lemma, since we have a slightly different language, but the proof is exactly the same). QED

Note that this last proposition would not hold, were we to interpret the construction in the topos of trees $\mathcal{S}$, since we would only take the intersection of the first ω approximations. Thus, we need to work in the topos $\mathbf{Sh}(\omega_1)$.

As a consequence, we can add the following principle to our logic

$$e : \mathbf{Tm} \mid \emptyset \vdash \square(e\text{↥}) \to e\uparrow$$

which enables us to prove adequacy of the logical relation with respect to contextual must-approximation.

$\uparrow$ **is constant** Using Proposition 3.A.23 we have that $\uparrow$ is $\neg\neg$-closed. From this it follows that $\square\uparrow = \uparrow$, hence $\uparrow$ is constant.

Chapter 4

Programming and Reasoning with Guarded Recursion for Coinductive Types

This chapter is an extended version of

> Ranald Clouston, Aleš Bizjak, Hans Bugge Grathwohl, and Lars Birkedal.
>
> Programming and reasoning with guarded recursion for coinductive typeso.
>
> In *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015*, pages 407–421, 2015.

Abstract

We present the guarded lambda-calculus, an extension of the simply typed lambda-calculus with guarded recursive and coinductive types. The use of guarded recursive types ensures the productivity of well-typed programs. Guarded recursive types may be transformed into coinductive types by a type-former inspired by modal logic and Atkey-McBride clock quantification, allowing the typing of acausal functions. We give a call-by-name operational semantics for the calculus, and define adequate denotational semantics in the topos of trees. The adequacy proof entails that the evaluation of a program always terminates. We demonstrate the expressiveness of the calculus by showing the definability of solutions to Rutten's behavioural differential equations. We introduce a program logic with Löb induction for reasoning about the contextual equivalence of programs.

4.1 Introduction

The problem of ensuring that functions on coinductive types are well-defined has prompted a wide variety of work into productivity checking, and rule formats for coalgebra. *Guarded recursion* [34] guarantees productivity and unique solutions by requiring that recursive calls be nested under a constructor, such as cons (written ::) for streams. This can sometimes be established by a simple syntactic check, as for the stream toggle and binary stream function interleave below:

```
toggle = 1 :: 0 :: toggle
interleave (x :: xs) ys = x :: interleave ys xs
```

Such syntactic checks, however, are often too blunt and exclude many valid definitions. For example the *regular paperfolding sequence*, the sequence of left and right turns (encoded as 1 and 0) generated by repeatedly folding a piece of paper in half, can be defined via the function interleave as follows [43]:

```
paperfolds = interleave toggle paperfolds
```

This definition is productive, but the putative definition below, which also applies interleave to two streams and so apparently is just as well-typed, is not:

```
paperfolds' = interleave paperfolds' toggle
```

This equation is satisfied by any stream whose *tail* is the regular paperfolding sequence, so lacks a unique solution. Unfortunately the syntactic productivity checker of the proof assistant Coq [45] will reject both definitions.

A more flexible approach, first suggested by Nakano [72], is to guarantee productivity via *types*. A new modality, for which we follow Appel et al. [9] by writing ► and using the name 'later', allows us to distinguish between data we have access to now, and data which we have only later. This ► must be used to guard self-reference in type definitions, so for example *guarded streams* of natural numbers are defined by the guarded recursive equation

$$\mathsf{Str}^{\mathsf{g}} \triangleq \mathbf{N} \times \blacktriangleright \mathsf{Str}^{\mathsf{g}}$$

asserting that stream heads are available now, but tails only later. The type of interleave will be $\mathsf{Str}^{\mathsf{g}} \to \blacktriangleright\mathsf{Str}^{\mathsf{g}} \to \mathsf{Str}^{\mathsf{g}}$, capturing the fact the (head of the) first argument is needed immediately, but the second argument is needed only later. In term definitions the types of self-references will then be guarded by ► also. For example interleave paperfolds′ toggle becomes ill-formed, as the paperfolds′ self-reference has type $\blacktriangleright\mathsf{Str}^{\mathsf{g}}$, rather than $\mathsf{Str}^{\mathsf{g}}$, but

$$\mathsf{interleave}\ \mathsf{toggle}\ \mathsf{paperfolds}$$

will be well-formed.

Adding $\blacktriangleright$ alone to the simply typed λ-calculus enforces a discipline more rigid than productivity. For example the obviously productive stream function

```
every2nd (x :: x' :: xs) = x :: every2nd xs
```

cannot be typed because it violates *causality* [57]: elements of the result stream depend on deeper elements of the argument stream. In some settings, such as reactive programming, this is a desirable property, but for productivity guarantees alone it is too restrictive. We need the ability to remove $\blacktriangleright$ in a controlled way. This is provided by the *clock quantifiers* of Atkey and McBride [11], which assert that all data is available now. This does not trivialise the guardedness requirements because there are side-conditions controlling when clock quantifiers may be introduced. Moreover clock quantifiers transform guarded recursive types into first-class *coinductive* types, with guarded recursion defining the rule format for their manipulation.

Our presentation departs from Atkey and McBride's [11] by regarding the 'everything now' operator as a unary type-former, written $\blacksquare$ and called 'constant', rather than a quantifier. Observing that the types $\blacksquare A \to A$ and $\blacksquare A \to \blacksquare\blacksquare A$ are always inhabited allows us to see the type-former, via the Curry-Howard isomorphism, as an *S4* modality, and hence base our operational semantics on the established typed calculi for intuitionistic S4 (IS4) of Bierman and de Paiva [17]. This is sufficient to capture all examples in the literature, which use only one clock; for examples that require multiple clocks we suggest extending our calculus to a *multimodal* logic.

In this paper we present the guarded λ-calculus, gλ, extending the simply typed λ-calculus with coinductive and guarded recursive types. We define call-by-name operational semantics, which blocks non-termination via recursive definitions unfolding indefinitely. We define adequate denotational semantics in the topos of trees [22] and as a consequence prove normalisation. We introduce a program logic $Lg\lambda$ for reasoning about the denotations of gλ-programs; given adequacy this permits proofs about the operational behaviour of terms. The logic is based on the internal logic of the topos of trees, with modalities $\triangleright, \square$ on predicates, and Löb induction for reasoning about functions on both guarded recursive and coinductive types. We demonstrate the expressiveness of the calculus by showing the definability of solutions to Rutten's behavioural differential equations [83], and show that $Lg\lambda$ can be used to reason about them, as an alternative to standard bisimulation-based arguments.

We have implemented the gλ-calculus in Agda, a process we found helpful when fine-tuning the design of our calculus. The implementation, with many examples, is available at http://cs.au.dk/~hbugge/gl-agda.zip.

4.2 Guarded λ-calculus

This section presents the guarded λ-calculus, written gλ, its call-by-name operational semantics, and its types, then gives some examples.

Definition 4.2.1. gλ-*terms* are given by the grammar

$$\begin{array}{rcl} t & ::= & x \mid \langle\rangle \mid \mathsf{zero} \mid \mathsf{succ}\,t \mid \langle t,t\rangle \mid \pi_d t \mid \lambda x.t \mid tt \mid \mathsf{fold}\,t \mid \mathsf{unfold}\,t \\ & \mid & \mathsf{next}\,t \mid \mathsf{prev}\,\sigma.t \mid \mathsf{box}\,\sigma.t \mid \mathsf{unbox}\,t \mid t \circledast t \end{array}$$

where $d \in \{1,2\}$, x is a variable and $\sigma = [x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]$, usually abbreviated $[\vec{x} \leftarrow \vec{t}]$, is a list of variables paired with terms.

$\mathsf{prev}[\vec{x} \leftarrow \vec{t}].t$ and $\mathsf{box}[\vec{x} \leftarrow \vec{t}].t$ bind all variables of $\vec{x}$ in t, but *not* in $\vec{t}$. We write $\mathsf{prev}\,\iota.t$ for $\mathsf{prev}[\vec{x} \leftarrow \vec{x}].t$ where $\vec{x}$ is a list of all free variables of t. If furthermore t is closed we simply write $\mathsf{prev}\,t$. We will similarly write $\mathsf{box}\,\iota.t$ and $\mathsf{box}\,t$. We adopt the convention that prev and box have highest precedence. ♦

We may extend gλ with sums; for space reasons we leave these to Appendix 4.C.

Definition 4.2.2. The *reduction rules* on closed gλ-terms are

$$\begin{array}{rcll} \pi_d\langle t_1,t_2\rangle & \mapsto & t_d & (d \in \{1,2\}) \\ (\lambda x.t_1)t_2 & \mapsto & t_1[t_2/x] & \\ \mathsf{unfold}\,\mathsf{fold}\,t & \mapsto & t & \\ \mathsf{prev}[\vec{x} \leftarrow \vec{t}].t & \mapsto & \mathsf{prev}\,t[\vec{t}/\vec{x}] & (\vec{x}\ \textit{non-empty}) \\ \mathsf{prev}\,\mathsf{next}\,t & \mapsto & t & \\ \mathsf{unbox}(\mathsf{box}[\vec{x} \leftarrow \vec{t}].t) & \mapsto & t[\vec{t}/\vec{x}] & \\ \mathsf{next}\,t_1 \circledast \mathsf{next}\,t_2 & \mapsto & \mathsf{next}(t_1 t_2) & \end{array}$$

♦

The rules above look like standard β-reduction, removing ‘roundabouts’ of introduction then elimination, with the exception of those regarding prev and next. An apparently more conventional β-rule for these term-formers would be

$$\mathsf{prev}[\vec{x} \leftarrow \vec{t}].(\mathsf{next}\,t) \mapsto t[\vec{t}/\vec{x}]$$

but where $\vec{x}$ is non-empty this would require us to reduce an open term to derive $\mathsf{next}\,t$. We take the view that reduction of open terms is undesirable within a call-by-name discipline, so first apply the substitution without eliminating prev.

The final rule is not a true β-rule, as $\circledast$ is neither introduction nor elimination, but is necessary to enable function application under a next and hence allow, for example, manipulation of the tail of a stream. It corresponds to the ‘homomorphism’ equality for applicative functors [69].

We next impose our call-by-name strategy on these reductions.

$$\frac{}{\nabla, \alpha \vdash \alpha} \qquad \frac{}{\nabla \vdash \mathbf{1}} \qquad \frac{}{\nabla \vdash \mathbf{N}} \qquad \frac{\nabla \vdash A_1 \qquad \nabla \vdash A_2}{\nabla \vdash A_1 \times A_2} \qquad \frac{\nabla \vdash A_1 \qquad \nabla \vdash A_2}{\nabla \vdash A_1 \to A_2}$$

$$\frac{\nabla, \alpha \vdash A}{\nabla \vdash \mu\alpha.A}\ \alpha\,\mathsf{guarded\,in}\,A \qquad \frac{\nabla \vdash A}{\nabla \vdash \blacktriangleright A} \qquad \frac{\cdot \vdash A}{\nabla \vdash \blacksquare A}$$

Figure 4.1: Type formation for the gλ-calculus

Definition 4.2.3. *Values* are terms of the form

$$\langle\rangle \mid \mathsf{succ}^n\,\mathsf{zero} \mid \langle t, t\rangle \mid \lambda x.t \mid \mathsf{fold}\,t \mid \mathsf{box}\,\sigma.t \mid \mathsf{next}\,t$$

where succ^n is a list of zero or more succ operators, and t is any term. ♦

Definition 4.2.4. *Evaluation contexts* are defined by the grammar

$$E \quad ::= \quad \cdot \mid \mathsf{succ}\,E \mid \pi_d E \mid E t \mid \mathsf{unfold}\,E \mid \mathsf{prev}\,E \mid \mathsf{unbox}\,E \mid E \circledast t \mid v \circledast E$$

♦

If we regard $\circledast$ as a variant of function application, it is surprising in a call-by-name setting to reduce on both its sides. However both sides must be reduced until they have main connective next before the reduction rule for $\circledast$ may be applied. Thus the order of reductions of gλ-terms cannot be identified with the call-by-name reductions of the corresponding λ-calculus term with the novel connectives erased.

Definition 4.2.5. *Call-by-name reduction* has format $E[t] \mapsto E[u]$, where $t \mapsto u$ is a reduction rule. From now the symbol $\mapsto$ will be reserved to refer to call-by-name reduction. We use $\rightsquigarrow$ for the reflexive transitive closure of $\mapsto$. ♦

Lemma 4.2.6. *The call-by-name reduction relation $\mapsto$ is deterministic.* ◊

Definition 4.2.7. gλ*-types* are defined inductively by the rules of Figure 4.1. ∇ is a finite set of *type variables*. A variable α is *guarded in* a type A if all occurrences of α are beneath an occurrence of $\blacktriangleright$ in the syntax tree. We adopt the convention that unary type-formers bind closer than binary type-formers. ♦

Note the side condition on the μ type-former, and the prohibition on $\blacksquare A$ for open A, which can also be understood as a prohibition on applying $\mu\alpha$ to any α with $\blacksquare$ above it. The intuition for these restrictions is that unique fixed points exist only where the variable is displaced in time by a $\blacktriangleright$, but $\blacksquare$ cancels out this displacement by giving 'everything now'.

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{}{\Gamma \vdash \langle\rangle : \mathbf{1}} \qquad \frac{}{\Gamma \vdash \mathsf{zero} : \mathbf{N}} \qquad \frac{\Gamma \vdash t : \mathbf{N}}{\Gamma \vdash \mathsf{succ}\, t : \mathbf{N}}$$

$$\frac{\Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : B}{\Gamma \vdash \langle t_1, t_2 \rangle : A \times B} \qquad \frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \pi_d t : A_d} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}$$

$$\frac{\Gamma \vdash t_1 : A \to B \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1 t_2 : B} \qquad \frac{\Gamma \vdash t : A[\mu\alpha.A/\alpha]}{\Gamma \vdash \mathsf{fold}\, t : \mu\alpha.A} \qquad \frac{\Gamma \vdash t : \mu\alpha.A}{\Gamma \vdash \mathsf{unfold}\, t : A[\mu\alpha.A/\alpha]}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{next}\, t : \blacktriangleright A} \qquad \frac{\begin{array}{c} x_1 : A_1, \ldots, x_n : A_n \vdash t : \blacktriangleright A \\ \Gamma \vdash t_1 : A_1 \quad \cdots \quad \Gamma \vdash t_n : A_n \end{array}}{\Gamma \vdash \mathsf{prev}[x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n].t : A}\ A_1, \ldots, A_n\ \mathsf{constant}$$

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash t : A \qquad \Gamma \vdash t_1 : A_1 \quad \cdots \quad \Gamma \vdash t_n : A_n}{\Gamma \vdash \mathsf{box}[x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n].t : \blacksquare A}\ A_1, \ldots, A_n\ \mathsf{constant}$$

$$\frac{\Gamma \vdash t : \blacksquare A}{\Gamma \vdash \mathsf{unbox}\, t : A} \qquad \frac{\Gamma \vdash t_1 : \blacktriangleright(A \to B) \qquad \Gamma \vdash t_2 : \blacktriangleright A}{\Gamma \vdash t_1 \circledast t_2 : \blacktriangleright B}$$

Figure 4.2: Typing rules for the gλ-calculus

Definition 4.2.8. The *typing judgments* are given in Figure 4.2. There $d \in \{1, 2\}$, and the *typing contexts* Γ are finite sets of pairs $x : A$ where x is a variable and A a closed type. Closed types are *constant* if all occurrences of $\blacktriangleright$ are beneath an occurrence of $\blacksquare$ in their syntax tree. ♦

The *constant* types exist 'all at once', due to the absence of $\blacktriangleright$ or presence of $\blacksquare$; this condition corresponds to the freeness of the clock variable in Atkey and McBride [11] (recalling that we use only one clock in this work). Its use as a side-condition to $\blacksquare$-introduction in Figure 4.2 recalls (but is more general than) the 'essentially modal' condition for natural deduction for IS4 of Prawitz [80]. The term calculus for IS4 of Bierman and de Paiva [17], on which this calculus is most closely based, uses the still more restrictive requirement that $\blacksquare$ be the main connective. This would preclude some functions that seem desirable, such as the isomorphism $\lambda n.\, \mathsf{box}\, \iota.n : \mathbf{N} \to \blacksquare\mathbf{N}$.

In examples prev usually appears in its syntactic sugar forms

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash t : \blacktriangleright A}{\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash \mathsf{prev}\, \iota.t : A}\ A_1, \ldots, A_n\ \mathsf{constant} \qquad \frac{\vdash t : \blacktriangleright A}{\Gamma \vdash \mathsf{prev}\, t : A}$$

and similarly for box; the more general form is nonetheless necessary because

$(\mathsf{prev}\,\iota.t)[\vec{u}/\vec{x}] = \mathsf{prev}[\vec{x} \leftarrow \vec{u}].t$. Getting substitution right in this setting is somewhat delicate. For example our reduction rule $\mathsf{prev}[\vec{x} \leftarrow \vec{t}].t \mapsto \mathsf{prev}\,t[\vec{t}/\vec{x}]$ breaches subject reduction on open terms (but not for closed terms). See Bierman and de Paiva [17] for more discussion of substitution with respect to IS4.

Lemma 4.2.9 (Subject Reduction). *$\vdash t : A$ and $t \rightsquigarrow u$ implies $\vdash u : A$.* ◊

Example 4.2.10.

(i) The type of guarded recursive streams of natural numbers, $\mathsf{Str}^{\mathsf{g}}$, is defined as $\mu\alpha.\mathbf{N}\times\blacktriangleright\alpha$. These provide the setting for all examples below, but other definable types include infinite binary trees, as $\mu\alpha.\mathbf{N}\times\blacktriangleright\alpha\times\blacktriangleright\alpha$, and potentially infinite lists, as $\mu\alpha.\mathbf{1}+(\mathbf{N}\times\blacktriangleright\alpha)$.

(ii) We define guarded versions of the stream functions cons (written infix as ::), head, and tail as obvious:

$$:: \triangleq \lambda n.\lambda s.\,\mathsf{fold}\langle n,s\rangle : \mathbf{N} \to \blacktriangleright\mathsf{Str}^{\mathsf{g}} \to \mathsf{Str}^{\mathsf{g}}$$
$$\mathsf{hd}^{\mathsf{g}} \triangleq \lambda s.\pi_1\,\mathsf{unfold}\,s : \mathsf{Str}^{\mathsf{g}} \to \mathbf{N} \qquad \mathsf{tl}^{\mathsf{g}} \triangleq \lambda s.\pi_2\,\mathsf{unfold}\,s :: \mathsf{Str}^{\mathsf{g}} \to \blacktriangleright\mathsf{Str}^{\mathsf{g}}$$

then use the ⊛ term-former for observations deeper into the stream:

$$\begin{aligned}\mathsf{2nd}^{\mathsf{g}} &\triangleq \lambda s.(\mathsf{next}\,\mathsf{hd}^{\mathsf{g}}) \circledast (\mathsf{tl}^{\mathsf{g}}\,s) : \mathsf{Str}^{\mathsf{g}} \to \blacktriangleright\mathbf{N}\\ \mathsf{3rd}^{\mathsf{g}} &\triangleq \lambda s.(\mathsf{next}\,\mathsf{2nd}^{\mathsf{g}}) \circledast (\mathsf{tl}^{\mathsf{g}}\,s) : \mathsf{Str}^{\mathsf{g}} \to \blacktriangleright\blacktriangleright\mathbf{N}\ \cdots\end{aligned}$$

(iii) Following Abel and Vezzosi [2, Sec. 3.4] we may define a fixed point combinator fix with type $(\blacktriangleright A \to A) \to A$ for any A. We use this to define a stream by iteration of a function: iterate takes as arguments a natural number and a function, but the function is not used until the 'next' step of computation, so we may reflect this with our typing:

$$\mathsf{iterate} \triangleq \lambda f.\,\mathsf{fix}\,\lambda g.\lambda n.n :: (g \circledast (f \circledast \mathsf{next}\,n)) : \blacktriangleright(\mathbf{N} \to \mathbf{N}) \to \mathbf{N} \to \mathsf{Str}^{\mathsf{g}}$$

We may hence define the guarded stream of natural numbers

$$\mathsf{nats} \triangleq \mathsf{iterate}\,(\mathsf{next}\,\lambda n.\,\mathsf{succ}\,n)\,\mathsf{zero}.$$

(iv) With interleave, following our discussion in the introduction, we again may reflect in our type that one of our arguments is not required until the next step, defining the term interleave as:

$$\mathsf{fix}\,\lambda g.\lambda s.\lambda t.(\mathsf{hd}^{\mathsf{g}}\,s) :: (g \circledast t \circledast \mathsf{next}(\mathsf{tl}^{\mathsf{g}}\,s)) : \mathsf{Str}^{\mathsf{g}} \to \blacktriangleright\mathsf{Str}^{\mathsf{g}} \to \mathsf{Str}^{\mathsf{g}}$$

This typing decision is essential to define the paper folding stream:

$$\begin{aligned}\mathsf{toggle} &\triangleq \mathsf{fix}\,\lambda s.(\mathsf{succ}\,\mathsf{zero}) :: (\mathsf{next}(\mathsf{zero} :: s))\\ \mathsf{paperfolds} &\triangleq \mathsf{fix}\,\lambda s.\,\mathsf{interleave}\,\mathsf{toggle}\,s\end{aligned}$$

Note that the unproductive definition with $\mathsf{interleave}\, s\, \mathsf{toggle}$ cannot be made to type check: informally, $s : \blacktriangleright\mathsf{Str}^{\mathsf{g}}$ cannot be converted into a $\mathsf{Str}^{\mathsf{g}}$ by prev, as it is in the scope of a variable s whose type $\mathsf{Str}^{\mathsf{g}}$ is not constant. To see a less artificial non-example, try to define a filter function on streams which eliminates elements that fail some boolean test.

(v) μ-types are in fact *unique* fixed points, so carry both final coalgebra and initial algebra structure. To see the latter, observe that we can define

$$\mathsf{foldr} \triangleq \mathsf{fix}\,\lambda g \lambda f.\lambda s.f\langle \mathsf{hd}^{\mathsf{g}}\, s, g \circledast \mathsf{next}\, f \circledast \mathsf{tl}^{\mathsf{g}}\, s\rangle : ((\mathbf{N}\times\blacktriangleright A)\to A)\to \mathsf{Str}^{\mathsf{g}} \to A$$

and hence for example $\mathsf{map}^{\mathsf{g}}\, h : \mathsf{Str}^{\mathsf{g}} \to \mathsf{Str}^{\mathsf{g}}$ is $\mathsf{foldr}\,\lambda x.(h\pi_1 x) :: (\pi_2 x)$.

(vi) The $\blacksquare$ type-former lifts guarded recursive streams to real coinductive streams, as we will make precise in Ex. 4.3.4. Let $\mathsf{Str} \triangleq \blacksquare\mathsf{Str}^{\mathsf{g}}$. We define

$$\mathsf{hd} : \mathsf{Str} \to \mathbf{N}\text{and} \qquad\qquad \mathsf{tl} : \mathsf{Str} \to \mathsf{Str}$$

by

$$\mathsf{hd} = \lambda s.\, \mathsf{hd}^{\mathsf{g}}(\mathsf{unbox}\, s)$$

and

$$\mathsf{tl} = \lambda s.\, \mathsf{box}\, \iota.\, \mathsf{prev}\, \iota.\, \mathsf{tl}^{\mathsf{g}}(\mathsf{unbox}\, s),$$

and hence define observations deep into streams whose results bear no trace of $\blacktriangleright$, for example $\mathsf{2nd} \triangleq \lambda s.\, \mathsf{hd}(\mathsf{tl}\, s) : \mathsf{Str} \to \mathbf{N}$.

In general boxed functions lift to functions on boxed types by

$$\mathsf{lim} \triangleq \lambda f.\lambda x.\, \mathsf{box}\, \iota.(\mathsf{unbox}\, f)(\mathsf{unbox}\, x) : \blacksquare(A \to B) \to \blacksquare A \to \blacksquare B$$

(vii) The more sophisticated acausal function $\mathsf{every2nd} : \mathsf{Str} \to \mathsf{Str}^{\mathsf{g}}$ is

$$\mathsf{fix}\,\lambda g.\lambda s.(\mathsf{hd}\, s) :: (g \circledast (\mathsf{next}(\mathsf{tl}(\mathsf{tl}\, s)))).$$

Note that it must take a *coinductive* stream Str as argument. The function with coinductive result type is then $\lambda s.\, \mathsf{box}\, \iota.\, \mathsf{every2nd}\, s : \mathsf{Str} \to \mathsf{Str}$.

♦

4.3 Denotational Semantics and Normalisation

This section gives denotational semantics for gλ-types and terms, as objects and arrows in the topos of trees [22], the presheaf category over the first infinite ordinal ω (we give a concrete definition below). These semantics are shown to be sound and, by a logical relations argument, adequate with respect to the operational semantics. Normalisation follows as a corollary of this argument. Note that for space reasons many proofs, and some lemmas, appear in Appendix 4.A.

Definition 4.3.1. The *topos of trees* $\mathcal{S}$ has, as objects X, families of sets $X_1, X_2, \ldots$ indexed by the positive integers, equipped with families of *restriction functions* $r_i^X : X_{i+1} \to X_i$ indexed similarly. Arrows $f : X \to Y$ are families of functions $f_i : X_i \to Y_i$ indexed similarly obeying the naturality condition $f_i \circ r_i^X = r_i^Y \circ f_{i+1}$. ♦

$\mathcal{S}$ is a cartesian closed category with products defined pointwise. Its exponential A^B has, as its component sets $(A^B)_i$, the set of i-tuples $(f_1 : A_1 \to B_1, \ldots, f_i : A_i \to B_i)$ obeying the naturality condition, and projections as restriction functions.

Definition 4.3.2.

- The category of sets **Set** is a full subcategory of $\mathcal{S}$ via the functor $\Delta : \mathbf{Set} \to \mathcal{S}$ with $(\Delta Z)_i = Z$, $r_i^{\Delta Z} = id_Z$, and $(\Delta f)_i = f$. Objects in this subcategory are called *constant objects*. In particular the terminal object 1 of $\mathcal{S}$ is $\Delta\{*\}$ and the *natural numbers object* is $\Delta\mathbb{N}$;
- Δ is left adjoint to $hom_{\mathcal{S}}(1,-)$; write $\blacksquare$ for $\Delta \circ hom_{\mathcal{S}}(1,\text{-}) : \mathcal{S} \to \mathcal{S}$. $\mathsf{unbox} : \blacksquare \stackrel{.}{\to} id_{\mathcal{S}}$ is the counit of the resulting comonad. Concretely $\mathsf{unbox}_i(x) = x_i$, i.e. the i'th component of $x : 1 \to X$ applied to $*$;
- $\blacktriangleright : \mathcal{S} \to \mathcal{S}$ is defined by $(\blacktriangleright X)_1 = \{*\}$ and $(\blacktriangleright X)_{i+1} = X_i$, with $r_1^{\blacktriangleright X}$ defined uniquely and $r_{i+1}^{\blacktriangleright X} = r_i^X$. Its action on arrows $f : X \to Y$ is $(\blacktriangleright f)_1 = id_{\{*\}}$ and $(\blacktriangleright f)_{i+1} = f_i$. The natural transformation $\mathsf{next} : id_{\mathcal{S}} \stackrel{.}{\to} \blacktriangleright$ has next_1 unique and $\mathsf{next}_{i+1} = r_i^X$ for any X.

♦

Definition 4.3.3. We interpet types in context $\nabla \vdash A$, where ∇ contains n free variables, as functors $[\![\nabla \vdash A]\!] : (\mathcal{S}^{op} \times \mathcal{S})^n \to \mathcal{S}$, usually written $[\![A]\!]$. This mixed variance definition is necessary as variables may appear negatively or positively.

- $[\![\nabla, \alpha \vdash \alpha]\!]$ is the projection of the objects or arrows corresponding to *positive* occurrences of α, e.g. $[\![\alpha]\!](\vec{W}, X, Y) = Y$;
- $[\![\mathbf{1}]\!]$ and $[\![\mathbf{N}]\!]$ are the constant functors $\Delta\{*\}$ and $\Delta\mathbb{N}$ respectively;
- $[\![A_1 \times A_2]\!](\vec{W}) = [\![A_1]\!](\vec{W}) \times [\![A_2]\!](\vec{W})$ and likewise for $\mathcal{S}$-arrows;
- $[\![A_1 \to A_2]\!](\vec{W}) = [\![A_2]\!](\vec{W})^{[\![A_2]\!](\vec{W}')}$ where $\vec{W}'$ is $\vec{W}$ with odd and even elements switched to reflect change in polarity, i.e.
$$(X_1, Y_1, \ldots)' = (Y_1, X_1, \ldots);$$
- $[\![\blacktriangleright A]\!], [\![\blacksquare A]\!]$ are defined by composition with the functors $\blacktriangleright, \blacksquare$ (Definition 4.3.2).

- $[\![\mu\alpha.A]\!](\vec{W}) = \mathsf{Fix}(F)$, where $F : (\mathcal{S}^{op} \times \mathcal{S}) \to \mathcal{S}$ is the functor given by $F(X,Y) = [\![A]\!](\vec{W},X,Y)$ and $\mathsf{Fix}(F)$ is the unique (up to isomorphism) X such that $F(X,X) \cong X$. The existence of such X relies on F being a suitably locally contractive functor, which follows by Birkedal et al [22, Sec. 4.5] and the fact that $\blacksquare$ is only ever applied to closed types. This restriction on $\blacksquare$ is necessary because the functor $\blacksquare$ is not strong.

♦

Example 4.3.4. $[\![\mathsf{Str}^{\mathsf{g}}]\!]_i = \mathbb{N}^i$, with projections as restriction functions, so is an object of *approximations* of streams – first the head, then the first two elements, and so forth. $[\![\mathsf{Str}]\!]_i = \mathbb{N}^\omega$ at all levels, so is the constant object of streams. More generally, any polynomial functor F on **Set** can be assigned a gλ-type A_F with a free type variable α that occurs guarded. The denotation of $\blacksquare\mu\alpha.A_F$ is the constant object of the carrier of the final coalgebra for F [71, Theorem 2]. ♦

Lemma 4.3.5. *The interpretation of a recursive type is isomorphic to the interpretation of its unfolding:* $[\![\mu\alpha.A]\!](\vec{W}) \cong [\![A[\mu\alpha.A/\alpha]]\!](\vec{W})$. ◊

Lemma 4.3.6. *Closed constant types denote constant objects in* $\mathcal{S}$. ◊

Note that the converse does not apply; for example $[\![\blacktriangleright 1]\!]$ is a constant object.

Definition 4.3.7. We interpret typing contexts $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ as $\mathcal{S}$-objects $[\![\Gamma]\!] \triangleq [\![A_1]\!] \times \cdots \times [\![A_n]\!]$ and hence interpret typed terms-in-context $\Gamma \vdash t : A$ as $\mathcal{S}$-arrows $[\![\Gamma \vdash t : A]\!] : [\![\Gamma]\!] \to [\![A]\!]$ (usually written $[\![t]\!]$) as follows.

$[\![x]\!]$ is the projection $[\![\Gamma]\!] \times [\![A]\!] \to [\![A]\!]$. $[\![\mathsf{zero}]\!]$ and $[\![\mathsf{succ}\, t]\!]$ are as obvious. Term-formers for products and function spaces are interpreted via the cartesian closed structure of $\mathcal{S}$. Exponentials are not pointwise, so we give explicitly:

- $[\![\lambda x.t]\!]_i(\gamma)_j$ maps $a \mapsto [\![\Gamma, x : A \vdash t : B]\!]_j(\gamma\restriction_j, a)$, where $\gamma\restriction_j$ is the result of applying restriction functions to $\gamma \in [\![\Gamma]\!]_i$ to get an element of $[\![\Gamma]\!]_j$;
- $[\![t_1 t_2]\!]_i(\gamma) = ([\![t_1]\!]_i(\gamma)_i) \circ [\![t_2]\!]_i(\gamma)$;

$[\![\mathsf{fold}\, t]\!]$ and $[\![\mathsf{unfold}\, t]\!]$ are defined via composition with the isomorphisms of Lemma 4.3.5. $[\![\mathsf{next}\, t]\!]$ and $[\![\mathsf{unbox}\, t]\!]$ are defined by composition with the natural transformations introduced in Definition 4.3.2. The final three cases are

- $[\![\mathsf{prev}[x_1 \leftarrow t_1, \ldots].t]\!]_i(\gamma) \triangleq [\![t]\!]_{i+1}([\![t_1]\!]_i(\gamma), \ldots)$, where $[\![t_1]\!]_i(\gamma) \in [\![A_1]\!]_i$ is also in $[\![A_1]\!]_{i+1}$ by Lemma 4.3.6;
- $[\![\mathsf{box}[x_1 \leftarrow t_1, \ldots].t]\!]_i(\gamma)_j = [\![t]\!]_j([\![t_1]\!]_i(\gamma), \ldots)$, again using Lemma 4.3.6;

- $[\![t_1 \circledast t_2]\!]_1$ is defined uniquely; $[\![t_1 \circledast t_2]\!]_{i+1}(\gamma) \triangleq ([\![t_1]\!]_{i+1}(\gamma)_i) \circ [\![t_2]\!]_{i+1}(\gamma)$.

♦

Lemma 4.3.8. *Given typed terms in context $x_1 : A_1, \ldots, x_m : A_m \vdash t : A$ and $\Gamma \vdash t_k : A_k$ for $1 \leq k \leq m$, $\left[\!\left[t[\vec{t}/\vec{x}]\right]\!\right]_i(\gamma) = [\![t]\!]_i([\![t_1]\!]_i(\gamma), \ldots, [\![t_m]\!]_i(\gamma))$.* ♢

Theorem 4.3.9 (Soundness)**.** *If $t \rightsquigarrow u$ then $[\![t]\!] = [\![u]\!]$.* ♢

We now define a logical relation between our denotational semantics and terms, from which both normalisation and adequacy will follow. Doing this inductively proves rather delicate, because induction on size will not support reasoning about our values, as fold refers to a larger type in its premise. This motivates a notion of *unguarded size* under which $A[\mu\alpha.A/\alpha]$ is 'smaller' than $\mu\alpha.A$. But under this metric $\blacktriangleright A$ is smaller than A, so next now poses a problem. But the meaning of $\blacktriangleright A$ at index $i+1$ is determined by A at index i, and so, as in Birkedal et al [19], our relation will also induct on index. This in turn creates problems with box, whose meaning refers to all indexes simultaneously, motivating a notion of *box depth*, allowing us finally to attain well-defined induction.

Definition 4.3.10. The *unguarded size* us of an open type follows the obvious definition for type size, except that $\mathsf{us}(\blacktriangleright A) = 0$.

The *box depth* bd of an open type is

- $\mathsf{bd}(A) = 0$ for $A \in \{\alpha, \mathbf{0}, \mathbf{1}, \mathbf{N}\}$;
- $\mathsf{bd}(A \times B) = \mathsf{min}(\mathsf{bd}(A), \mathsf{bd}(B))$, and similarly for $\mathsf{bd}(A \to B)$;
- $\mathsf{bd}(\mu\alpha.A) = \mathsf{bd}(A)$, and similarly for $\mathsf{bd}(\blacktriangleright A)$;
- $\mathsf{bd}(\blacksquare A) = \mathsf{bd}(A) + 1$.

♦

Lemma 4.3.11. *(i) α guarded in A implies $\mathsf{us}(A[B/\alpha]) \leq \mathsf{us}(A)$.*

(ii) $\mathsf{bd}(B) \leq \mathsf{bd}(A)$ implies $\mathsf{bd}(A[B/\alpha]) \leq \mathsf{bd}(A)$

♢

Definition 4.3.12. The family of relations R_i^A, indexed by closed types A and positive integers i, relates elements of the semantics $a \in [\![A]\!]_i$ and closed typed terms $t : A$ and is defined as

- $* R_i^{\mathbf{1}} t$ iff $t \rightsquigarrow \langle\rangle$;
- $n R_i^{\mathbf{N}} t$ iff $t \rightsquigarrow \mathsf{succ}^n\, \mathsf{zero}$;
- $(a_1, a_2) R_i^{A_1 \times A_2} t$ iff $t \rightsquigarrow \langle t_1, t_2 \rangle$ and $a_d R_i^{A_d} t_d$ for $d \in \{1, 2\}$;

- $fR_i^{A\to B}t$ iff $t \rightsquigarrow \lambda x.s$ and for all $j \leq i$, $aR_j^A u$ implies $f_j(a)R_j^B s[u/x]$;
- $aR_i^{\mu\alpha.A}t$ iff $t \rightsquigarrow \mathsf{fold}\,u$ and $h_i(a)R_i^{A[\mu\alpha.A/\alpha]}u$, where h is the "unfold" isomorphism for the recursive type (ref. Lemma 4.3.5);
- $aR_i^{\blacktriangleright A}t$ iff $t \rightsquigarrow \mathsf{next}\,u$ and, where $i > 1$, $aR_{i-1}^A u$.
- $aR_i^{\blacksquare A}t$ iff $t \rightsquigarrow \mathsf{box}\,u$ and for all j, $a_j R_j^A u$;

This is well-defined by induction on the lexicographic ordering on box depth, then index, then unguarded size. First the ■ case strictly decreases box depth, and no other case increases it (ref. Lemma 4.3.11.(ii) for μ-types). Second the ▶ case strictly decreases index, and no other case increases it (disregarding ■). Finally all other cases strictly decrease unguarded size, as seen via Lemma 4.3.11.(i) for μ-types. ♦

Lemma 4.3.13 (Fundamental Lemma)**.** *Take $\Gamma = (x_1 : A_1, \dots, x_m : A_m)$, $\Gamma \vdash t : A$, and $\vdash t_k : A_k$ for $1 \leq k \leq m$. Then for all i, if $a_k R_i^{A_k} t_k$ for all k, then*

$$[\![\Gamma \vdash t : A]\!]_i\,(\vec{a})\,R_i^A\,t[\vec{t}/\vec{x}].$$

◊

Theorem 4.3.14 (Adequacy and Normalisation)**.**

(i) For all closed terms $\vdash t : A$ it holds that $[\![t]\!]_i\,R_i^A t$;

(ii) $[\![\vdash t : \mathbf{N}]\!]_i = n$ implies $t \rightsquigarrow \mathsf{succ}^n\,\mathsf{zero}$;

(iii) All closed typed terms evaluate to a value.

◊

Proof. (i) specialises Lemma 4.3.13 to closed types. (ii), (iii) hold by (i) and inspection of Definition 4.3.12. QED

Definition 4.3.15. Typed *contexts* with typed holes are defined as obvious. Two terms $\Gamma \vdash t : A, \Gamma \vdash u : A$ are *contextually equivalent*, written $t \simeq_{\mathsf{ctx}} u$, if for all *closing* contexts C of type $\mathbf{N}$, the terms $C[t]$ and $C[u]$ reduce to the same value. ♦

Corollary 4.3.16. *$[\![t]\!] = [\![u]\!]$ implies $t \simeq_{\mathsf{ctx}} u$.* ◊

Proof. $[\![C[t]]\!] = [\![C[u]]\!]$ by compositionality of the denotational semantics . Then by Theorem 2 they reduce to the same value. QED

4.4 Logic for Guarded Lambda Calculus

This section presents our program logic $Lg\lambda$ for the guarded λ-calculus. The logic is an extension of the internal language of $\mathcal{S}$ [22, 30]. Thus it extends multisorted intuitionistic higher-order logic with two propositional modalities $\triangleright$ and $\Box$, pronounced "later" and "always" respectively. The term language of $Lg\lambda$ includes the terms of $\mathrm{g}\lambda$, and the types of $Lg\lambda$ include types definable in $\mathrm{g}\lambda$. We write Ω for the type of propositions, and also for the subobject classifier of $\mathcal{S}$.

The rules for *definitional equality* extend the usual $\beta\eta$-laws for functions and products with new equations for the new $\mathrm{g}\lambda$ constructs, listed in Figure 4.3.

$$\frac{\Gamma \vdash t : A[\mu\alpha.A/\alpha]}{\Gamma \vdash \mathsf{unfold}(\mathsf{fold}\, t) = t} \qquad \frac{\Gamma \vdash t : \mu\alpha.A}{\Gamma \vdash \mathsf{fold}(\mathsf{unfold}\, t) = t} \qquad \frac{\Gamma \vdash t_1 : A \to B \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash \mathsf{next}\, t_1 \circledast \mathsf{next}\, t_2 = \mathsf{next}(t_1\, t_2)}$$

$$\frac{\Gamma_\blacksquare \vdash t : A \qquad \Gamma \vdash \vec{t} : \Gamma_\blacksquare}{\Gamma \vdash \mathsf{prev}[\vec{x} \leftarrow \vec{t}].(\mathsf{next}\, t) = t\left[\vec{t}/\vec{x}\right]} \qquad \frac{\Gamma_\blacksquare \vdash t : \blacktriangleright A \qquad \Gamma \vdash \vec{t} : \Gamma_\blacksquare}{\Gamma \vdash \mathsf{next}\left(\mathsf{prev}[\vec{x} \leftarrow \vec{t}].t\right) = t\left[\vec{t}/\vec{x}\right]}$$

$$\frac{\Gamma_\blacksquare \vdash t : A \qquad \Gamma \vdash \vec{t} : \Gamma_\blacksquare}{\Gamma \vdash \mathsf{unbox}(\mathsf{box}[\vec{x} \leftarrow \vec{t}].t) = t\left[\vec{t}/\vec{x}\right]} \qquad \frac{\Gamma_\blacksquare \vdash t : \blacksquare A \qquad \Gamma \vdash \vec{t} : \Gamma_\blacksquare}{\Gamma \vdash \mathsf{box}[\vec{x} \leftarrow \vec{t}].\, \mathsf{unbox}\, t = t\left[\vec{t}/\vec{x}\right]}$$

Figure 4.3: Additional equations. The context $\Gamma_\blacksquare$ is assumed constant.

Definition 4.4.1. A type X is *total and inhabited* if the formula $\mathrm{Total}(X) \equiv \forall x : \blacktriangleright X, \exists x' : X, \mathbf{next}(x') =_{\blacktriangleright X} x$ is valid. ♦

All of the $\mathrm{g}\lambda$-types defined in Sec. 4.2 are total and inhabited (see Appendix 4.E for a proof using the semantics of the logic), but that is not the case when we include sum types as the empty type is not inhabited.

Corresponding to the modalities $\blacktriangleright$ and $\blacksquare$ on types, we have modalities $\triangleright$ and $\Box$ on formulas. The modality $\triangleright$ is used to express that a formula holds only "later", that is, after a time step. It is given by a function symbol $\triangleright : \Omega \to \Omega$. The $\Box$ modality is used to express that a formula holds for all time steps. Unlike the $\triangleright$ modality, $\Box$ on formulas does not arise from a function on Ω [26]. As with box, it is only well-behaved in constant contexts, so we will only allow $\Box$ in such contexts. The rules for $\triangleright$ and $\Box$ are listed in Figure 4.4.

The $\triangleright$ modality can in fact be defined in terms of $\mathsf{lift} : \blacktriangleright\Omega \to \Omega$ (called *succ* by Birkedal et al [22]) as $\triangleright = \mathsf{lift} \circ \mathsf{next}$. The lift function will be useful since it allows us to define predicates over guarded types, such as predicates on $\mathrm{Str}^{\mathrm{g}}$.

$$\frac{}{\Gamma \mid \Xi, (\triangleright\varphi \Rightarrow \varphi) \vdash \varphi}\ \textsc{Löb} \qquad \frac{}{\Gamma, x : X \mid \exists y : Y, \triangleright\varphi(x,y) \vdash \triangleright(\exists y : Y, \varphi(x,y))}\ \exists\triangleright$$

$$\frac{}{\Gamma, x : X \mid \triangleright(\forall y : Y, \varphi(x,y)) \vdash \forall y : Y, \triangleright\varphi(x,y)}\ \forall\triangleright \qquad \frac{}{\Gamma \mid \Xi, \varphi \vdash \triangleright\varphi}$$

$$\frac{\star \in \{\wedge, \vee, \Rightarrow\}}{\Gamma \mid \triangleright(\varphi \star \psi) \dashv\vdash \triangleright\varphi \star \triangleright\psi} \qquad \frac{\Gamma \mid \neg\neg\varphi \vdash \psi}{\Gamma \mid \varphi \vdash \square\psi} \qquad \frac{\Gamma \mid \varphi \vdash \square\psi}{\Gamma \mid \neg\neg\varphi \vdash \psi} \qquad \frac{\Gamma \mid \varphi \vdash \psi}{\Gamma \mid \square\varphi \vdash \square\psi}$$

$$\frac{}{\Gamma \mid \square\varphi \vdash \varphi} \qquad \frac{}{\Gamma \mid \square\varphi \vdash \square\square\varphi} \qquad \frac{}{\forall x, y : X.\, \triangleright(x =_X y) \Leftrightarrow \mathsf{next}\, x =_{\blacktriangleright X} \mathsf{next}\, y}\ \mathrm{EQ}^{\triangleright}_{\mathsf{next}}$$

Figure 4.4: Rules for $\triangleright$ and $\square$. The judgement $\Gamma \mid \Xi \vdash \varphi$ expresses that in typing context Γ, hypotheses in Ξ prove φ. The converse entailment in $\forall\triangleright$ and $\exists\triangleright$ rules holds if Y is *total and inhabited*. In all rules involving the $\square$ the context Γ is assumed constant.

The semantics of the logic is given in $\mathcal{S}$; terms are interpreted as morphisms of $\mathcal{S}$ and formulas are interpreted via the subobject classifier. We do not present the semantics here; except for the new terms of $\mathrm{g}\lambda$, whose semantics are defined in Sec. 4.3, the semantics are as in [22, 26].

Later we will come to the problem of proving $x =_{\blacksquare A} y$ from $\mathsf{unbox}\, x =_A \mathsf{unbox}\, y$, where x, y have type $\blacksquare A$. This in general does not hold, but using the semantics of $L\mathrm{g}\lambda$ we can prove the proposition below.

Proposition 4.4.2. *The formula* $\square(\mathsf{unbox}\, x =_A \mathsf{unbox}\, y) \Rightarrow x =_{\blacksquare A} y$ *is valid.* ◊

There exists a fixed-point combinator of type $(\blacktriangleright A \to A) \to A$ for all types A in the logic (not only those of in $\mathrm{g}\lambda$) [22, Theorem 2.4]; we also write fix for it.

Proposition 4.4.3. *For any term* $f : \blacktriangleright A \to A$ *we have* $\mathsf{fix}\, f =_A f\,(\mathsf{next}(\mathsf{fix}\, f))$ *and, if* u *is any other term such that* $f(\mathsf{next}\, u) =_A u$*, then* $u =_A \mathsf{fix}\, f$*.* ◊

In particular this can be used for recursive definitions of predicates. For instance if $P : \mathbb{N} \to \Omega$ is a predicate on natural numbers we can define a predicate $P_{\mathsf{Str}^{\mathrm{g}}}$ on $\mathsf{Str}^{\mathrm{g}}$ expressing that P holds for all elements of the stream:

$$P_{\mathsf{Str}^{\mathrm{g}}} \triangleq \mathsf{fix}\, \lambda r. \lambda xs. P(\mathsf{hd}^{\mathrm{g}}\, xs) \wedge \mathsf{lift}\,(r \circledast (\mathsf{tl}^{\mathrm{g}}\, xs)) : \mathsf{Str}^{\mathrm{g}} \to \Omega.$$

The logic may be used to prove contextual equivalence of programs:

Theorem 4.4.4. *Let* t_1 *and* t_2 *be two* $\mathrm{g}\lambda$ *terms of type* A *in context* Γ*. If the sequent* $\Gamma \mid \emptyset \vdash t_1 =_A t_2$ *is provable then* t_1 *and* t_2 *are contextually equivalent.* ◊

Proof. Recall that equality in the internal logic of a topos is just equality of morphisms. Hence t_1 and t_2 denote same morphism from Γ to A. Adequacy (Cor. 4.3.16) then implies that t_1 and t_2 are contextually equivalent. $\mathfrak{QED}$

Example 4.4.5. We list some properties provable using the logic. Except for the first property all proof details are in Appendix 4.B.

(i) For any $f : A \to B$ and $g : B \to C$ we have

$$(\mathsf{map}^{\mathsf{g}}\, f) \circ (\mathsf{map}^{\mathsf{g}}\, g) =_{\mathsf{Str}^{\mathsf{g}} \to \mathsf{Str}^{\mathsf{g}}} \mathsf{map}^{\mathsf{g}}(f \circ g).$$

Unfolding the definition of $\mathsf{map}^{\mathsf{g}}$ from Ex. 4.2.10(vi) and using β-rules and Proposition 4.4.3 we have $\mathsf{map}^{\mathsf{g}}\, f\, xs = f\,(\mathsf{hd}^{\mathsf{g}}\, xs) :: (\mathsf{next}(\mathsf{map}^{\mathsf{g}}\, f) \circledast (\mathsf{tl}^{\mathsf{g}}\, xs))$. Equality of functions is extensional so we have to prove

$$\Phi \triangleq \forall xs : \mathsf{Str}^{\mathsf{g}}, \mathsf{map}^{\mathsf{g}}\, f\,(\mathsf{map}^{\mathsf{g}}\, g\, xs) =_{\mathsf{Str}^{\mathsf{g}}} \mathsf{map}^{\mathsf{g}}(f \circ g)\, xs.$$

The proof is by Löb induction, so we assume $\triangleright\Phi$ and take $xs : \mathsf{Str}^{\mathsf{g}}$. Using the above property of $\mathsf{map}^{\mathsf{g}}$ we unfold $\mathsf{map}^{\mathsf{g}}\, f\,(\mathsf{map}^{\mathsf{g}}\, g\, xs)$ to

$$f\,(g\,(\mathsf{hd}^{\mathsf{g}}\, xs)) :: (\mathsf{next}(\mathsf{map}^{\mathsf{g}}\, f) \circledast ((\mathsf{next}(\mathsf{map}^{\mathsf{g}}\, g)) \circledast \mathsf{tl}^{\mathsf{g}}\, xs))$$

and we unfold $\mathsf{map}^{\mathsf{g}}(f \circ g)\, xs$ to $f\,(g\,(\mathsf{hd}^{\mathsf{g}}\, xs)) :: (\mathsf{next}(\mathsf{map}^{\mathsf{g}}(f \circ g)) \circledast \mathsf{tl}^{\mathsf{g}}\, xs)$. Since $\mathsf{Str}^{\mathsf{g}}$ is a total type there is a $xs' : \mathsf{Str}^{\mathsf{g}}$ such that $\mathsf{next}\, xs' = \mathsf{tl}^{\mathsf{g}}\, xs$. Using this and the rule for $\circledast$ we have

$$\mathsf{next}(\mathsf{map}^{\mathsf{g}}\, f) \circledast ((\mathsf{next}(\mathsf{map}^{\mathsf{g}}\, g)) \circledast \mathsf{tl}^{\mathsf{g}}\, xs) =_{\blacktriangleright \mathsf{Str}^{\mathsf{g}}} \mathsf{next}(\mathsf{map}^{\mathsf{g}}\, f\,(\mathsf{map}^{\mathsf{g}}\, g\, xs'))$$

and $\mathsf{next}(\mathsf{map}^{\mathsf{g}}(f \circ g)) \circledast \mathsf{tl}^{\mathsf{g}}\, xs =_{\blacktriangleright \mathsf{Str}^{\mathsf{g}}} \mathsf{next}(\mathsf{map}^{\mathsf{g}}(f \circ g)\, xs')$. From the induction hypothesis $\triangleright\Phi$ we have $\triangleright(\mathsf{map}^{\mathsf{g}}(f \circ g)\, xs' =_{\mathsf{Str}^{\mathsf{g}}} \mathsf{map}^{\mathsf{g}}\, f\,(\mathsf{map}^{\mathsf{g}}\, g\, xs'))$ and so rule $\textsc{eq}^{\triangleright}_{\mathsf{next}}$ concludes the proof.

(ii) We can also reason about acausal functions. For any $n : \mathbf{N}, f : \mathbf{N} \to \mathbf{N}$,

$$\mathsf{every2nd}(\mathsf{box}\,\iota.\, \mathsf{iterate}\,(\mathsf{next}\, f)\, n) =_{\mathsf{Str}^{\mathsf{g}}} \mathsf{iterate}\,(\mathsf{next}\, f^2)\, n,$$

where f^2 is $\lambda m. f\,(f\, m)$. The proof again uses Löb induction.

(iii) Since our logic is higher-order we can state and prove very general properties, for instance the following general property of map

$$\begin{aligned}&\forall P, Q : (\mathbb{N} \to \Omega), \forall f : \mathbb{N} \to \mathbb{N}, (\forall x : \mathbb{N}, P(x) \Rightarrow Q(f(x)))\\ &\Rightarrow \forall xs : \mathsf{Str}^{\mathsf{g}}, P_{\mathsf{Str}^{\mathsf{g}}}(xs) \Rightarrow Q_{\mathsf{Str}^{\mathsf{g}}}(\mathsf{map}^{\mathsf{g}}\, f\, xs).\end{aligned}$$

The proof illustrates the use of the property $\mathsf{lift} \circ \mathsf{next} = \triangleright$.

(iv) Given a closed term (we can generalise to terms in constant contexts) f of type $A \to B$ we have $\mathsf{box}\, f$ of type $\blacksquare(A \to B)$. Define $\mathcal{L}(f) = \mathsf{lim}(\mathsf{box}\, f)$ of type $\blacksquare A \to \blacksquare B$. For any closed term $f : A \to B$ and $x : \blacksquare A$ we can then prove $\mathsf{unbox}(\mathcal{L}(f)\, x) =_B f\, (\mathsf{unbox}\, x)$. Then using Proposition 4.4.2 we can, for instance, prove $\mathcal{L}(f \circ g) = \mathcal{L}(f) \circ \mathcal{L}(g)$.

For functions of arity k we define $\mathcal{L}_k$ using $\mathcal{L}$, and analogous properties hold, e.g. we have $\mathsf{unbox}(\mathcal{L}_2(f)\, x\, y) = f\, (\mathsf{unbox}\, x)(\mathsf{unbox}\, y)$, which allows us to transfer equalities proved for functions on guarded types to functions on $\blacksquare$'d types; see Sec. 4.5 for an example.

♦

4.5 Behavioural Differential Equations in gλ

In this section we demonstrate the expressivity of our approach by showing how to construct solutions to behavioural differential equations [83] in gλ, and how to reason about such functions in Lgλ, rather than with bisimulation as is more traditional. These ideas are best explained via a simple example.

Supposing addition $+ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is given, then pointwise addition of streams, plus, can be defined by the following behavioural differential equation

$$\mathsf{hd}(\mathsf{plus}\, \sigma_1\, \sigma_2) = \mathsf{hd}\, \sigma_1 + \mathsf{hd}\, \sigma_2 \qquad\qquad \mathsf{tl}(\mathsf{plus}\, \sigma_1\, \sigma_2) = \mathsf{plus}(\mathsf{tl}\, \sigma_1)\, (\mathsf{tl}\, \sigma_2).$$

To define the solution to this behavioural differential equation in gλ, we first translate it to a function on guarded streams $\mathsf{plus}^{\mathsf{g}} : \mathsf{Str}^{\mathsf{g}} \to \mathsf{Str}^{\mathsf{g}} \to \mathsf{Str}^{\mathsf{g}}$, as

$$\mathsf{plus}^{\mathsf{g}} \triangleq \mathsf{fix}\, \lambda f. \lambda s_1. \lambda s_2. (\mathsf{hd}^{\mathsf{g}}\, s_1 + \mathsf{hd}^{\mathsf{g}}\, s_2) :: (f \circledast (\mathsf{tl}^{\mathsf{g}}\, s_1) \circledast (\mathsf{tl}^{\mathsf{g}}\, s_2))$$

then define $\mathsf{plus} : \mathsf{Str} \to \mathsf{Str} \to \mathsf{Str}$ by $\mathsf{plus} = \mathcal{L}_2(\mathsf{plus}^{\mathsf{g}})$. By Proposition 4.4.3 we have

$$\mathsf{plus}^{\mathsf{g}} = \lambda s_1. \lambda s_2. (\mathsf{hd}^{\mathsf{g}}\, s_1 + \mathsf{hd}^{\mathsf{g}}\, s_2) :: ((\mathsf{next}\, \mathsf{plus}^{\mathsf{g}}) \circledast (\mathsf{tl}^{\mathsf{g}}\, s_1) \circledast (\mathsf{tl}^{\mathsf{g}}\, s_2)). \tag{4.1}$$

This definition of plus satisfies the specification given by the behavioural differential equation above. Let $\sigma_1, \sigma_2 : \mathsf{Str}$ and recall that $\mathsf{hd} = \mathsf{hd}^{\mathsf{g}} \circ \lambda s.\, \mathsf{unbox}\, s$. Then use Ex. 4.4.5.(iv) and equality (4.1) to get $\mathsf{hd}(\mathsf{plus}\, \sigma_1 \sigma_2) = \mathsf{hd}\, \sigma_1 + \mathsf{hd}\, \sigma_2$.

For tl we proceed similarly, also using that $\mathsf{tl}^{\mathsf{g}}(\mathsf{unbox}\, \sigma) = \mathsf{next}(\mathsf{unbox}(\mathsf{tl}\, \sigma))$ which can be proved using the β-rule for box and the η-rule for next.

Since $\mathsf{plus}^{\mathsf{g}}$ is defined via guarded recursion we can reason about it with Löb induction, for example to prove that it is commutative. Ex. 4.4.5.(iv) and Proposition 4.4.2 then immediately give that plus on *coinductive* streams Str is commutative.

Once we have defined $\mathsf{plus}^{\mathrm{g}}$ we can use it when defining other functions on streams, for instance stream multiplication $\otimes$ which is specified by equations

$$\mathsf{hd}(\sigma_1 \otimes \sigma_2) = (\mathsf{hd}\,\sigma_1) \cdot (\mathsf{hd}\,\sigma_2) \quad \mathsf{tl}(\sigma_1 \otimes \sigma_2) = (\rho(\mathsf{hd}\,\sigma_1) \otimes (\mathsf{tl}\,\sigma_2)) \oplus ((\mathsf{tl}\,\sigma_1) \otimes \sigma_2)$$

where $\rho(n)$ is a stream with head n and tail a stream of zeros, and $\cdot$ is multiplication of natural numbers, and using $\oplus$ as infix notation for plus. We can define $\otimes^{\mathrm{g}} : \mathsf{Str}^{\mathrm{g}} \to \mathsf{Str}^{\mathrm{g}} \to \mathsf{Str}^{\mathrm{g}}$ by $\otimes^{\mathrm{g}} \triangleq$

$$\begin{aligned}&\mathsf{fix}\,\lambda f.\lambda s_1.\lambda s_2.((\mathsf{hd}^{\mathrm{g}}\,s_1) \cdot (\mathsf{hd}^{\mathrm{g}}\,s_2))::\\ &\qquad\qquad (\mathsf{next}\,\mathsf{plus}^{\mathrm{g}} \circledast (f \circledast \mathsf{next}\,\iota^{\mathrm{g}}(\mathsf{hd}^{\mathrm{g}}\,s_1) \circledast \mathsf{tl}^{\mathrm{g}}\,s_2) \circledast (f \circledast \mathsf{tl}^{\mathrm{g}}\,s_1 \circledast \mathsf{next}\,s_2))\end{aligned}$$

then define $\otimes = \mathcal{L}_2\,(\otimes^{\mathrm{g}})$. It can be shown that the function $\otimes$ so defined satisfies the two defining equations above. Note that the guarded $\mathsf{plus}^{\mathrm{g}}$ is used to define $\otimes^{\mathrm{g}}$, so our approach is *modular* in the sense of [70].

The example above generalises, as we can show that any solution to a behavioural differential equation in **Set** can be obtained via guarded recursion together with $\mathcal{L}_k$. The formal statement is somewhat technical and can be found in Appendix 4.D.

4.6 Discussion

Following Nakano [72], the ► modality has been used as type-former for a number of λ-calculi for guarded recursion. Nakano's calculus and some successors [2, 57, 88] permit only *causal* functions. The closest such work to ours is that of Abel and Vezzosi [2], but due to a lack of destructor for ► their (strong) normalisation result relies on a somewhat artificial operational semantics where the number of nexts that can be reduced under is bounded by some fixed natural number.

Atkey and McBride's extension of such calculi to acausal functions [11] forms the basis of this paper. We build on their work by (aside from various minor changes such as eliminating the need to work modulo first-class type isomorphisms) introducing normalising operational semantics, an adequacy proof with respect to the topos of trees, and a program logic.

An alterative approach to type-based productivity guarantees are *sized types*, introduced by Hughes et al [50] and now extensively developed, for example integrated into a variant of System F_ω [1]. Our approach offers some advantages, such as adequate denotational semantics, and a notion of program proof without appeal to dependent types, but extensions with realistic language features (e.g. following Møgelberg [71]) clearly need to be investigated.

Acknowledgements

We gratefully acknowledge discussions with Andreas Abel, Tadeusz Litak, Stefan Milius, Rasmus Møgelberg, Filip Sieczkowski, and Andrea Vezzosi, and the comments of the reviewers. This research was supported in part by the ModuRes Sapere Aude Advanced Grant from The Danish Council for Independent Research for the Natural Sciences (FNU). Aleš Bizjak is supported in part by a Microsoft Research PhD grant.

4.A Proofs about the Denotational Semantics and Normalisation

This section contains proofs of statements in Section 4.3.

of Lemma 4.3.6. By induction on type formation, with $\blacktriangleright A$ case omitted, $\blacksquare A$ a base case, and $\mu\alpha.A$ considered only where α is not free in A. $\mathfrak{QED}$

of Lemma 4.3.8. By induction on the typing of t. We present the cases particular to our calculus.

$\mathsf{next}\,t$: case $i = 1$ is trivial. $\left\llbracket \mathsf{next}\,t[\vec{t}/\vec{x}] \right\rrbracket_{i+1}(\gamma) = r_i^{\llbracket A \rrbracket} \circ \left\llbracket t[\vec{t}/\vec{x}] \right\rrbracket_{i+1}(\gamma) = r_i^{\llbracket A \rrbracket} \circ \llbracket t \rrbracket_{i+1}(\llbracket t_1 \rrbracket_{i+1}(\gamma),\ldots)$ by induction, which is $\llbracket \mathsf{next}\,t \rrbracket_{i+1}(\llbracket t_1 \rrbracket_{i+1}(\gamma),\ldots)$.

$\left\llbracket (\mathsf{prev}[\vec{y} \leftarrow \vec{u}].t)[\vec{t}/\vec{x}] \right\rrbracket_i(\gamma) = \left\llbracket \mathsf{prev}[\vec{y} \leftarrow \vec{u}[\vec{t}/\vec{x}]].t \right\rrbracket_i(\gamma)$, which by definition is

$$\llbracket t \rrbracket_{i+1}(\left\llbracket u_1[\vec{t}/\vec{x}] \right\rrbracket_i(\gamma),\ldots) = \llbracket t \rrbracket_{i+1}(\llbracket u_1 \rrbracket_i(\llbracket t_1 \rrbracket_i(\gamma),\ldots),\ldots)$$

by induction, which is $\llbracket \mathsf{prev}[\vec{y} \leftarrow \vec{u}].t \rrbracket_i(\llbracket t_1 \rrbracket_i(\gamma),\ldots)$.

By definition $\left\llbracket \mathsf{box}[\vec{y} \leftarrow \vec{u}[\vec{t}/\vec{x}]].t \right\rrbracket_i(\gamma)_j = \llbracket t \rrbracket_j(\left\llbracket u_1[\vec{t}/\vec{x}] \right\rrbracket_i(\gamma),\ldots)$, which by induction equals $\llbracket t \rrbracket_j(\llbracket u_1 \rrbracket_i(\llbracket t_1 \rrbracket_i(\gamma),\ldots),\ldots) = \llbracket \mathsf{box}[\vec{y} \leftarrow \vec{u}].t \rrbracket_i(\llbracket t_1 \rrbracket_i(\gamma),\ldots)_j$.

$\left\llbracket \mathsf{unbox}\,t[\vec{t}/\vec{x}] \right\rrbracket_i(\gamma) = \left\llbracket t[\vec{t}/\vec{x}] \right\rrbracket_i(\gamma)_i = \llbracket t \rrbracket_i(\llbracket t_1 \rrbracket_i(\gamma),\ldots)_i$ by induction, which is $\llbracket \mathsf{unbox}\,t \rrbracket_i(\llbracket t_1 \rrbracket_i(\gamma),\ldots)$.

$u_1 \circledast u_2$: case $i = 1$ is trivial. $\left\llbracket (u_1 \circledast u_2)[\vec{t}/\vec{x}] \right\rrbracket_{i+1}(\gamma) = (\left\llbracket u_1[\vec{t}/\vec{x}] \right\rrbracket_{i+1}(\gamma)_i) \circ \left\llbracket u_2[\vec{t}/\vec{x}] \right\rrbracket_{i+1}(\gamma) = (\llbracket u_1 \rrbracket_{i+1}(\llbracket t_1 \rrbracket_{i+1}(\gamma),\ldots)_i) \circ \llbracket u_2 \rrbracket_{i+1}(\llbracket t_1 \rrbracket_{i+1}(\gamma),\ldots)$, which is $\llbracket u_1 \circledast u_2 \rrbracket_{i+1}(\llbracket t_1 \rrbracket_{i+1}(\gamma),\ldots)$. $\mathfrak{QED}$

of Soundness Theorem 4.3.9. We verify the reduction rules of Definition 4.2.2; extending this to any evaluation context, and to $\rightsquigarrow$, is easy. The product reduction case is standard, and function case requires Lemma 4.3.8. $\mathsf{unfold}\,\mathsf{fold}$ is the application of mutually inverse arrows.

$\left\llbracket \mathsf{prev}[\vec{x} \leftarrow \vec{t}].t \right\rrbracket_i = \llbracket t \rrbracket_{i+1}(\llbracket t_1 \rrbracket_i,\ldots)$. Each t_k is closed, so is denoted by an arrow from 1 to the constant $\mathcal{S}$-object $\llbracket A_k \rrbracket$, so by naturality $\llbracket t_k \rrbracket_i = \llbracket t_k \rrbracket_{i+1}$. But $\llbracket t \rrbracket_{i+1}(\llbracket t_1 \rrbracket_{i+1},\ldots) = \left\llbracket t[\vec{t}/\vec{x}] \right\rrbracket_{i+1}$ by Lemma 4.3.8, which is $\left\llbracket \mathsf{prev}\,t[\vec{t}/\vec{x}] \right\rrbracket_i$.

$\llbracket \mathsf{prev}\,\mathsf{next}\,t \rrbracket_i = \llbracket \mathsf{next}\,t \rrbracket_{i+1} = \llbracket t \rrbracket_i$.

$\left\llbracket \mathsf{unbox}(\mathsf{box}[\vec{x} \leftarrow \vec{t}].t) \right\rrbracket_i = (\left\llbracket \mathsf{box}[\vec{x} \leftarrow \vec{t}].t \right\rrbracket_i)_i = \llbracket t \rrbracket_i(\llbracket t_1 \rrbracket_i,\ldots) = \left\llbracket t[\vec{t}/\vec{x}] \right\rrbracket_i$.

With $\circledast$-reduction, index 1 is trivial. $\llbracket \mathsf{next}\,t_1 \circledast \mathsf{next}\,t_2 \rrbracket_{i+1} = (\llbracket \mathsf{next}\,t_1 \rrbracket_{i+1})_i \circ \llbracket \mathsf{next}\,t_2 \rrbracket_{i+1} = (r_i^{\llbracket A \to B \rrbracket} \circ \llbracket t_1 \rrbracket_{i+1})_i \circ r_i^{\llbracket A \rrbracket} \circ \llbracket t_2 \rrbracket_{i+1} = (\llbracket t_1 \rrbracket_i \circ r_i^1)_i \circ \llbracket t_2 \rrbracket_i \circ r_i^1$ by naturality, which is $(\llbracket t_1 \rrbracket_i)_i \circ \llbracket t_2 \rrbracket_i = \llbracket t_1 t_2 \rrbracket_i = \llbracket t_1 t_2 \rrbracket_i \circ r_i^1 = r_i^{\llbracket B \rrbracket} \circ \llbracket t_1 t_2 \rrbracket_{i+1} = \llbracket \mathsf{next}(t_1 t_2) \rrbracket_{i+1}$. $\mathfrak{QED}$

of Lemma 4.3.11. By induction on the construction of the type A.

(i) follows with only interesting case the variable case – A cannot be α because of the requirement that α be guarded in A.

(ii) follows with interesting cases: variable case enforces $bd(B) = 0$; binary type-formers $\times, \to$ have for example $\mathsf{bd}(A_d) \geq \mathsf{bd}(A_1 \times A_2)$, so $\mathsf{bd}(A_d) \geq bd(B)$ and the induction follows; $\blacksquare A$ by construction has no free variables. 𝔔𝔈𝔇

Lemma 4.A.1. *If $t \rightsquigarrow u$ and $aR_i^A u$ then $aR_i^A t$.* ◊

Proof. All cases follow similarly; consider $A_1 \times A_2$. $(a_1, a_2)R_i^{A_1 \times A_2} u$ implies $u \rightsquigarrow \langle t_1, t_2 \rangle$, where this value obeys some property. But then $t \rightsquigarrow \langle t_1, t_2 \rangle$ similarly. 𝔔𝔈𝔇

Lemma 4.A.2. *$aR_{i+1}^A t$ implies $r_i^{[\![A]\!]}(a) R_i^A t$.* ◊

Proof. Cases $\mathbf{1}, \mathbf{N}$ are trivial. Case $\times$ follows by induction because restrictions are defined pointwise. Case μ follows by induction and the naturality of the isomorphism h. Case $\blacksquare A$ follows because $r_i^{[\![\blacksquare A]\!]}(a) = a$.

For $A \to B$ take $j \leq i$ and $a' R_j^A u$. By the downwards closure in the definition of $R_{i+1}^{A \to B}$ we have $f_j(a') R_j^B s[u/x]$. But $f_j = (r_i^{[\![A \to B]\!]}(f))_j$.

With $\blacktriangleright A$, case $i = 1$ is trivial, so take $i = j + 1$. $aR_{j+2}^{\blacktriangleright A} t$ means $t \rightsquigarrow \mathsf{next}\, u$ and $aR_{j+1}^A u$, so by induction $r_j^{[\![A]\!]}(a) R_j^A u$, so $r_{j+1}^{[\![\blacktriangleright A]\!]}(a) R_j^A u$ as required. 𝔔𝔈𝔇

Lemma 4.A.3. *If $aR_i^A t$ and A is constant, then $aR_j^A t$ for all j.* ◊

Proof. Easy induction on types, ignoring $\blacktriangleright A$ and treating $\blacksquare A$ as a base case. 𝔔𝔈𝔇

We finally turn to the proof of the Fundamental Lemma.

of Lemma 4.3.13. By induction on the typing $\Gamma \vdash t : A$. $\langle\rangle, \mathsf{zero}$ cases are trivial, and $\langle u_1, u_2 \rangle, \mathsf{fold}\, t$ cases follow by easy induction.

succ: If $t[\vec{t}/\vec{x}]$ reduces to $\mathsf{succ}^l\, \mathsf{zero}$ for some l then $\mathsf{succ}\, t[\vec{t}/\vec{x}]$ reduces to $\mathsf{succ}^{l+1}\, \mathsf{zero}$, as we may reduce under the succ.

$\pi_d t$: If $[\![t]\!]_i(\vec{a}) R_i^{A_1 \times A_2} t[\vec{t}/\vec{x}]$ then $t[\vec{t}/\vec{x}] \rightsquigarrow \langle u_1, u_2 \rangle$ and u_d is related to the d'th projection of $[\![t]\!]_i(\vec{a})$. But then $\pi_d t[\vec{t}/\vec{x}] \rightsquigarrow \pi_d \langle u_1, u_2 \rangle \mapsto u_d$. Lemma 4.A.1 completes the case.

$\lambda x.t$: Taking $j \leq i$ and $aR_j^A u$, we must show that

$$[\![\lambda x.t]\!]_i(\vec{a})_j(a) R_j^B t[\vec{t}/\vec{x}][u/x].$$

The left hand side is $[\![t]\!]_j(\vec{a}\restriction_j, a)$. For each k, $a_k \restriction_j R_j^{A_k} t_k$ by Lemma 4.A.2, and induction completes the case.

$u_1 u_2$: By induction $u_1[\vec{t}/\vec{x}] \rightsquigarrow \lambda x.s$ and $[\![u_1]\!]_k(\vec{a})_k([\![u_2]\!]_k(\vec{a})) R_i^B s[u_2[\vec{t}/\vec{x}]/x]$. Now $(u_1 u_2) \rightsquigarrow (\lambda x.s)(u_2[\vec{t}/\vec{x}]) \mapsto s[u_2[\vec{t}/\vec{x}]/x]$, and Lemma 4.A.1 completes.

$\mathsf{unfold}\, t$: we reduce under unfold, then reduce $\mathsf{unfold}\, \mathsf{fold}$, and then use Lemma 4.A.1.

$\mathsf{next}\,t$: Trivial for index 1. For $i = j+1$, if for each k, $a_k R^{A_k}_{j+1} t_k$ then by Lemma 4.A.2 $r_j^{\llbracket A_k \rrbracket}(a_k) R^{A_k}_j t_k$. Then by induction $\llbracket t \rrbracket_j \circ r_j^{\llbracket A_1 \rrbracket \times \cdots \llbracket A_m \rrbracket}(\vec{a}) R^A_j t[\vec{t}/\vec{x}]$, whose left side is by naturality $r_j^{\llbracket A \rrbracket} \circ \llbracket t \rrbracket_{j+1}(\vec{a}) = \llbracket \mathsf{next}\,t \rrbracket_{j+1}(\vec{a})$.

$\mathsf{prev}[\vec{y} \leftarrow \vec{u}].t$: $\llbracket u_k \rrbracket_i (\vec{a}) R^{A_k}_i u_k[\vec{t}/\vec{x}]$ by induction, so $\llbracket u_k \rrbracket_i (\vec{a}) R^{A_k}_{i+1} u_k[\vec{t}/\vec{x}]$ by Lemma 4.A.3. Then $\llbracket t \rrbracket_{i+1} (\llbracket u_1 \rrbracket_i (\vec{a}), \ldots) R^{\blacktriangleright A}_{i+1} t[u_1[\vec{t}/\vec{x}]/y_1, \ldots]$ by induction, so we have $t[u_1[\vec{t}/\vec{x}]/y_1, \ldots] \rightsquigarrow \mathsf{next}\,s$ with $\llbracket t \rrbracket_{i+1} (\llbracket u_1 \rrbracket_k (\vec{a}), \ldots) R^A_i s$. The left hand side is $\llbracket \mathsf{prev}[\vec{y} \leftarrow \vec{u}].t \rrbracket_i (\vec{a})$, while $\mathsf{prev}[\vec{y} \leftarrow \vec{u}[\vec{t}/\vec{x}]].t \mapsto \mathsf{prev}\,t[u_1[\vec{t}/\vec{x}]/y_1, \ldots] \rightsquigarrow \mathsf{prev}\,\mathsf{next}\,s \mapsto s$, so Lemma 4.A.1 completes.

$\mathsf{box}[\vec{y} \leftarrow \vec{u}].t$: To show $\llbracket \mathsf{box}[\vec{y} \leftarrow \vec{u}].t \rrbracket_i (\vec{a}) R^{\blacksquare A}_i \mathsf{box}[\vec{y} \leftarrow \vec{u}].t)[\vec{t}/\vec{x}]$, we observe that the right hand side reduces in one step to $\mathsf{box}\,t[u_1[\vec{t}/\vec{x}]/y_1, \ldots]$. The j'th element of the left hand side is $\llbracket t \rrbracket_j (\llbracket u_1 \rrbracket_k (\vec{a}), \ldots)$. We need to show this is related by R^A_j to $t[u_1[\vec{t}/\vec{x}]/y_1, \ldots]$; this follows by Lemma 4.A.3 and induction.

$\mathsf{unbox}\,t$: By induction $t[\vec{t}/\vec{x}] \rightsquigarrow \mathsf{box}\,u$, so $\mathsf{unbox}\,t[\vec{t}/\vec{x}] \rightsquigarrow \mathsf{unbox}\,\mathsf{box}\,u \mapsto u$. By induction $\llbracket t \rrbracket_i (\vec{a})_i R^A_i u$, so $\llbracket \mathsf{unbox}\,t \rrbracket_i (\vec{a}) R^A_i u$, and Lemma 4.A.1 completes.

$u_1 \circledast u_2$: Index 1 is trivial so set $i = j+1$. $\llbracket u_2 \rrbracket_{j+1} (\vec{a}) R^{\blacktriangleright A}_{j+1} u_2[\vec{t}/\vec{x}]$ implies $u_2[\vec{t}/\vec{x}] \rightsquigarrow \mathsf{next}\,s_2$ with $\llbracket u_2 \rrbracket_{j+1} (\vec{a}) R^A_j s_2$. Similarly $u_1 \rightsquigarrow \mathsf{next}\,s_1$ and $s_1 \rightsquigarrow \lambda x.s$ with $(\llbracket u_1 \rrbracket_{j+1} (\vec{a})_j) \circ \llbracket u_2 \rrbracket_{j+1} (\vec{a}) R^B_j s[s_2/x]$. The left hand side is exactly $\llbracket u_1 \circledast u_2 \rrbracket_{j+1} (\vec{a})$. Now $u_1 \circledast u_2 \rightsquigarrow \mathsf{next}\,s_1 \circledast u_2 \rightsquigarrow \mathsf{next}\,s_1 \circledast \mathsf{next}\,s_2 \mapsto \mathsf{next}(s_1 s_2)$, and $s_1 s_2 \rightsquigarrow (\lambda x.s)s_2 \mapsto s[s_2/x]$, completing the proof. 𝔔𝔈𝔇

4.B Example Proofs in $Lg\lambda$

We first record a substitution property of box and prev for later use.

Lemma 4.B.1. *Let $A_1, \ldots, A_k$ and B be constant types and C any type. If we have $x : B \vdash t : C$ and $y_1 : A_k, \ldots, y_k : A_k \vdash t' : B$ then*

$$\mathsf{box}\,[x \leftarrow t'].t =_{\blacksquare C} \mathsf{box}\,\iota.t[t'/x].$$

If $C = \blacktriangleright D$ then we also have

$$\mathsf{prev}\,[x \leftarrow t'].t =_D \mathsf{prev}\,\iota.t[t'/x]$$

◊

We can prove the first part of the lemma in the logic, using Proposition 4.4.2 and the β-rule for box. We can also prove the second part of the lemma for *total and inhabited types D* with the rules we have stated so far using the β-rule for next. For arbitrary D we can prove the lemma using the semantics.

Acausal Example

To see that Löb induction can be used to prove properties of recursively defined acausal functions we show that for any $n : \mathbf{N}$ and any $f : \mathbf{N} \to \mathbf{N}$ we have

$$\mathsf{every2nd}\,(\mathsf{box}\,\iota.\,\mathsf{iterate}\,(\mathsf{next}\,f\,)\,n) =_{\mathsf{Str}^{\mathsf{g}}} \mathsf{iterate}\,(\mathsf{next}\,f^{2})\,n,$$

where we write f^2 for $\lambda n. f\,(f\,n)$. We first derive the intermediate result

$$\forall m : \mathbf{N}, \mathsf{tl}\,(\mathsf{box}\,\iota.\,\mathsf{iterate}\,(\mathsf{next}\,f\,)\,m) =_{\mathsf{Str}} \mathsf{box}\,\iota.\,\mathsf{iterate}\,(\mathsf{next}\,f\,)\,(f\,m), \qquad (4.2)$$

by unfolding and applying Proposition 4.4.3:

$$\begin{aligned}
\mathsf{tl}\,(\mathsf{box}\,\iota.\,\mathsf{iterate}\,(\mathsf{next}\,f\,)\,m) &= \mathsf{box}\,[s \leftarrow \mathsf{box}\,\iota.\,\mathsf{iterate}\,(\mathsf{next}\,f\,)\,m].\,\mathsf{prev}\,\iota.\,\mathsf{tl}^{\mathsf{g}}(\mathsf{unbox}\,s) \\
&= \mathsf{box}\,\iota.\,\mathsf{prev}\,\iota.\,\mathsf{tl}^{\mathsf{g}}(\mathsf{iterate}\,(\mathsf{next}\,f\,)\,m) \\
&\qquad\qquad\qquad\qquad\qquad \text{(by Lemma 4.}B\text{.1)} \\
&= \mathsf{box}\,\iota.\,\mathsf{prev}\,\iota.\,\mathsf{next}\,(\mathsf{iterate}\,(\mathsf{next}\,f\,)\,(f\,m)) \\
&= \mathsf{box}\,\iota.\,\mathsf{iterate}\,(\mathsf{next}\,f\,)\,(f\,m).
\end{aligned}$$

Now assume

$$\triangleright\Big(\forall n : \mathbf{N}, \mathsf{every2nd}(\mathsf{box}\,\iota.\,\mathsf{iterate}\,(\mathsf{next}\,f\,)\,n) =_{\mathsf{Str}^{\mathsf{g}}} \mathsf{iterate}\,(\mathsf{next}\,f^{2})\,n\Big), \qquad (4.3)$$

then by Löb induction we can derive

$$\begin{aligned}
&\mathsf{every2nd}\,(\mathsf{box}\,\iota.\,\mathsf{iterate}\,(\mathsf{next}\,f\,)\,n) \\
&\qquad = n :: \mathsf{next}\,(\mathsf{every2nd}\,(\mathsf{tl}\,(\mathsf{tl}\,(\mathsf{box}\,\iota.\,\mathsf{iterate}\,(\mathsf{next}\,f\,)\,n)))) \\
&\qquad = n :: \mathsf{next}\,(\mathsf{every2nd}\,(\mathsf{box}\,\iota.\,\mathsf{iterate}\,(\mathsf{next}\,f\,)\,(f\,(f\,n)))) \qquad \text{(by 4.2)} \\
&\qquad = n :: \mathsf{next}\,(\mathsf{iterate}\,(\mathsf{next}\,f^{2})\,(f\,(f\,n))) \qquad \text{(by 4.3 and } \textsc{eq}^{\triangleright}_{\mathsf{next}}\text{)} \\
&\qquad = \mathsf{iterate}\,(\mathsf{next}\,f^{2})\,n.
\end{aligned}$$

Higher-Order Logic Example

We now prove

$$\begin{aligned}
&\forall P, Q : (\mathbb{N} \to \Omega), \forall f : \mathbb{N} \to \mathbb{N}, (\forall x : \mathbb{N}, P(x) \Rightarrow Q(f(x))) \\
&\Rightarrow \forall xs : \mathsf{Str}, P_{\mathsf{Str}^{\mathsf{g}}}(xs) \Rightarrow Q_{\mathsf{Str}^{\mathsf{g}}}(\mathsf{map}^{\mathsf{g}}\,f\,xs).
\end{aligned}$$

This is a simple property of $\mathsf{map}^{\mathsf{g}}$, but the proof shows how the pieces fit together. Recall that $\mathsf{map}^{\mathsf{g}}$ satisfies $\mathsf{map}^{\mathsf{g}}\,f\,xs = f\,(\mathsf{hd}^{\mathsf{g}}\,xs)::(\mathsf{next}(\mathsf{map}^{\mathsf{g}}\,f\,)\circledast(\mathsf{tl}^{\mathsf{g}}\,xs))$. We prove the property by Löb induction. So let P and Q be predicates on $\mathbb{N}$ and f a function on $\mathbb{N}$ that satisfies $\forall x : \mathbb{N}, P(x) \Rightarrow Q(f(x))$. To use Löb induction assume

$$\triangleright(\forall xs : \mathsf{Str}, P_{\mathsf{Str}^{\mathsf{g}}}(xs) \Rightarrow Q_{\mathsf{Str}^{\mathsf{g}}}(\mathsf{map}^{\mathsf{g}}\,f\,xs)) \qquad (4.4)$$

and let xs be a stream satisfying $P_{\mathsf{Str}^{\mathsf{g}}}$. Unfolding $P_{\mathsf{Str}^{\mathsf{g}}}(xs)$ we get $P(\mathsf{hd}^{\mathsf{g}}\,xs)$ and $\mathsf{lift}(\mathsf{next}\,P_{\mathsf{Str}^{\mathsf{g}}} \circledast (\mathsf{tl}^{\mathsf{g}}\,xs))$ and we need to prove $Q(\mathsf{hd}^{\mathsf{g}}(\mathsf{map}^{\mathsf{g}}\,f\,xs))$ and also $\mathsf{lift}(\mathsf{next}\,Q_{\mathsf{Str}^{\mathsf{g}}} \circledast (\mathsf{tl}^{\mathsf{g}}(\mathsf{map}^{\mathsf{g}}\,f\,xs)))$. The first is easy since $Q(\mathsf{hd}^{\mathsf{g}}(\mathsf{map}^{\mathsf{g}}\,f\,xs)) = Q(f\,(\mathsf{hd}^{\mathsf{g}}\,xs))$. For the second we have $\mathsf{tl}^{\mathsf{g}}(\mathsf{map}^{\mathsf{g}}\,f\,xs) = \mathsf{next}(\mathsf{map}^{\mathsf{g}}\,f) \circledast (\mathsf{tl}^{\mathsf{g}}\,xs)$. Since Str is a total and inhabited type there is a stream xs' such that $\mathsf{next}\,xs' = \mathsf{tl}^{\mathsf{g}}\,xs$. This gives $\mathsf{tl}^{\mathsf{g}}(\mathsf{map}^{\mathsf{g}}\,f\,xs) = \mathsf{next}(\mathsf{map}^{\mathsf{g}}\,f\,xs')$ and so our desired result reduces to $\mathsf{lift}(\mathsf{next}(Q_{\mathsf{Str}^{\mathsf{g}}}(\mathsf{map}^{\mathsf{g}}\,f\,xs')))$ and $\mathsf{lift}(\mathsf{next}\,P_{\mathsf{Str}^{\mathsf{g}}} \circledast (\mathsf{tl}^{\mathsf{g}}\,xs))$ is equivalent to $\mathsf{lift}(\mathsf{next}(P_{\mathsf{Str}^{\mathsf{g}}}(xs')))$. Now $\mathsf{lift} \circ \mathsf{next} = \triangleright$ and so what we have to prove is $\triangleright(Q_{\mathsf{Str}^{\mathsf{g}}}(\mathsf{map}^{\mathsf{g}}\,f\,xs'))$ from $\triangleright(P_{\mathsf{Str}^{\mathsf{g}}}(xs'))$, which follows directly from the induction hypothesis (4.4).

4.C Sums

This appendix extends Secs. 4.2, 4.3 and 4.4 to add sum types to the gλ-calculus. and to logic $L\mathrm{g}\lambda$.

Binary sums in Atkey and McBride [11] come with the type isomorphism $\blacksquare A + \blacksquare B \cong \blacksquare(A+B)$, but there are not in general terms witnessing this isomorphism. Likewise if binary sums are added to our calculus as obvious we may define the term

$$\lambda x.\,\mathsf{box}\,\iota.\,\mathsf{case}\,x\,\mathsf{of}\,x_1.\,\mathsf{in}_1\,\mathsf{unbox}\,x_1\,;x_2.\,\mathsf{in}_2\,\mathsf{unbox}\,x_2 : \blacksquare A + \blacksquare B \to \blacksquare(A+B)$$

but no inverse is definable in general. We believe such a map may be useful when working with guarded recursive types involving sum, such as the type of potentially infinite lists, and in any case the isomorphism is valid in the topos of trees and so it is harmless for us to reflect this in our calculus. We do this via a new term-former box^+ allowing us to define

$$\lambda x.\,\mathsf{box}^+\,\iota.\,\mathsf{unbox}\,x : \blacksquare(A+B) \to \blacksquare A + \blacksquare B$$

This construct may be omitted without effecting the results of this section.

Definition 4.C.1 (ref. Defs. 4.2.1,4.2.2,4.2.3,4.2.4,4.2.7,4.2.8)**.** gλ-*terms* are given by the grammar

$$t \quad ::= \quad \cdots \mid \mathsf{abort}\,t \mid \mathsf{in}_d\,t \mid \mathsf{case}\,t\,\mathsf{of}\,x_1.t;x_2.t \mid \mathsf{box}^+\,\sigma.t$$

where $d \in \{1,2\}$, and x_1, x_2 are variables. We abbreviate terms with box^+ as for prev and box.

The *reduction rules* on closed gλ-terms with sums are

$$\begin{array}{rcll} \mathsf{case}\,\mathsf{in}_d\,t\,\mathsf{of}\,x_1.t_1;x_2.t_2 & \mapsto & t_d[t/x_d] & (d \in \{1,2\}) \\ \mathsf{box}^+[\vec{x} \leftarrow \vec{t}].t & \mapsto & \mathsf{box}^+\,t[\vec{t}/\vec{x}] & (\vec{x}\textit{ non-empty}) \\ \mathsf{box}^+\,\mathsf{in}_i\,t & \mapsto & \mathsf{in}_i\,\mathsf{box}\,t & \end{array}$$

Values are terms of the form

$$\cdots \mid \mathsf{in}_1\,t \mid \mathsf{in}_2\,t$$

$$\frac{}{\nabla \vdash \mathbf{0}} \qquad \frac{\nabla \vdash A_1 \qquad \nabla \vdash A_2}{\nabla \vdash A_1 + A_2}$$

Figure 4.5: Type formation for sums in the gλ-calculus

$$\frac{\Gamma \vdash t : \mathbf{0}}{\Gamma \vdash \mathsf{abort}\, t : A} \qquad \frac{\Gamma \vdash t : A_d}{\Gamma \vdash \mathsf{in}_d\, t : A_1 + A_2}$$

$$\frac{\Gamma \vdash t : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash t_1 : A \qquad \Gamma, x_2 : A_2 \vdash t_2 : A}{\Gamma \vdash \mathsf{case}\, t\, \mathsf{of}\, x_1.t_1 ; x_2.t_2 : A}$$

$$\frac{\begin{array}{c} x_1 : A_1, \ldots, x_n : A_n \vdash t : B_1 + B_2 \\ \Gamma \vdash t_1 : A_1 \qquad \cdots \qquad \Gamma \vdash t_n : A_n \end{array}}{\Gamma \vdash \mathsf{box}^+[x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n].t : \blacksquare B_1 + \blacksquare B_2}\; A_1, \ldots, A_n\, \mathsf{constant}$$

Figure 4.6: Typing rules for sums in the gλ-calculus

Evaluation contexts are defined by the grammar

$$E \quad ::= \quad \cdots \mid \mathsf{abort}\, E \mid \mathsf{case}\, E\, \mathsf{of}\, x_1.t_1 ; x_2.t_2 \mid \mathsf{box}^+\, E$$

gλ-*types* for sums are defined inductively by the rules of Figure 4.5, and the new *typing judgments* are given in Figure 4.6, where $d \in \{1,2\}$. ♦

We now consider denotational semantics. Note that the initial object of $\mathcal{S}$ is $\Delta\emptyset$ (ref. Definition 4.3.2), while binary coproducts in $\mathcal{S}$ are defined pointwise. By naturality it holds that for any arrow $f : X \to Y + Z$ and $x \in X$, $f_i(x)$ must be an element of the same side of the sum for all i.

Definition 4.C.2 (ref. Defs. 4.3.3,4.3.7).

- $[\![\mathbf{0}]\!]$ is the constant functor $\Delta\emptyset$;
- $[\![A_1 + A_2]\!](\vec{W}) = [\![A_1]\!](\vec{W}) + [\![A_2]\!](\vec{W})$ and likewise for $\mathcal{S}$-arrows.

Term-formers for sums are intepreted via $\mathcal{S}$-coproducts, with abort, in_d and case defined as usual, and box^+ defined as follows.

- Let $[\![t]\!]_j([\![t_1]\!]_i(\gamma), \ldots, [\![t_n]\!]_i(\gamma))$ (which is well-defined by Lemma 4.3.6) be $[a_j, d]$ as j ranges, recalling that $d \in \{1,2\}$ is the same for all i. Define $a : 1 \to [\![A_d]\!]$ to have j'th element a_j. Then $\left[\!\left[\mathsf{box}^+[\vec{x} \leftarrow \vec{t}].t\right]\!\right]_i(\gamma) \triangleq [a, d]$.

♦

We now proceed to the sum cases of our proofs.

$\mathsf{box}^+[\vec{y} \leftarrow \vec{u}].t$ *case of Lemma 4.3.8.* By induction we have

$$\left[\!\!\left[u_k[\vec{t}/\vec{x}] \right]\!\!\right]_i (\gamma) = [\![u_k]\!]_i ([\![t_1]\!]_i (\gamma), \ldots).$$

Hence

$$[\![t]\!]_j \left(\left[\!\!\left[u_1[\vec{t}/\vec{x}] \right]\!\!\right]_i (\gamma), \ldots\right) = [\![t]\!]_j ([\![u_1]\!]_i ([\![t_1]\!]_i (\gamma), \ldots), \ldots)$$

as required. $\mathfrak{QED}$

box^+ *cases of Soundness Theorem 4.3.9.* Because each $[\![A_k]\!]$ is a constant object by Lemma 4.3.6, $[\![t_k]\!]_i = [\![t_k]\!]_j$ for all i, j. Hence $\left[\!\!\left[\mathsf{box}^+[\vec{x} \leftarrow \vec{t}].t \right]\!\!\right]_i$ is defined via components $[\![t]\!]_j ([\![t_1]\!]_j, \ldots)$ and $\left[\!\!\left[\mathsf{box}^+ \, t[\vec{t}/\vec{x}] \right]\!\!\right]$ is defined via components $\left[\!\!\left[t[\vec{t}/\vec{x}] \right]\!\!\right]_j$. These are equal by Lem 4.3.8.

$[\![\mathsf{box}^+ \, \mathsf{in}_d \, t]\!]_i$ is the d'th injection into the function with j'th component $[\![t]\!]_j$, and likewise for $[\![\mathsf{in}_d \, \mathsf{box} \, t]\!]_i$. $\mathfrak{QED}$

Definition 4.C.3 (ref. Definition 4.3.12).

- $[a, d] R_i^{A_1 + A_2} t$ iff $t \rightsquigarrow \mathsf{in}_d \, u$ for $d = 1$ or 2, and $a R_i^{A_d} u$.

Note that $R_i^{\mathbf{0}}$ is (necessarily) everywhere empty. ♦

for Lemmas 4.A.1 and 4.A.2. For $\mathbf{0}$ cases the premise fails so the the lemmas are vacuous. $+$ cases follow as for $\times$. $\mathfrak{QED}$

ref. Fundamental Lemma 4.3.13. abort: The induction hypothesis states that $[\![t]\!]_k (\vec{a}) R_k^{\mathbf{0}} t[\vec{t}/\vec{x}]$, but this is not possible, so the theorem holds vacuously.

$\mathsf{in}_d \, t$ case follows by easy induction.

$\mathsf{case} \, t \, \mathsf{of} \, y_1.u_1; y_2.u_2$: If $[\![t]\!]_i (\vec{a}) R_i^{A_1+A_2} t[\vec{t}/\vec{x}]$ then $t[\vec{t}/\vec{x}] \rightsquigarrow \mathsf{in}_d \, u$ for some $d \in \{1, 2\}$, with $[\![t]\!]_i (\vec{a}) = [a, d]$ and $a R_i^{A_d} u$. Then $[\![u_d]\!]_i (\vec{a}, a) R_k^A u_d[\vec{t}/\vec{x}, u/y_d]$. Now $(\mathsf{case} \, t \, \mathsf{of} \, y_1.u_1; y_2.u_2)[\vec{t}/\vec{x}] \rightsquigarrow \mathsf{case} \, \mathsf{in}_d \, u \, \mathsf{of} \, y_1.(u_1[\vec{t}/\vec{x}]); y_2.(u_2[\vec{t}/\vec{x}])$, which reduces to $u_d[\vec{t}/\vec{x}, u/y_i]$, and Lemma 4.A.1 completes.

$\mathsf{box}^+[\vec{y} \leftarrow \vec{u}].t$: $[\![u_k]\!]_i (\vec{a}) R_i^{A_k} u_k[\vec{t}/\vec{x}]$ by induction, so $[\![u_k]\!]_i (\vec{a}) R_j^{A_k} u_k[\vec{t}/\vec{x}]$ for any j by Lemma 4.A.3. By induction $[\![t]\!]_j ([\![u_1]\!]_k (\vec{a}), \ldots) R_j^{B_1+B_2} t[u_1[\vec{t}/\vec{x}]/y_1, \ldots]$. If $[\![t]\!]_j ([\![u_1]\!]_k (\vec{a}), \ldots)$ is some $[b_j, d]$ we have $t[u_1[\vec{t}/\vec{x}]/y_1, \ldots] \rightsquigarrow \mathsf{in}_d \, s$ for some s satisfying $b_j R_j^{B_d} s$. Now

$$(\mathsf{box}^+[\vec{y} \leftarrow \vec{u}].t)[\vec{t}/\vec{x}] \mapsto \mathsf{box}^+ \, t[u_1[\vec{t}/\vec{x}]/y_1, \ldots] \rightsquigarrow \mathsf{box}^+ \, \mathsf{in}_d \, s,$$

which finally reduces to $\mathsf{in}_d \, \mathsf{box} \, s$, which yields the result. $\mathfrak{QED}$

The logic $Lg\lambda$ may be extended to sums via the usual $\beta\eta$-laws and commuting conversions for binary sums and the equational version of the box^+ rule (ref. Figure 4.3):

$$\frac{\Gamma_{\blacksquare} \vdash t : B_d \qquad \Gamma \vdash \vec{t} : \Gamma_{\blacksquare}}{\Gamma \vdash \mathsf{box}^+[\vec{x} \leftarrow \vec{t}].(\mathsf{in}_d \, t) = \mathsf{in}_d(\mathsf{box}[\vec{x} \leftarrow \vec{t}].t)}$$

4.D Proof of Definability of Solutions of Behavioural Differential Equations in gλ

An equivalent presentation of the topos of trees is as sheaves over ω (with Alexandrov topology) $\mathbf{Sh}(\omega)$. In this section it is more convenient to work with sheaves than with presheaves because the global sections functor Γ[1] in the sequence of adjoints

$$\Pi_1 \dashv \Delta \dashv \Gamma$$

where

$$\begin{array}{ccc} \Pi_1 : \mathcal{S} \to \mathbf{Set} & \Delta : \mathbf{Set} \to \mathcal{S} & \Gamma : \mathcal{S} \to \mathbf{Set} \\ \Pi_1(X) = X(1) & \Delta(a)(\alpha) = \begin{cases} 1 & \text{if } \alpha = 0 \\ a & \text{otherwise} \end{cases} & \Gamma(X) = X(\omega) \end{array}$$

is just evaluation at ω, i.e. the limit is already present. This simplifies notation. Another advantage is that $\blacktriangleright : \mathcal{S} \to \mathcal{S}$ is given as

$$\begin{aligned} (\blacktriangleright X)(\nu + 1) &= X(\nu) \\ (\blacktriangleright X)(\alpha) &= X(\alpha) \end{aligned}$$

where α is a limit ordinal (either 0 or ω) which means that $\blacktriangleright X(\omega) = X(\omega)$ and as a consequence, $\mathbf{next}_\omega = \mathrm{id}_{X(\omega)}$ and $\Gamma(\blacktriangleright X) = \Gamma(X)$ for any $X \in \mathcal{S}$ and so $\blacksquare(\blacktriangleright X) = \blacksquare X$ for any X so we don't have to deal with mediating isomorphisms.

First we have a simple statement, but useful later, since it gives us a precise goal to prove later when considering the interpretation.

Lemma 4.D.1. *Let X, Y be objects of $\mathcal{S}$. Let $F : \blacktriangleright\left(Y^X\right) \to Y^X$ be a morphism in $\mathcal{S}$ and $\underline{F}$ a* function *in* $\mathbf{Set}$ *from $Y(\omega)^{X(\omega)}$ to $Y(\omega)^{X(\omega)}$. Suppose that the diagram*

$$\begin{tikzcd} \Gamma\left(\blacktriangleright\left(Y^X\right)\right) \arrow[r, "\Gamma(F)"] \arrow[d, "\mathsf{lim}"] & \Gamma(Y^X) \arrow[d, "\mathsf{lim}"] \\ Y(\omega)^{X(\omega)} \arrow[r, "\underline{F}"] & Y(\omega)^{X(\omega)} \end{tikzcd}$$

where $\mathsf{lim}\left(\{g_\nu\}_{\nu=0}^{\omega}\right) = g_\omega$ commutes. By Banach's fixed point theorem F has a unique fixed point, say $u : 1 \to Y^X$.

Then $\mathsf{lim}(\Gamma(u)(*)) = \mathsf{lim}(\Gamma(\mathbf{next} \circ u)(*)) = \Gamma(\mathbf{next} \circ u)(*)_\omega = u_\omega(*)_\omega$ is a fixed point of $\underline{F}$. ◊

[1] This standard notation for this functor should not to be confused with our notation for typing contexts.

Proof. The proof is trivial.

$$\begin{aligned}\underline{F}\,(\mathsf{lim}(\Gamma(u)(*))) &= \mathsf{lim}(\Gamma(F)(\Gamma(\mathbf{next}\circ u)(*)))\\ &= \mathsf{lim}(\Gamma(F\circ\mathbf{next}\circ u)(*)) = \mathsf{lim}(\Gamma(u)(*)).\end{aligned}$$

𝔔𝔈𝔇

Note that lim is not an isomorphism. There are (in general) many more functions from $X(\omega)$ to $Y(\omega)$ than those that arise from natural transformations. The ones that arise from natural transformations are the *non-expansive* ones.

Behavioural Differential Equations

Let Σ_A be a signature of function symbols with two types, A and Str. Suppose we wish to define a new k-ary operation given the signature Σ_A. We need to provide two terms h_f and t_f (standing for *head* and *tail*). h_f has to be a term using function symbols in signature Σ_A and have type

$$x_1 : A, x_2 : A, \cdots, x_k : A \vdash h_f : A$$

and t_f has to be a term in the signature extended with a new function symbol f of type $(\mathsf{Str})^k \to \mathsf{Str}$ and have type

$$x_1 : A, \cdots, x_k : A, y_1 : \mathsf{Str}, \cdots, y_k : \mathsf{Str}, z_1 : \mathsf{Str}, \cdots, z_k : \mathsf{Str} \vdash t_f : \mathsf{Str}$$

In the second term the variables x (intuitively) denote the head elements of the streams, the variables y denote the streams, and the variables z denote the tails of the streams.

We now define two interpretations of h_f and t_f. First in the topos of trees and then in **Set**.

We choose a set $a \in \mathbf{Set}$ and define $[\![A]\!]_{\mathcal{S}} = \Delta(a)$ and $[\![\mathsf{Str}]\!]_{\mathcal{S}} = \mu X.\Delta(a) \times \blacktriangleright(X)$. To each function symbol $g \in \Sigma$ of type $\tau_1, \ldots, \tau_n \to \tau_{n+1}$ we assign a morphism

$$[\![g]\!]_{\mathcal{S}} : [\![\tau_1]\!]_{\mathcal{S}} \times [\![\tau_2]\!]_{\mathcal{S}} \times \cdots \times [\![\tau_n]\!]_{\mathcal{S}} \to [\![\tau_{n+1}]\!]_{\mathcal{S}}.$$

Then we define the interpretation of h_f by induction as a morphism of type $[\![A]\!]_{\mathcal{S}}^k \to [\![A]\!]_{\mathcal{S}}$ by

$$\begin{aligned}[\![x_i]\!]_{\mathcal{S}} &= \pi_i\\ [\![g(t_1, t_2, \ldots, t_n)]\!]_{\mathcal{S}} &= [\![g]\!]_{\mathcal{S}} \circ \langle [\![t_1]\!]_{\mathcal{S}}, [\![t_2]\!]_{\mathcal{S}}, \cdots, [\![t_n]\!]_{\mathcal{S}} \rangle.\end{aligned}$$

For t_f we interpret the types and function symbols in Σ_A in the same way. But recall that t_f also contains a function symbol f. So the denotation of t_f will be a morphism with the following type

$$\left[\!\!\left[t_f \right]\!\!\right]_{\mathcal{S}} : \blacktriangleright\left([\![\mathsf{Str}]\!]_{\mathcal{S}}^{[\![\mathsf{Str}]\!]_{\mathcal{S}}^k} \right) \times [\![A]\!]_{\mathcal{S}}^k \times [\![\mathsf{Str}]\!]_{\mathcal{S}}^k \times (\blacktriangleright([\![\mathsf{Str}]\!]_{\mathcal{S}}))^k \to \blacktriangleright([\![\mathsf{Str}]\!]_{\mathcal{S}})$$

and is defined as follows

$$\begin{aligned}
[\![x_i]\!]_{\mathcal{S}} &= \mathbf{next} \circ \iota \circ \pi_{x_i} \\
[\![y_i]\!]_{\mathcal{S}} &= \mathbf{next} \circ \pi_{y_i} \\
[\![z_i]\!]_{\mathcal{S}} &= \pi_{z_i} \\
[\![g(t_1, t_2, \ldots, t_n)]\!]_{\mathcal{S}} &= \blacktriangleright([\![g]\!]_{\mathcal{S}}) \circ \mathbf{can} \circ \langle [\![t_1]\!]_{\mathcal{S}}, [\![t_2]\!]_{\mathcal{S}}, \cdots, [\![t_n]\!]_{\mathcal{S}} \rangle \qquad \text{if } g \neq f \\
[\![f(t_1, t_2, \ldots, t_k)]\!]_{\mathcal{S}} &= \mathbf{eval} \circ \left\langle J \circ \pi_f, \mathbf{can} \circ \langle [\![t_1]\!]_{\mathcal{S}}, [\![t_2]\!]_{\mathcal{S}}, \cdots, [\![t_k]\!]_{\mathcal{S}} \rangle \right\rangle
\end{aligned}$$

where **can** is the canonical isomorphism witnessing that ► preserves products, **eval** is the evaluation map and ι is the suitably encoded morphism that when given a constructs the stream with head a and tail all zeros. This exists and is easy to construct.

Next we define the denotation of h_f and t_f in **Set**. We set $[\![A]\!]_{\mathbf{Set}} = a$ and $[\![\mathsf{Str}]\!]_{\mathbf{Set}} = [\![\mathsf{Str}]\!]_{\mathcal{S}}(\omega)$. For each function symbol in Σ_A we define $[\![g]\!]_{\mathbf{Set}} = \Gamma[\![g]\!]_{\mathcal{S}} = \left([\![g]\!]_{\mathcal{S}}\right)_{\omega}$.

We then define $\left[\!\left[h_f\right]\!\right]_{\mathbf{Set}}$ as a function

$$[\![A]\!]_{\mathbf{Set}}^k \to [\![A]\!]_{\mathbf{Set}}$$

exactly the same as we defined $\left[\!\left[h_f\right]\!\right]_{\mathcal{S}}$.

$$\begin{aligned}
[\![x_i]\!]_{\mathbf{Set}} &= \pi_i \\
[\![g(t_1, t_2, \ldots, t_n)]\!]_{\mathbf{Set}} &= [\![g]\!]_{\mathbf{Set}} \circ \langle [\![t_1]\!]_{\mathbf{Set}}, [\![t_2]\!]_{\mathbf{Set}}, \cdots, [\![t_n]\!]_{\mathbf{Set}} \rangle.
\end{aligned}$$

The denotation of t_f is somewhat different in the way that we do not guard the tail and the function being defined with a ►. We define

$$\left[\!\left[t_f\right]\!\right]_{\mathbf{Set}} : [\![\mathsf{Str}]\!]_{\mathbf{Set}}^{[\![\mathsf{Str}]\!]_{\mathbf{Set}}^k} \times [\![A]\!]_{\mathbf{Set}}^k \times [\![\mathsf{Str}]\!]_{\mathbf{Set}}^k \times ([\![\mathsf{Str}]\!]_{\mathbf{Set}})^k \to [\![\mathsf{Str}]\!]_{\mathbf{Set}}$$

as follows

$$\begin{aligned}
[\![x_i]\!]_{\mathbf{Set}} &= \iota \circ \pi_{x_i} \\
[\![y_i]\!]_{\mathbf{Set}} &= \pi_{y_i} \\
[\![z_i]\!]_{\mathbf{Set}} &= \pi_{z_i} \\
[\![g(t_1, t_2, \ldots, t_n)]\!]_{\mathbf{Set}} &= [\![g]\!]_{\mathbf{Set}} \circ \langle [\![t_1]\!]_{\mathbf{Set}}, [\![t_2]\!]_{\mathbf{Set}}, \cdots, [\![t_n]\!]_{\mathbf{Set}} \rangle \qquad \text{if } g \neq f \\
[\![f(t_1, t_2, \ldots, t_k)]\!]_{\mathbf{Set}} &= \mathbf{eval} \circ \left\langle \pi_f, \langle [\![t_1]\!]_{\mathbf{Set}}, [\![t_2]\!]_{\mathbf{Set}}, \cdots, [\![t_k]\!]_{\mathbf{Set}} \rangle \right\rangle
\end{aligned}$$

where ι is again the same operation, this time on actual streams in **Set**.

We then define

$$\underline{F} : [\![\mathsf{Str}]\!]_{\mathbf{Set}}^{[\![\mathsf{Str}]\!]_{\mathbf{Set}}^k} \to [\![\mathsf{Str}]\!]_{\mathbf{Set}}^{[\![\mathsf{Str}]\!]_{\mathbf{Set}}^k}$$

as

$$\underline{F}(\varphi)(\vec{\sigma}) = \Gamma(\mathbf{fold})\left(\left(\left[\!\left[h_f\right]\!\right]_{\mathbf{Set}}(\mathbf{hd}(\vec{\sigma})), \left[\!\left[t_f\right]\!\right]_{\mathbf{Set}}(\varphi, \mathbf{hd}(\vec{\sigma}), \vec{\sigma}, \mathbf{tl}(\vec{\sigma}))\right)\right)$$

where **hd** and **tl** are head and tail functions (extended in the obvious way to tuples). Here **fold** is the isomorphism witnessing that guarded streams are indeed the fixed point of the defining functor.

Similarly we define

$$F : \blacktriangleright\left([\![\mathsf{Str}]\!]_{\mathcal{S}}^{[\![\mathsf{Str}]\!]_{\mathcal{S}}^{k}}\right) \to [\![\mathsf{Str}]\!]_{\mathcal{S}}^{[\![\mathsf{Str}]\!]_{\mathcal{S}}^{k}}$$

as the exponential transpose Λ of

$$F' = \mathbf{fold} \circ \left\langle [\![h_f]\!] \circ \vec{hd} \circ \pi_2, [\![t_f]\!]_{\mathcal{S}} \circ \left(\mathrm{id}_{\blacktriangleright\left([\![\mathsf{Str}]\!]_{\mathcal{S}}^{[\![\mathsf{Str}]\!]_{\mathcal{S}}^{k}}\right)} \times \left\langle \vec{\mathbf{hd}}, \mathrm{id}_{[\![\mathsf{Str}]\!]_{\mathcal{S}}^{k}}, \vec{\mathbf{tail}} \right\rangle \right) \right\rangle$$

Proposition 4.D.2. *For the above defined F and $\underline{F}$ we have*

$$\mathsf{lim} \circ \Gamma(F) = \underline{F} \circ \mathsf{lim}$$

◊

Proof. Let $\varphi \in \Gamma\left(\blacktriangleright\left([\![\mathsf{Str}]\!]_{\mathcal{S}}^{[\![\mathsf{Str}]\!]_{\mathcal{S}}^{k}}\right)\right) = \Gamma\left([\![\mathsf{Str}]\!]_{\mathcal{S}}^{[\![\mathsf{Str}]\!]_{\mathcal{S}}^{k}}\right)$. We have

$$\mathsf{lim}(\Gamma(F)(\varphi)) = \mathsf{lim}\,(F_\omega(\varphi)) = F_\omega(\varphi)_\omega$$

and

$$\underline{F}(\mathsf{lim}(\varphi)) = \underline{F}\,(\varphi_\omega)$$

Now both of these are elements of $[\![\mathsf{Str}]\!]_{\mathbf{Set}}^{[\![\mathsf{Str}]\!]_{\mathbf{Set}}^{k}}$, meaning genuine functions in **Set**, so to show they are equal we use elements. Let $\vec{\sigma} \in [\![\mathsf{Str}]\!]_{\mathbf{Set}}^{k}$.

We are then required to show

$$\underline{F}\,(\varphi_\omega)\,(\vec{\sigma}) = F_\omega(\varphi)_\omega(\vec{\sigma})$$

Recall that $F = \Lambda(F')$ (exponential transpose) so $F_\omega(\varphi)_\omega(\vec{\sigma}) = F'_\omega(\varphi, \vec{\sigma})$. Now recall that composition in $\mathcal{S}$ is just composition of functions at each stage and products in $\mathcal{S}$ are defined pointwise and that $\mathbf{next}_\omega$ is the identity function.

Moreover, the morphism **hd** gets mapped by Γ to **hd** in **Set** and the same holds for **tl**. For the latter it is important that $\Gamma(\blacktriangleright(X)) = \Gamma(X)$ for any X.

We thus get

$$F'_\omega(\varphi, \vec{\sigma}) = \mathbf{fold}_\omega\left(([\![h_f]\!]_{\mathcal{S}})_\omega\,(\mathbf{hd}(\vec{\sigma})), \left([\![t_f]\!]_{\mathcal{S}}\right)_\omega (\varphi, \mathbf{hd}(\vec{\sigma}), \vec{\sigma}, \mathbf{tl}(\vec{\sigma}))\right)$$

And for $\underline{F}\,(\varphi_\omega)\,(\vec{\sigma})$ we have

$$\underline{F}\,(\varphi_\omega)\,(\vec{\sigma}) = \mathbf{fold}_\omega\left([\![h_f]\!]_{\mathbf{Set}}\,(\mathbf{hd}\,(\vec{\sigma})), \left([\![t_f]\!]_{\mathbf{Set}}\right)(\varphi_\omega, \mathbf{hd}(\vec{\sigma}), \vec{\sigma}, \mathbf{tl}(\vec{\sigma}))\right)$$

It is now easy to see that these two are equal. The proof is by induction on the structure of h_f and t_f. The variable cases are trivial, but crucially use the fact that $\mathbf{next}_\omega$ is the identity. The cases for function symbols in Σ_A are trivial since their denotations in **Set** are defined to be the correct ones. The case for f goes through similarly since application at ω only uses φ at ω. $\mathfrak{QED}$

Theorem 4.D.3. *Let (Σ_1, Σ_2) be a signature and $\mathcal{I}$ its interpretation. Let (h_f, t_f) be a behavioural differential equation defining a k-ary function f using function symbols in Σ. The right-hand sides of h_f and t_f define a term Φ_f^{g} of type*

$$\Phi_f^{\mathsf{g}} : \blacktriangleright(\underbrace{\mathsf{Str}^{\mathsf{g}} \to \mathsf{Str}^{\mathsf{g}} \to \cdots \mathsf{Str}^{\mathsf{g}}}_{k+1}) \to (\underbrace{\mathsf{Str}^{\mathsf{g}} \to \mathsf{Str}^{\mathsf{g}} \to \cdots \mathsf{Str}^{\mathsf{g}}}_{k+1}).$$

and a term Φ_f of type

$$\Phi_f : (\underbrace{\mathsf{Str} \to \mathsf{Str} \to \cdots \mathsf{Str}}_{k+1}) \to (\underbrace{\mathsf{Str} \to \mathsf{Str} \to \cdots \mathsf{Str}}_{k+1}).$$

by using $\mathcal{L}_{a_j^g}\left(\mathcal{I}(g_j)\right)$ for interpretations of function symbols g_j.

Let $f^{\mathsf{g}} = \mathsf{fix}\Phi_f^{\mathsf{g}}$ be the fixed point of Φ_f^{g}. Then $f = \mathcal{L}_k(\mathsf{box} f^{\mathsf{g}})$ is a fixed point of Φ_f which in turn implies that it satisfies equations h_f and t_f. ◊

Proof. Use Proposition 4.D.2 together with Lemma 4.D.1 together with the observation that **Set** is a full subcategory of $\mathcal{S}$ with Δ being the inclusion.

We also use the fact that for a closed term $u : A \to B$ (which is interpreted as a morphism from 1 to B^A) the denotation of $\mathcal{L}(u)$ at stage ν and argument $*$ is $\mathsf{lim}(\Gamma(u)(*))$. $\mathfrak{QED}$

Discussion

What we have shown is that for each behavioural differential equation that defines a function on streams and *can be specified as a standalone function depending only on previously defined functions*, i.e. it is not defined mutually with some other function, there is a fixed point. It is straightforward to extend to mutually recursive definitions by defining a product of functions in the same way as we did for a single function, but notationally this gets quite heavy.

More importantly, suppose we start by defining an operation f on streams first, and the only function symbols in Σ_A operate on A, i.e. all have type $A^k \to A$ for some k. Assume that these function symbols are given denotations in $\mathcal{S}$ as $\Delta(g)$ for some function g in **Set**. Then the denotation in **Set** is just g.

The fixed point f *in* $\mathcal{S}$ is then a morphism from 1 to the suitable exponential. Let $\overline{f}$ be the uncurrying of f. Then $\mathsf{lim}(\Gamma(f)(*)) = \Gamma(\overline{f})$.

Thus if we continue defining new functions which use f, we then choose $\overline{f}$ as the denotation of the function symbol f. The property $\mathsf{lim}(\Gamma(u)(*)) = \Gamma(\overline{f})$ then says that the f that is used in the definition is the f that was defined previously.

4.E About Total and Inhabited Types

An object in $\mathcal{S}$ is *total and inhabited* if all components are non-empty and all restriction functions are surjective. We have the following easy proposition.

Proposition 4.E.1. *Let $P : \mathcal{S} \to \mathcal{S}$ be a functor such that if X is a total and inhabited object, so is $P(X)$, i.e. P restricts to the full subcategory of total and inhabited objects.*

If P is locally contractive then its fixed point is total and inhabited. ◊

Proof. We use the equivalence between the full subcategory $ti\mathcal{S}$ of $\mathcal{S}$ of *total and inhabited* objects and the category of complete bisected *non-empty* ultrametric spaces $\mathcal{M}$. We know that the category $\mathcal{M}$ is an M-category [20] and thus so is $ti\mathcal{S}$. It is easy to see that locally contractive functors in $\mathcal{S}$ are locally contractive in the M-category sense. Hence if P is locally contractive and restricts to $ti\mathcal{S}$ its fixed point is in $ti\mathcal{S}$. $\mathfrak{QED}$

Corollary 4.E.2. *Let P be a non-zero polynomial functor whose coefficients and exponents are total and inhabited. The functor $P \circ \blacktriangleright$ is locally contractive and its unique fixed point is total and inhabited.* ◊

Proof. Products and non-empty coproducts of total and inhabited objects are total and inhabited. Similarly, if X and Y are total and inhabited, so is X^Y. So any non-zero polynomial functor P whose coefficients are all total and inhabited restricts to $ti\mathcal{S}$. The functor $\blacktriangleright$ restricts to $ti\mathcal{S}$ as well (but note that it *does not* restrict to the subcategory of total objects $t\mathcal{S}$). Polynomial functors on $\mathcal{S}$ are also strong and so the functor $P \circ \blacktriangleright$ is locally contractive. Hence by Proposition 4.E.1 its unique fixed point is a total and inhabited object. $\mathfrak{QED}$

In particular guarded streams of any total and inhabited type themselves form a total and inhabited type.

Chapter 5

A Model of Guarded Recursion with Clock Synchronisation

This chapter is a revised and extended version of

> Ale Bizjak and Rasmus Ejlers Mgelberg.
>
> A model of guarded recursion with clock synchronisation.
>
> *Electronic Notes in Theoretical Computer Science*, 319:83 – 101, 2015.
>
> The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).

The section describing the syntax and the typing rules of the calculus was removed since it is superseded by Chapter 7 of this thesis. Instead the sections describing the model are now extended to include more proofs and explanations.

Abstract

Guarded recursion is an approach to solving recursive type equations where the type variable appears guarded by a modality to be thought of as a delay for one time step. Atkey and McBride proposed a calculus in which guarded recursion can be used when programming with coinductive data, allowing productivity to be captured in types. The calculus uses *clocks* representing time streams and *clock quantifiers* which allow limited and controlled elimination of modalities. The calculus has since been extended to dependent types by Møgelberg.

In previous versions of this calculus, different clocks represented separate time streams and *clock synchronisation* was prohibited. In this paper we show that allowing clock synchronisation is safe by constructing a new model of guarded recursion and clocks. This result will simplify the type theory by removing freshness restrictions from typing rules.

5.1 Introduction

Guarded recursion [72] is an approach to solving recursive type equations where the type variable appears *guarded* by a $\blacktriangleright$ (pronounced "later") modal type operator. In particular the type variable could appear positively or negatively or both, e.g. the equation $\sigma = 1 + \blacktriangleright(\sigma \to \sigma)$ has a unique solution [22]. On the term level the *guarded fixed point combinator* $\mathsf{fix}_\tau : (\blacktriangleright \tau \to \tau) \to \tau$ satisfies the equation $f\,(\mathsf{next}\,(\mathsf{fix}_\tau f)) = \mathsf{fix}_\tau f$ for any $f : \blacktriangleright \tau \to \tau$. Here $\mathsf{next} : \tau \to \blacktriangleright \tau$ is an operation that "freezes" an element that we have available now so that it is only available in the next time step.

One situation where guarded recursive types are useful is when faced with an unsolvable type equation. These arise for example when modelling programming languages with sophisticated features. In this case a solution to a guarded version of the equation often turns out to suffice, as shown in [22].

But guarded recursive versions of polymorphic type equations are also useful in type theory, even in settings where inductive and coinductive solutions to these equations are assumed to exist. To see this, consider the coinductive type of streams Str, i.e., the final coalgebra for the functor $\mathbb{S}(X) = \mathbb{N} \times X$. Proof assistants like Coq [68] and Agda [73] allow programmers to construct streams using recursive definitions, but to ensure consistency, these must be *productive*, i.e., one must be able to compute the first n elements of a stream in finite time. Coq and Agda inspect recursive definitions for productivity by a *syntactic property* that is often overly conservative and does not interact well with higher-order functions.

Using the type of *guarded streams* $\mathrm{Str}^{\mathrm{g}}$, i.e., the *unique* type satisfying the equation $\mathrm{Str}^{\mathrm{g}} = \mathbb{N} \times \blacktriangleright \mathrm{Str}^{\mathrm{g}}$, one can encode productivity in types: a productive recursive stream definition is exactly a term of type $\blacktriangleright \mathrm{Str}^{\mathrm{g}} \to \mathrm{Str}^{\mathrm{g}}$. To combine the benefits of coinductive and guarded recursive types, Atkey and McBride [11] suggested a simply typed calculus with clock variables κ representing time streams, each with associated $\blacktriangleright^\kappa$ type constructors, and universal quantification over clocks $\forall\kappa$. If we think of the type τ as being time-indexed along κ, then the type $\forall\kappa.\tau$ contains only elements which are available for all time steps. The relationship between the two notions of streams can then be captured by the encoding of the coinductive stream type as $\mathrm{Str} = \forall\kappa.\mathrm{Str}^{\mathrm{g}\kappa}$. This encoding works for a general class of coinductive types including those given by polynomial functors, and these results were since extended to the dependently typed setting by Møgelberg [71]. In both cases the encodings were proved sound with respect to a denotational model.

Clock synchronisation In the calculus for guarded recursion with clocks, typing judgements are given in a context of clocks Δ, which is just a finite set of names for clocks, as well as a context of term variables Γ. Clock variables κ are simply names, there are no constants or operations on them, and there is no type of clocks. The introduction and elimination rules for $\forall\kappa$ as defined by Atkey and McBride [11] are

$$\frac{\Gamma \vdash_{\Delta,\kappa} t : \tau}{\Gamma \vdash_{\Delta} \Lambda\kappa.t : \forall\kappa.\tau} \qquad \frac{\Gamma \vdash_{\Delta,\kappa'} t : \forall\kappa.\tau \qquad \kappa' \notin \forall\kappa.\tau}{\Gamma \vdash_{\Delta,\kappa'} t[\kappa'] : \tau[\kappa'/\kappa]} \tag{5.1}$$

These rules are very similar to those for polymorphic types in System F [46], except for the freshness side condition on the elimination rule ensuring that the clocks κ and κ' are not synchronised in τ. The side condition makes the rule syntactically not well-behaved. For instance it is not clear that the β-rule for clock application preserves types.

This becomes a more serious problem in dependent type theory. The rule Møgelberg [71] considers for clock instantiation is

$$\frac{\kappa \notin \mathrm{fc}(\Gamma) \qquad \Gamma \vdash_{\Delta,\kappa} \tau \text{ type} \qquad \Gamma, \Gamma' \vdash_{\Delta,\kappa'} t : \forall\kappa.\tau}{\Gamma, \Gamma' \vdash_{\Delta,\kappa'} t[\kappa'] : \tau[\kappa'/\kappa]}$$

where the side condition requires that none of the types τ depends on contain the clock κ. The reason for the additional clock context Γ' is to ensure that the calculus is closed under weakening. However, closure under substitution was overlooked and the rules do not appear to be sufficient to derive the substitution property such as

$$\frac{\Gamma, x : \tau \vdash_{\Delta} t : \sigma \qquad \Gamma \vdash_{\Delta} s : \tau}{\Gamma \vdash_{\Delta} t[s/x] : \sigma[s/x]}$$

which is necessary for a well-behaved dependent type theory.

The restriction on clock instantiation comes from the denotational models of guarded recursion. The original work on guarded recursion [18, 22] models a type as a presheaf over the ordered natural numbers, i.e., a diagram of the form

$$X(1) \longleftarrow X(2) \longleftarrow X(3) \longleftarrow \cdots$$

For example, the guarded recursive type of streams satisfying $\mathrm{Str}^g = \mathbb{N} \times \blacktriangleright \mathrm{Str}^g$ is modelled by the presheaf with $X(n) = \mathbb{N}^n$. In this model $\blacktriangleright$ shifts a type one step to the right inserting a singleton set in the end of the sequence.

This model was generalised by Møgelberg [71] (Atkey and McBride [11] use essentially the same idea) to multiple clocks by simply indexing by multiple copies of natural numbers. Thus, conceptually, a type with clocks κ_1 and κ_2 was modelled as a two dimensional diagram of sets (as in the left part of Figure 5.1). In this model there is no semantic correspondent to clock substitution. In particular, if τ is a type with two free clocks κ_1 and κ_2, then the denotation of $\tau[\kappa_1/\kappa_2]$ should be a one dimensional diagram, but this is in general not the diagonal of the denotation of τ, as one might expect. Semantically, one reason is that taking the diagonal does not commute, for instance, with the cartesian closed structure.

$$\begin{tikzcd}
\vdots \arrow[d] & \vdots \arrow[d] & \vdots \arrow[d] & & \vdots \arrow[d] \\
X(1,3) \arrow[d] & X(2,3) \arrow[l] \arrow[d] & X(3,3) \arrow[l] \arrow[d] \arrow[rr, bend right] & \cdots \arrow[l] & X(3) \arrow[d] \\
X(1,2) \arrow[d] & X(2,2) \arrow[l] \arrow[d] \arrow[rrr, bend right] & X(3,2) \arrow[l] \arrow[d] & \cdots \arrow[l] & X(2) \arrow[d] \\
X(1,1) \arrow[rrrr, bend right] & X(2,1) \arrow[l] & X(3,1) \arrow[l] & \cdots \arrow[l] & X(1)
\end{tikzcd}$$

Figure 5.1: A type with two free clocks in the new model.

We propose a new model which supports clock substitution that preserves all the constructs of type theory in the correct way. The model verifies soundness (up to solving the coherence problem, see Section 5.6) of the rules (5.1) as understood in dependent type theory, but without the freshness side condition on the elimination rule. Technically, this makes the calculus an instance of *polymorphic dependent type theory* [51] with a family of modalities $\blacktriangleright^{\kappa}$ indexed by clocks κ.

In the new model a type depending on two clocks κ_1 and κ_2 is modelled as a commutative diagrams of the form in Figure 5.1: the two dimensional grid on the left represents the type X when clocks κ_1 and κ_2 are not identified and the vertical diagram on the right represents the type X when clocks κ_1 and κ_2 become synchronised. The arrows inside the two and one dimensional diagrams describe the evolution of elements when the clocks decrease and the arrows from the diagonal of the diagram on the left to the diagram on the right describe how the elements change when the clocks are synchronised. This also explains why there are no arrows from the vertical diagram on the right to the diagram on the left. Once the clocks are identified there is no way to disentangle them. To model the substitution κ_1/κ_2 we simply take the right vertical part of the diagram.

With more clocks the denotation of a type becomes more complex. For instance when we have three clocks the denotation will have a three dimensional diagram (representing the state when none of the clocks are identified), three two dimensional diagrams (representing the state when two of the clocks are identified) and a one dimensional diagram, representing the state when all of the clocks are identified. Arrows between the different diagrams are given according to the following schema

$$\begin{tikzcd}
 & \kappa_1, \kappa_2, \kappa_3 \arrow[dl, bend right] \arrow[d] \arrow[dr, bend left] & \\
\kappa_1 = \kappa_2, \kappa_3 \arrow[dr, bend right] & \kappa_1 = \kappa_3, \kappa_2 \arrow[d] & \kappa_1, \kappa_2 = \kappa_3 \arrow[dl, bend left] \\
 & \kappa_1 = \kappa_2 = \kappa_3 &
\end{tikzcd}$$

where, for example, $\kappa_1 = \kappa_2, \kappa_3$ represents the diagram where clocks κ_1 and κ_2 are identified, and κ_3 is independent of the two.

5.2 The Basics of the New Model

We fix a countable set of clocks $\mathrm{CV} = \{\kappa_1, \kappa_2, \ldots\}$. The model we construct can be briefly described as follows. Let CV be the full subcategory of the category **Set** on *finite subsets* of CV. We build an indexed category $\mathfrak{GR}$ on the opposite of the category CV. For each finite set of clocks Δ, the category $\mathfrak{GR}(\Delta)$ is a model of extensional dependent type theory: term variable contexts $\Gamma \vdash_\Delta$, types $\Gamma \vdash_\Delta A\,\mathrm{type}$ and terms $\Gamma \vdash_\Delta t : A$ are interpreted in $\mathfrak{GR}(\Delta)$. For any $f : \Delta_1 \to \Delta_2$ the reindexing functor $\mathfrak{GR}(f) : \mathfrak{GR}(\Delta_1) \to \mathfrak{GR}(\Delta_2)$, which is used to model clock substitution, preserves all the structure required for modeling dependent type theory. Finally, for any inclusion $\iota : \Delta \to \Delta, \kappa$ the reindexing functor $\mathfrak{GR}(\iota) : \mathfrak{GR}(\Delta) \to \mathfrak{GR}(\Delta, \kappa)$ has a right adjoint $\forall\kappa$ which is used to interpret quantification over clocks.

By applying the Grothendieck construction to the indexed category $\mathfrak{GR}$ we can then construct a *PDTT*-structure [51] which is a notion of a model of polymorphic dependent type theory. The PDTT-structure in our case is a collection of fibrations and comprehension categories

$$\begin{tikzcd}
\mathbb{D} \arrow[r, "\mathcal{P}"] \arrow[dr, "q"] & \mathbb{A}^{\to} \arrow[d, "\mathsf{cod}"] & & \\
 & \mathbb{A} \arrow[dr, "r"] & \mathbb{B} \arrow[r, "\mathcal{Q}"] \arrow[d, "\mathrm{id}_{\mathbb{B}}"] & \mathbb{B}^{\to} \arrow[dl, "\mathsf{cod}"] \\
 & & \mathbb{B} &
\end{tikzcd} \tag{5.2}$$

where $\mathbb{B} = \mathrm{CV}^{\mathrm{op}}$ and $\mathcal{Q}$ maps $\Delta \subseteq \mathrm{CV}$ to the inclusion $\iota : \Delta \to \Delta, \kappa_\Delta$, where κ_Δ is the chosen clock not in Δ. The fibration r is the fibration arising from the Grothendieck construction applied to the indexed category $\mathfrak{GR}$, hence objects of $\mathbb{A}$ are pairs (Δ, X) with $\Delta \subseteq^{\mathrm{fin}} \mathrm{CV}$ and $X \in \mathfrak{GR}(\Delta)$. The functor r simply maps (Δ, X) to Δ. The comprehension $\mathcal{P}$ is constructed from comprehensions of $\mathfrak{GR}(\Delta)$.

The indexed category $\mathfrak{GR}$

For each $\Delta \subseteq^{\text{fin}} \mathrm{CV}$ the category $\mathfrak{GR}(\Delta)$ is the category of presheaves over the poset $\mathfrak{I}(\Delta)$ which we describe first. To understand the definition of the poset $\mathfrak{I}(\Delta)$ it is useful to keep in mind the example in Figure 5.1. Let Δ be a finite set of clocks. An element $\mathfrak{I}(\Delta)$ should indicate what is the state of clocks, i.e., which clock are identified, and it should indicate how much time is left on each clock. Hence elements of $\mathfrak{I}(\Delta)$ should be pairs (E,δ) of an equivalence relation E on Δ and a function $\delta : \Delta \to \mathbb{N}$. Since identified clocks should have the same amount of time remaining, the function δ should respect E. The order on $\mathfrak{I}(\Delta)$ should allow us to get from state represented by (E,δ) to (E',δ') whenever E' identifies more clocks than E and there is no more time left on δ' than on δ. This makes sense because we want to be able to substitute clocks, and substitution, in general, can identify clocks. On the other hand once the clocks are identified we can no longer separate them, hence we should not be able to get from a state where more clocks are identified to a state where fewer of them are. With this in mind, here are the precise definitions.

Definition 5.2.1. For $\Delta \subseteq^{\text{fin}} \mathrm{CV}$ let $\mathfrak{E}(\Delta)$ be the set of equivalence relations on Δ (considered as subsets of $\Delta \times \Delta$).

The order relation on $\mathfrak{E}(\Delta)$ is the *opposite of the refinement order*, concretely

$$E \geq E' \leftrightarrow E \subseteq E'$$

(note the reverse inclusion). Or in other words, $E' \leq E$ if whenever two elements are related by E, they are also related by E'.

The top element for this ordering is the diagonal relation $\mathfrak{d}_\Delta$. The bottom element is the relation that equates everything.

For a function $f : \Delta_1 \to \Delta_2$ let $\mathfrak{E}(f) : \mathfrak{E}(\Delta_2) \to \mathfrak{E}(\Delta_1)$ be the function defined by pullback as

$$\mathfrak{E}(f)(E) = \{(\kappa_1,\kappa_2) \mid (f(\kappa_1), f(\kappa_2)) \in E\},$$

i.e., clocks κ_1 and κ_2 are related by $\mathfrak{E}(f)(E)$ if they become related by E after substitution with f. ♦

Definition 5.2.2. Let Δ be a finite set of clocks. The poset $\mathfrak{I}(\Delta)$ has elements pairs (E,δ) where $E \in \mathfrak{E}(\Delta)$ is an equivalence relation and $\delta : \Delta \to \mathbb{N}$ is a function that respects E. This means that if $(\kappa_1,\kappa_2) \in E$ then $\delta(\kappa_1) = \delta(\kappa_2)$.

The order on $\mathfrak{I}(\Delta)$ is component-wise:

$$(E,\delta) \geq (E',\delta') \leftrightarrow E \geq E' \land \delta \geq \delta'.$$

where the ordering on functions is pointwise.

For a function $f : \Delta_1 \to \Delta_2$ the function $\mathfrak{I}(f) : \mathfrak{I}(\Delta_2) \to \mathfrak{I}(\Delta_1)$ is defined as

$$\mathfrak{I}(f)(E,\delta) = (\mathfrak{E}(f)(E), \delta \circ f).$$ ♦

Lemma 5.2.3. *For any function $f : \Delta_1 \to \Delta_2$ the functions $\mathfrak{E}(f)$ and $\mathfrak{I}(f)$ are monotone. Hence $\mathfrak{E}$ and $\mathfrak{I}$ are functors from* CV^{op} *to the category of partially ordered sets and monotone functions.* ◊

The proof is a simple computation.

For use in Section 5.5 below we record a property of surjective substitutions. It is proved by a simple computation.

Lemma 5.2.4. *Let $f : \Delta_1 \to \Delta_2$ be a function between clock contexts. If f is* surjective *then $\mathfrak{E}(f)$ and $\mathfrak{I}(f)$ are* injective. ◊

More abstractly, this property also follows from the fact that in CV any surjective function s is a split epimorphism, hence $\mathfrak{E}(s)$ and $\mathfrak{I}(s)$ are split monomorphisms and monomorphisms in the category of posets and monotone functions are precisely the injective monotone functions.

We also have a dual lemma. Its proof is again simple.

Lemma 5.2.5. *Let $f : \Delta_1 \to \Delta_2$ be a function between clock contexts. If f is* injective *then $\mathfrak{E}(f)$ and $\mathfrak{I}(f)$ are* surjective. ◊

Definition 5.2.6. Let Δ be a finite set of clocks. The category $\mathfrak{GR}(\Delta)$ is the category $\mathbf{Set}^{(\mathfrak{I}(\Delta))^{\mathrm{op}}}$ of (contravariant) $\mathfrak{I}(\Delta)$-indexed set valued presheaves.

For a function $f : \Delta_1 \to \Delta_2$ let $\mathfrak{GR}(f) : \mathfrak{GR}(\Delta_1) \to \mathfrak{GR}(\Delta_2)$ be the functor defined by precomposition with $\mathfrak{I}(f)$. Concretely

$$\mathfrak{GR}(f)(X) = X \circ \mathfrak{I}(f) \qquad \text{and} \qquad \mathfrak{GR}(f)(\alpha)_{(E,\delta)} = \alpha_{\mathfrak{I}(f)(E,\delta)}$$

where X is an object of $\mathfrak{GR}(\Delta_1)$, α is a natural transformation in $\mathfrak{GR}(\Delta_1)$ and $(E,\delta) \in \mathfrak{I}(\Delta_2)$. We will also use the notation f^* for the functor $\mathfrak{GR}(f)$. ♦

Basic properties of $\mathfrak{GR}$

For each finite set of clocks the category $\mathfrak{GR}(\Delta)$ is a presheaf topos, hence it is a model of extensional dependent type theory. As mentioned above we aim to use the functors $\mathfrak{GR}(f)$ to interpret clock substitution and this means that these functors must preserve constructs used to interpret dependent type theory, in particular dependent products.

Recall first that limits and colimits in presheaf categories are constructed pointwise. Hence because the functors $\mathfrak{GR}(f)$ are given by precomposition they preserve this canonical choice of limits and colimits *on the nose*. Moreover, it is a standard result that each functor $\mathfrak{GR}(f)$ has a left and right adjoint [66, Theorem VII.2.2] which are given by left and right Kan extensions. However only the right adjoint to the functor $\mathfrak{GR}(\iota)$ for an inclusion $\iota : \Delta \to \Delta, \kappa$ seems to be interesting for our application. We will describe it very concretely in Section 5.4.

The fact that each of the functors $\mathfrak{GR}(f)$ also preserve exponentials and local exponentials is more involved and importantly the canonical choice of exponentials and dependent products will not be preserved on the nose, but only up to a canonical isomorphism.

We now show that for each f the functor $\mathfrak{GR}(f)$ is a locally cartesian closed functors.

Definition 5.2.7. Let P and Q be two posets. An order-preserving function $\varphi : P \to Q$ is a *fibration* if for every $p \in P$ and $q \in Q$ such that $q \leq \varphi(p)$ the set

$$B_{p,q} = \{p' \leq p \mid \varphi(p') = q\}$$

has a *top* element $u(p,q)$ and moreover whenever $q_1 \leq q_2$, also $u(p,q_1) \leq u(p,q_2)$. ♦

This definition is equivalent to a standard definition of a fibration [51]. Indeed, the cartesian lifting of the morphism $q \leq \varphi(p)$ is precisely the morphism $u(p,q) \leq p$. One immediate consequence of φ being a fibration is that for each p, the restriction $\varphi_p : \downarrow p \to \downarrow \varphi(p)$ is a retraction with section given by the assignment $q \mapsto u(p,q)$. In particular φ_p is surjective.

One of main reasons *fibrations* are useful is the following property.

Proposition 5.2.8. *Let P and Q be two posets and $\varphi : P \to Q$* a fibration. *The functor $\varphi^* : \mathbf{Set}^{Q^{op}} \to \mathbf{Set}^{P^{op}}$ given by precomposition with φ, i.e. $\varphi^*(X) = X \circ \varphi$, is a* locally cartesian closed functor. ◊

Proof. It is possible to show this directly, but φ being a fibration implies the assumption of Lemma C.3.3.8.(ii) of Johnstone [53] which shows in particular that the functor φ^* is locally cartesian closed by Proposition C.3.3.1 of *loc. cit.* $\mathfrak{QED}$

Next, we show the crucial property. The main result is Lemma 5.2.11. Its proof is split into Lemmas 5.2.9 and 5.2.10 for clarity. This property is the reason for introducing the new indexing poset. It does not hold for the indexing posets of [11, 71].

Lemma 5.2.9. *Let Δ be a finite set of clocks, κ a clock not in Δ and $\iota : \Delta \to \Delta, \kappa$ the inclusion. The functions $\mathfrak{E}(\iota) : \mathfrak{E}(\Delta,\kappa) \to \mathfrak{E}(\Delta)$ and $\mathfrak{I}(\iota) : \mathfrak{I}(\Delta,\kappa) \to \mathfrak{I}(\Delta)$ are both* fibrations. ◊

Proof. We prove the two claims separately. When proving that $\mathfrak{I}(\iota)$ is a fibration we use the construction of $u(E,F)$ from the first part.

$\mathfrak{E}(\iota)$ is a fibration Let $E \in \mathfrak{E}(\Delta,\kappa)$ and $F \in \mathfrak{E}(\Delta)$ such that $F \leq \mathfrak{E}(\iota)(E)$. We consider two cases.

- The simpler case is when κ is only related to itself by E. In this case we define $u(E,F) = F \cup \{(\kappa,\kappa)\}$. Because $\mathfrak{E}(\iota)$ simply restricts the given equivalence relation to the set Δ we immediately have

$\mathfrak{E}(\iota)(u(E,F)) = F$. Further, if $(\kappa_1,\kappa_2) \in E$ then either $\kappa_1 = \kappa_2 = \kappa$ or $(\kappa_1,\kappa_2) \in \mathfrak{E}(\iota)(E)$. In the first case $(\kappa_1,\kappa_2) \in F$ by definition and in the second case this follows from the assumption that $F \leq \mathfrak{E}(\iota)(E)$. Hence we have $u(E,F) \leq E$. Moreover if we have any other $E' \leq E$ such that $\mathfrak{E}(\iota)(E') = F$ we immediately see that it must be less than $u(E,F)$.

- In the second case we $(\kappa,\kappa') \in E$ for some $\kappa' \in \Delta$. We define $u(E,F)$ to be the extension of F to an equivalence relation on Δ,κ by relating κ to κ'. This is the equivalence relation generated by the relation $F \cup \{(\kappa,\kappa')\}$.

  The property $\mathfrak{E}(\iota)(u(E,F)) = F$ follows easily. If $E' \leq E$ is another such that $\mathfrak{E}(\iota)(E') = F$ then because $E' \leq E$ it must be that $(\kappa,\kappa') \in E'$ and so $E' \supseteq F \cup \{(\kappa,\kappa')\}$. Because $u(E,F)$ is defined to be the equivalence relation generated by $F \cup \{(\kappa,\kappa')\}$ we have by definition $E' \supseteq u(E,F)$ which by definition of the order on $\mathfrak{E}(\Delta,\kappa)$ means $E' \leq u(E,F)$.

Finally, to see that the assignment $u(E,-)$ is order-preserving suppose $F_1 \leq F_2$. Observe that the cases above in the construction of $u(E,F_1)$ and $u(E,F_2)$ only depend on the relation E and each case it is straightforward that $u(E,F_1) \leq u(E,F_2)$.

$\mathfrak{I}(\iota)$ **is a fibration** Let $(E,\delta) \in \mathfrak{I}(\Delta,\kappa)$ and $(F,\gamma) \in \mathfrak{I}(\Delta)$ be such that

$$(F,\gamma) \leq \mathfrak{I}(\iota)(E,\delta).$$

We again consider two cases.

- If κ is only related to itself in E then we define

  $$u((E,\delta),(F,\gamma)) = (u(E,F),\delta')$$

  where

  $$\delta'(\kappa') = \begin{cases} \gamma(\kappa') & \text{if } \kappa' \in \Delta \\ \delta(\kappa') & \text{otherwise} \end{cases}$$

  Note that because we require $\delta' \circ \iota = \gamma$ the function δ' is uniquely determined on Δ. Because $u(E,F)$ only relates κ to κ we have that δ' respects $u(E,F)$, hence $u((E,\delta),(F,\gamma))$ is well-defined as an element of $\mathfrak{I}(\Delta,\kappa)$. The fact that $\mathfrak{I}(\iota)$ maps it to (F,γ) is clear and the fact that it is the largest such also follows directly.

- If $(\kappa,\kappa_1) \in E$ for some $\kappa_1 \in \Delta$. We define

  $$u((E,\delta),(F,\gamma)) = (u(E,F),\delta')$$

where

$$\delta'(\kappa') = \begin{cases} \gamma(\kappa') & \text{if } \kappa' \in \Delta \\ \gamma(\kappa_1) & \text{if } \kappa' = \kappa \end{cases}$$

It is immediate from the description of $u(E,F)$ above that δ' respects it. Again, on the set Δ the function δ' is uniquely determined and because by assumption $\gamma \leq \delta \circ \iota$ we have that $\delta' \circ \iota \leq \delta \circ \iota$ so to show $\delta' \leq \delta$ we only need to check that $\delta'(\kappa) \leq \delta(\kappa)$. This holds because $(\kappa, \kappa_1) \in E$ implies $\delta(\kappa) = \delta(\kappa_1)$ which together with the assumption $\gamma \leq \delta \circ \iota$ shows the required relation.

The fact $u((E,\delta),(F,\gamma))$ is the largest such follows directly. In fact in this case $u((E,\delta),(F,\gamma))$ is unique as an element of $B_{(E,\delta),(F,\gamma)}$.

The fact that the assignment $u((E,\delta),-)$ is order-preserving again follows easily because the cases we considered in defining it only depend on E. In each case it is obvious that the assignment is order-preserving.

$\mathfrak{QED}$

Lemma 5.2.10. *Let Δ_1, Δ_2 be two finite sets of clocks and $f : \Delta_1 \to \Delta_2$ a* surjective *function. Then $\mathfrak{E}(f) : \mathfrak{E}(\Delta_2) \to \mathfrak{E}(\Delta_1)$ and $\mathfrak{I}(f) : \mathfrak{I}(\Delta_2) \to \mathfrak{I}(\Delta_1)$ are both* fibrations. ◊

Proof. The construction of $u(E,F)$ and $u((E,\delta),(F,\gamma))$ in this case is more direct.

$\mathfrak{E}(f)$ is a fibration Let $E \in \mathfrak{E}(\Delta_2)$ and $F \in \mathfrak{E}(\Delta_1)$ such that $F \leq \mathfrak{E}(f)(E)$. We define $u(E,F)$ as

$$u(E,F) = \left\{(\kappa, \kappa') \,\middle|\, \exists (\kappa_1, \kappa_2) \in F, f(\kappa_1) = \kappa \wedge f(\kappa_2) = \kappa' \right\}$$

It is not immediately clear that the relation we have defined is an equivalence relation. It is clearly symmetric and because f is surjective it is reflexive. To see that it is transitive suppose $(\kappa, \kappa') \in u(E,F)$ and $(\kappa', \kappa'') \in u(E,F)$. This gives us $(\kappa_1, \kappa_2) \in F$ and $(\kappa_1', \kappa_2') \in F$ such that $f(\kappa_1) = \kappa, f(\kappa_2) = f(\kappa_1') = \kappa'$ and $f(\kappa_2') = \kappa''$. If we have $(\kappa_1, \kappa_2') \in F$ then by definition $(\kappa, \kappa'') \in u(E,F)$. Because E is reflexive we have $(\kappa', \kappa') \in E$ and so $(\kappa_2, \kappa_1') \in \mathfrak{E}(f)(E)$. From the assumption $F \leq \mathfrak{E}(f)(E)$ we thus have $(\kappa_2, \kappa_1') \in F$ and so by transitivity of F we have $(\kappa_1, \kappa_2') \in F$ as needed.

Next, suppose $(\kappa, \kappa') \in E$ and κ_1, κ_2 such that $f(\kappa_1) = \kappa$ and $f(\kappa_2) = \kappa'$. Then $(\kappa_1, \kappa_2) \in \mathfrak{E}(f)(E)$ and from the assumption $F \leq \mathfrak{E}(f)(E)$ we have $(\kappa_1, \kappa_2) \in F$ which further implies $(\kappa, \kappa') \in u(E,F)$. We have thus shown $u(E,F) \leq E$.

To see that $\mathfrak{E}(f)(u(E,F)) = F$ we show two inclusions. The inclusion $F \subseteq \mathfrak{E}(f)(u(E,F))$ is immediate. For the converse inclusion assume $(\kappa,\kappa') \in \mathfrak{E}(f)(u(E,F))$. By definition $(f(\kappa), f(\kappa')) \in u(E,F)$ so there exist $(\kappa_1,\kappa_2) \in F$ such that $f(\kappa') = f(\kappa_1)$ and $f(\kappa') = f(\kappa_2)$. Because E is reflexive we have $(\kappa,\kappa_1) \in \mathfrak{E}(f)(E)$ and $(\kappa',\kappa_2) \in \mathfrak{E}(f)(E)$. From the assumption that $F \leq \mathfrak{E}(f)(E)$ we thus have $(\kappa,\kappa_1) \in F$ and $(\kappa',\kappa_2) \in F$ from which it follows that $(\kappa,\kappa') \in F$ as needed.

The fact that the assignment $u(E,F)$ is monotone in F is clear directly from the definition.

$\mathfrak{I}(f)$ is a fibration Let $(E,\delta) \in \mathfrak{I}(\Delta_2)$ and $(F,\gamma) \in \mathfrak{I}(\Delta_1)$ such that $(F,\gamma) \leq \mathfrak{I}(f)(E,\delta)$. Define $u((E,\delta),(F,\gamma)) = (u(E,F),\delta')$ where we define $\delta'(\kappa) = \gamma(\kappa')$ for some κ' such that $f(\kappa') = \kappa$. In order for this to be a good definition we need to show that it is independent of the chosen κ'. If $f(\kappa') = f(\kappa'')$ then $(\kappa',\kappa'') \in \mathfrak{E}(f)(E)$ because E is reflexive and so $(\kappa',\kappa'') \in F$. Because γ respects F we have $\gamma(\kappa') = \gamma(\kappa'')$, so δ' is well-defined good.

Using the same reasoning it is immediate that $\delta' \circ f = \gamma$ and because f is surjective δ' is unique such. To see that $\delta' \leq \delta$ we have $\delta(\kappa) = \delta(f(\kappa'))$ for some κ', because f is surjective. From the assumption $\gamma \leq \delta \circ f$ we thus get $\gamma(\kappa') \leq \delta(\kappa)$ for any κ and for any κ' such that $f(\kappa') = \kappa$. By definition $\gamma(\kappa') = \delta'(\kappa)$, hence $\delta'(\kappa) \leq \delta(\kappa)$.

Monotonicity of the assignment $u((E,\delta),(F,\gamma))$ is again immediate from the construction. 𝔔𝔈𝔇

Lemma 5.2.11. *Let Δ_1, Δ_2 be two finite sets of clocks and $f : \Delta_1 \to \Delta_2$ a function. Then $\mathfrak{E}(f) : \mathfrak{E}(\Delta_2) \to \mathfrak{E}(\Delta_1)$ and $\mathfrak{I}(f) : \mathfrak{I}(\Delta_2) \to \mathfrak{I}(\Delta_1)$ are both* fibrations. ◊

Proof. Any function $f : \Delta_1 \to \Delta_2$ factors as a surjection $s : \Delta_1 \to f[\Delta_1]$ onto the image $f[\Delta_1]$ of f followed by the inclusion of $f[\Delta_1]$ into Δ_2. In turn the inclusion $f[\Delta_1]$ to Δ_2 factors as a sequence of inclusions ι_i of the form $\Delta \to \Delta, \kappa$. Because $\mathfrak{E}$ and $\mathfrak{I}$ are contravariant functors this implies that for every f, $\mathfrak{E}(f)$ factors as $\mathfrak{E}(s) \circ \mathfrak{E}(\iota_1) \circ \cdots \circ \mathfrak{E}(\iota_n)$ for some n. Analogously for $\mathfrak{I}(f)$. Since composition of fibrations is a fibration Lemmas 5.2.9 and 5.2.10 conclude the proof of this lemma. 𝔔𝔈𝔇

Remark 5.2.12. A careful reader will have noticed that the preceding three lemmas do not rely on the natural numbers $\mathbb{N}$ being the underlying poset. More precisely, if we take any poset P and define $\mathfrak{I}(\Delta)$ to be pairs (E,δ) with $E \in \mathfrak{E}(\Delta)$ and $\delta : \Delta \to P$ with the same order as before then we can also define the action of $\mathfrak{I}$ on morphisms in the same way as before. The preceding lemmas then generalise to show that all morphism $\mathfrak{I}(f)$ are fibrations. ♦

Finally Lemma 5.2.11 and Proposition 5.2.8 combine to prove the following theorem.

Theorem 5.2.13. *Let $f : \Delta_1 \to \Delta_2$ be a function between clock contexts. The functor $\mathfrak{GR}(f)$ is a locally cartesian closed functor.* $\lozenge$

Remark 5.2.14. As we mentioned already the functors $\mathfrak{GR}(f)$ do preserve the natural choice of limits and colimits on the nose. However there does not appear to be a natural choice of exponentials or dependent products such that $\mathfrak{GR}(f)$ would preserve them on the nose. In particular the most problematic is the case when f is not surjective, which is the case that corresponds to clock weakening. The problem already appears with the inclusion of $\mathfrak{GR}(\emptyset)$, which is isomorphic to the category **Set**, into $\mathfrak{GR}(\kappa)$, which is isomorphic to the topos of trees. Given two objects $X, Y \in \mathfrak{GR}(\kappa)$ the canonical exponential $(X \Rightarrow Y)$ at $n \in \mathbb{N}$ consists of n-tuples of functions $f_i : X(i) \to Y(i)$ for $i = 1, 2, \ldots, n$ satisfying naturality conditions. If $X = \mathfrak{GR}(\iota)(X')$ and $Y = \mathfrak{GR}(\iota)(Y')$ then the naturality conditions force all $f_i : X(i) \to Y(i)$ to be equal, so the set of all such n-tuples is certainly isomorphic to the set of functions $X' \to Y'$, but not equal. And $X' \to Y'$ is precisely $\mathfrak{GR}(\iota)(X' \to Y')$ at n. ♦

5.3 The $\blacktriangleright^\kappa$ Functors

Let Δ be a clock context and $\kappa \in \Delta$. We now define the functor $\blacktriangleright^\kappa$ on $\mathfrak{GR}(\Delta)$ and the natural transformation $\mathrm{next}^\kappa : \mathrm{id}_{\mathfrak{GR}(\Delta)} \to \blacktriangleright^\kappa$ such that the triple $(\mathfrak{GR}(\Delta), \blacktriangleright^\kappa, \mathrm{next}^\kappa)$ is a model of guarded recursive terms [22, Definition 6.1].

Example 5.3.1. To understand the definition recall the diagram X with two clocks in Figure 5.1. We wish clock substitution to preserve $\blacktriangleright$ in the sense that $f^*(\blacktriangleright^{\kappa_1} \blacktriangleright^{\kappa_2} X)$ is the same as $\blacktriangleright^{\kappa_1} \blacktriangleright^{\kappa_1} (f^*(X))$ for all $X \in \mathfrak{GR}(\kappa_1, \kappa_2)$ and where $f : \kappa_1, \kappa_2 \to \kappa_2$ is the unique function. Recall that $f^*(X)$ projects out the one dimensional diagram on the right in Figure 5.1. We thus see that the diagram $\blacktriangleright^{\kappa_1} \blacktriangleright^{\kappa_2} X$ should be

$$\begin{tikzcd}
{} \arrow[d] & {} \arrow[d] & {} \arrow[d] & & {} \arrow[d] \\
1 \arrow[d] & X(1,2) \arrow[l] \arrow[d] & X(2,2) \arrow[l] \arrow[d] \arrow[rr, bend right] & \cdots \arrow[l] & X(1) \arrow[d] \\
1 \arrow[d] & X(1,1) \arrow[l] \arrow[d] \arrow[rrr, bend right] & X(2,1) \arrow[l] \arrow[d] & \cdots \arrow[l] & 1 \arrow[d] \\
1 \arrow[rrrr, bend right] & 1 \arrow[l] & 1 \arrow[l] & \cdots \arrow[l] & 1
\end{tikzcd}$$

In particular notice that the one dimensional diagram on the left is delayed twice, because it represents the state when κ_1 and κ_2 are identified. ♦

To define $\blacktriangleright^\kappa$ in general we start with an auxiliary definition.

Definition 5.3.2. Let $\kappa \in \Delta \subseteq^{\mathrm{fin}} \mathrm{CV}$, $E \in \mathfrak{E}(\Delta)$ and $\delta : \Delta \to \mathbb{N}$. The function $\delta^{-\kappa} : \Delta \to \mathbb{N}$ is defined as

$$\delta^{-\kappa}(\kappa') = \begin{cases} \max\{1, \delta(\kappa) - 1\} & \text{if } (\kappa, \kappa') \in E \\ \delta(\kappa') & \text{otherwise} \end{cases} \qquad \blacklozenge$$

The thing to notice in this definition is that all the clocks equivalent to κ have their remaining time decreased by 1. This is crucial for clock substitution to commute with $\blacktriangleright^\kappa$ in the appropriate way, as illustrated in Example 5.3.1 above. Decreasing the value of all the clocks related to κ also ensures that if δ respects E then so does $\delta^{-\kappa}$. This implies $(E, \delta^{-\kappa}) \in \mathfrak{I}(\Delta)$.

Technically $\delta^{-\kappa}$ depends on δ, κ and the equivalence relation E. We leave E implicit though since it can always be inferred from context. We record the basic properties of this construction in the following lemma whose proof is by simple computation.

Lemma 5.3.3.

- *For any* $(E, \delta) \in \mathfrak{E}(\Delta)$ *and any* $\kappa \in \Delta$, $(E, \delta^{-\kappa}) \leq (E, \delta)$.
- *If* $(E_1, \delta_1) \leq (E_2, \delta_2)$ *then* $(E_1, \delta_1^{-\kappa}) \leq (E_2, \delta_2^{-\kappa})$.
- *Let* $f : \Delta_1 \to \Delta_2$ *be a function and* $(E, \delta) \in \mathfrak{I}(\Delta_2)$. *For any* $\kappa \in \Delta_1$ *the pairs*

$$\left(\mathfrak{E}(f)(E), \delta^{-f(\kappa)} \circ f\right) \text{ and } \left(\mathfrak{E}(f)(E), (\delta \circ f)^{-\kappa}\right)$$

are both in $\mathfrak{I}(\Delta_1)$ *and moreover they are equal.* $\lozenge$

The definition of $\blacktriangleright^\kappa : \mathfrak{GR}(\Delta) \to \mathfrak{GR}(\Delta)$ is now simple.

Definition 5.3.4. Let $\kappa \in \Delta \subseteq^{\mathrm{fin}} \mathrm{CV}$ and X an object of $\mathfrak{GR}(\Delta)$. The action of the functor $\blacktriangleright^\kappa$ on objects is

$$\blacktriangleright^\kappa(X)(E, \delta) = \begin{cases} 1 & \text{if } \delta(\kappa) = 1 \\ X(E, \delta^{-\kappa}) & \text{otherwise} \end{cases}$$

$$\blacktriangleright^\kappa(X)((E_1, \delta_1) \leq (E_2, \delta_2)) = \begin{cases} ! & \text{if } \delta_1(\kappa) = 1 \\ X\left((E_1, \delta_1^{-\kappa}) \leq \left(E_2, \delta_2^{-\kappa}\right)\right) & \text{otherwise} \end{cases}$$

where 1 is the singleton set $\{*\}$ and ! is the unique arrow to 1. On morphisms

$$\blacktriangleright^\kappa(\alpha)_{E,\delta} = \begin{cases} \mathrm{id}_1 & \text{if } \delta(\kappa) = 1 \\ \alpha_{E,\delta^{-\kappa}} & \text{otherwise} \end{cases}$$

There is an associated natural transformation $\mathrm{next}^\kappa : \mathrm{id}_{\mathfrak{GR}(\Delta)} \to \blacktriangleright^\kappa$

$$\mathrm{next}^\kappa_{X(E,\delta)}(x) = \begin{cases} * & \text{if } \delta(\kappa) = 1 \\ X((E, \delta^{-\kappa}) \leq (E, \delta))(x) & \text{otherwise} \end{cases} \qquad \blacklozenge$$

It is easy to see that $\blacktriangleright^{\kappa}$ preserves all limits, since these are given pointwise and any limit of any diagram of terminal objects is a terminal object. It does not preserve colimits, however. For example it does not preserve the initial object.

Proposition 5.3.5 (Properties of $\blacktriangleright$)**.** *Let Δ_1 and Δ_2 be two clock contexts and $f : \Delta_1 \to \Delta_2$ a function between them. Let $\kappa \in \Delta_1$ be a clock. The following properties hold.*

1. *Let X, Y be two objects in $\mathfrak{GR}(\Delta_1)$ and $\alpha : Y \times \blacktriangleright^{\kappa}(X) \to X$ a natural transformation. There exists* a unique *$\beta : Y \to X$ such that*

$$\alpha \circ \left\langle id_Y, \mathrm{next}^{\kappa}_X \circ \beta \right\rangle = \beta.$$

We write $\mathrm{fix}^{\kappa}(\alpha)$ for this unique fixed point. Moreover, for any $\gamma : Z \to Y$

$$\mathrm{fix}^{\kappa}(\alpha) \circ \gamma = \mathrm{fix}^{\kappa}\left(\alpha \circ \gamma \times id_Y\right)$$

which expresses naturality of fixed points.

2. *Clock substitution preserves $\blacktriangleright$, i.e.*

$$f^* \circ \blacktriangleright^{\kappa} = \blacktriangleright^{f(\kappa)} \circ f^*,$$

and for every $X \in \mathfrak{GR}(\Delta)$,

$$f^*\left(\mathrm{next}^{\kappa}_X\right) = \mathrm{next}^{f(\kappa)}_{f^*(X)}.$$

3. *Let $\alpha : Y \times \blacktriangleright^{\kappa} X \to X$ be a morphism in $\mathfrak{GR}(\Delta_1)$. From the fact that f^* preserves products on the nose and the previous item the morphism $f^*(\alpha)$ has type $f^*(Y) \times \blacktriangleright^{f(\kappa)} f^*(X) \to f^*(X)$ and moreover*

$$f^*\left(\mathrm{fix}^{\kappa}(\alpha)\right) = \mathrm{fix}^{f(\kappa)}\left(f^*(\alpha)\right). \qquad \lozenge$$

Proof. The fixed point β at (E, δ) is defined by induction on $\delta(\kappa)$ as

$$\beta_{E,\delta}(y) = \begin{cases} \alpha_{E,\delta}(y, *) & \text{if } \delta(\kappa) = 1 \\ \alpha_{E,\delta}\left(y, \beta_{E,\delta^{-\kappa}}\left(Y\left((E, \delta^{-\kappa}) \leq (E, \delta)\right)(y)\right)\right) & \text{otherwise} \end{cases}$$

Naturality and uniqueness follow by an analogous induction on $\delta(\kappa)$.

Item (2) is shown by simple computation using Lemma 5.3.3. For item (3) we have

$$\begin{aligned} f^*\left(\mathrm{fix}^{\kappa}(\alpha)\right) &= f^*\left(\alpha \circ \left\langle \mathrm{id}_Y, \mathrm{next}^{\kappa}_X \circ \mathrm{fix}^{\kappa}(\alpha)\right\rangle\right) \\ &= f^*(\alpha) \circ \left\langle \mathrm{id}_{f^*(Y)}, \mathrm{next}^{f(\kappa)}_{f^*(X)} \circ f^*\left(\mathrm{fix}^{\kappa}(\alpha)\right)\right\rangle. \end{aligned}$$

from Item (2) and the fact that f^* preserves products on the nose. Uniqueness of the fixed point for $f^*(\alpha)$ (item (1)) then shows the desired equality. $\mathfrak{QED}$

The facts above show that for each clock context Δ and $\kappa \in \Delta$, the triple $(\mathfrak{GR}(\Delta), \blacktriangleright^{\kappa}, \mathrm{next}^{\kappa})$ is a model of guarded recursive terms [22, Definition 6.1]. Hence for each object $X \in \mathfrak{GR}(\Delta)$ the slice category $\mathfrak{GR}(\Delta)/X$ also admits a $\blacktriangleright^{\kappa}_{X}$ functor defined by pullback [22, Theorem 6.3]

$$\begin{tikzcd}
\blacktriangleright^{\kappa}_{X} Y \arrow[r] \arrow[d, "\blacktriangleright^{\kappa}_{X} \alpha"'] & \blacktriangleright^{\kappa} Y \arrow[d, "\blacktriangleright^{\kappa} \alpha"] \\
X \arrow[r, "\mathrm{next}^{\kappa}"] & \blacktriangleright^{\kappa} X
\end{tikzcd}$$

This comes with the associated morphism $\mathrm{next}^{\kappa,X}$ in $\mathfrak{GR}(\Delta)/X$. Moreover, for $f : \Delta \to \Delta'$ we easily conclude from Proposition 5.3.5 and the fact that the functor f^* preserves all limits on the nose that

$$f^*\left(\blacktriangleright^{\kappa}_{X} Y\right) = \blacktriangleright^{f(\kappa)}_{f^*(X)} f^*(Y).$$

Note that this is equality of objects in the slice over $f^*(X)$.

The functor $\blacktriangleright^{\kappa}_{X}$ also comes equipped with a natural transformation

$$\mathrm{next}^{\kappa,X} : \mathrm{id} \to \blacktriangleright^{\kappa}_{X}.$$

The component $\mathrm{next}^{\kappa,X}_{Y} : Y \to \blacktriangleright^{\kappa}_{X} Y$ for $\alpha : Y \to X$ in the slice over X is the unique dotted arrow

$$\begin{tikzcd}
Y \arrow[rrd, bend left, "\mathrm{next}^{\kappa}_{Y}"] \arrow[rd, dotted] \arrow[rdd, bend right, "\alpha"'] & & \\
& \blacktriangleright^{\kappa}_{X} Y \arrow[r] \arrow[d, "\blacktriangleright^{\kappa}_{X} \alpha"'] & \blacktriangleright^{\kappa} Y \arrow[d, "\blacktriangleright^{\kappa} \alpha"] \\
& X \arrow[r, "\mathrm{next}^{\kappa}"] & \blacktriangleright^{\kappa} X
\end{tikzcd}$$

which exists by naturality of next^{κ}. Again, because the functors f^* preserve pullbacks on the nose and Proposition 5.3.5, we have

$$f^*\left(\mathrm{next}^{\kappa,X}\right) = \mathrm{next}^{f(\kappa),f^*(X)}$$

so clock substitution behaves well also with respect to $\blacktriangleright^{\kappa}$ and next^{κ} in slices.

5.4 Clock Quantification

For any $\Delta \subseteq^{\mathrm{fin}} \mathrm{CV}$ and clock $\kappa \notin \Delta$ the inclusion function $\iota : \Delta \to \Delta, \kappa$ gives rise to the *weakening* functor $\iota^* : \mathfrak{GR}(\Delta) \to \mathfrak{GR}(\Delta, \kappa)$. Because ι^* is defined by precomposition with $\mathfrak{I}(\iota)$ it has a right (as well as left) adjoint [66, Theorem VII.2.2]. We shall call this right adjoint $\forall\kappa$ and in this section we provide a more explicit description of it, which will provide some more intuition behind it and its relation to coinductive types.

To understand the general definition let us look at two concrete instances. First, the functor $\mathfrak{GR}(\emptyset) \to \mathfrak{GR}(\kappa)$ maps a set X to a diagram which is constantly X and whose restrictions are identities, i.e., a constant presheaf. It is well known that the right adjoint to this functor is the limit functor.

More interesting is the case with two clocks from Figure 5.1. The object $\forall\kappa_2.X$ is a one dimensional diagram and at stage n it is the limit (in **Set**) of the diagram

$$X(n,1) \longleftarrow X(n,2) \longleftarrow X(n,3) \longleftarrow X(n,4) \longleftarrow \cdots$$

The idea is that the type $(\forall\kappa_2.X)(n)$ contains information about $X(n,k)$ for all times k. Note that in particular the one dimensional diagram which represents the state of X when the clocks κ_1 and κ_2 are identified is ignored.

To define the right adjoint of the inclusion in general we need some auxiliaries.

Lemma 5.4.1. *Let Δ be a clock context and $\iota : \Delta \to \Delta, \kappa$ the inclusion. Then $\mathfrak{E}(\iota) : \mathfrak{E}(\Delta,\kappa) \to \mathfrak{E}(\Delta)$ has a* right adjoint *$\iota^!$ defined explicitly as*

$$\iota^!(E) = \{(\kappa_1,\kappa_2) \mid (\kappa_1,\kappa_2) \in E \vee \kappa_1 = \kappa_2 = \kappa\} = E \cup \{(\kappa,\kappa)\} \qquad \Diamond$$

In contrast the function $\mathfrak{I}(\iota)$ does not have a right adjoint, the reason being that $\mathbb{N}$ does not have a top element. However for each $n \in \mathbb{N}$ we can define a function $\iota^!_n$

$$\begin{aligned} \iota^!_n &: \mathfrak{I}(\Delta) \to \mathfrak{I}(\Delta,\kappa) \\ \iota^!_n(E,\delta) &= (\iota^!(E), \delta^!_n) \end{aligned} \qquad \text{where} \qquad \delta^!_n(\kappa') = \begin{cases} \delta(\kappa') & \text{if } \kappa' \in \Delta \\ n & \text{if } \kappa' = \kappa \end{cases}$$

Using the description of $\iota^!$ in Lemma 5.4.1 it is immediate that $\delta^!_n$ respects $\iota^!(E)$. We record some useful properties for use below. The properties follow immediately from the definitions above.

Lemma 5.4.2. *Let Δ be a clock context, $\kappa \notin \Delta$ and $\iota : \Delta \to \Delta, \kappa$ the inclusion.*

1. *If $n \leq m$ and $(E,\delta) \leq (E',\delta')$ then $\iota^!_n(E,\delta) \leq \iota^!_m(E',\delta')$.*
2. *For any $(E,\delta) \in \mathfrak{I}(\Delta,\kappa)$ we have $(E,\delta) \leq \iota^!_{\delta(\kappa)}(\mathfrak{I}(\iota)(E,\delta))$.*
3. *For any $(E,\delta) \in \mathfrak{I}(\Delta)$ and any $n \in \mathbb{N}$ we have $\mathfrak{I}(\iota)\left(\iota^!_n(E,\delta)\right) = (E,\delta)$.*
4. *For any $(E,\delta) \in \mathfrak{I}(\Delta)$ and $\kappa' \in \Delta$, $\delta^{!\,-\kappa'}_n = \left(\delta^{-\kappa'}\right)^!_n$.*

5. *Let $f : \Delta_1 \to \Delta_2$ be a function between two clock contexts, and let $\kappa \notin \Delta_1 \cup \Delta_2$ be a clock. Let and $\iota : \Delta_1 \to \Delta_1, \kappa$ and $\upsilon : \Delta_2 \to \Delta_2, \kappa$ be the two inclusions. Then for any $n \in \mathbb{N}$ and any $(E, \delta) \in \mathbb{I}(\Delta_2)$*

$$\iota^!_n(\mathbb{I}(f)(E,\delta)) = \mathbb{I}(f + id_\kappa)(\upsilon^!_n(E,\delta))$$

where $f + id_\kappa$ is the obvious extension of f to a function $\Delta_1, \kappa \to \Delta_2, \kappa$. ◊

Remark 5.4.3. Analogously to Remark 5.2.12, Lemma 5.4.2 remains true if we replace the poset $\mathbb{N}$ by an arbitrary poset P. ♦

We are now ready to describe the right adjoint $\forall\kappa$ to ι^*. Let Δ be a clock context, κ a clock not in Δ and $\iota : \Delta \to \Delta, \kappa$ the inclusion.

Define $\forall\kappa : \mathfrak{GR}(\Delta, \kappa) \to \mathfrak{GR}(\Delta)$ on an object $X \in \mathfrak{GR}(\Delta, \kappa)$ at stage $(E, \delta) \in \mathbb{I}(\Delta)$ by taking the limit (in **Set**) of the diagram of restrictions

$$X\left(\iota^!_1(E,\delta)\right) \longleftarrow X\left(\iota^!_2(E,\delta)\right) \longleftarrow X\left(\iota^!_3(E,\delta)\right) \longleftarrow \cdots$$

where the arrows are X's restrictions using Lemma 5.4.2. The restrictions of $\forall\kappa.(X)$ and the action of $\forall\kappa$ on morphisms are determined purely formally from the universal properties of limits. The unit η of the adjunction is constructed using the universal property of the limit using Lemma 5.4.2.(3) which shows that the diagram

$$\iota^*(X)\left(\iota^!_1(E,\delta)\right) \longleftarrow \iota^*(X)\left(\iota^!_2(E,\delta)\right) \longleftarrow \iota^*(X)\left(\iota^!_3(E,\delta)\right) \longleftarrow \cdots \tag{5.3}$$

is a constant diagram. The counit ε is constructed with the projections of the limit together with Lemma 5.4.2.(2). In more detail, the component of the counit ε at an object X is a morphism $\varepsilon^X : \iota^*(\forall\kappa(X)) \to X$ in $\mathfrak{GR}(\Delta, \kappa)$ and so at stage (E, δ) we must define a function

$$\varepsilon^X_{(E,\delta)} : \iota^*(\forall\kappa(X))(E,\delta) \to X(E,\delta)$$

which is a function from the limit of

$$X\left(\iota^!_1(\mathbb{I}(\iota)(E,\delta))\right) \longleftarrow X\left(\iota^!_2(\mathbb{I}(\iota)(E,\delta))\right) \longleftarrow \cdots \longleftarrow X\left(\iota^!_{\delta(\kappa)}(\mathbb{I}(\iota)(E,\delta))\right) \longleftarrow \cdots$$

to $X(E,\delta)$. There is a projection from the limit to $X\left(\iota^!_{\delta(\kappa)}(\mathbb{I}(\iota)(E,\delta))\right)$ and from Lemma 5.4.2.(2) we have $(E,\delta) \le \iota^!_{\delta(\kappa)}(\mathbb{I}(\iota)(E,\delta))$ which means there is a function

$$X\left((E,\delta) \le \iota^!_{\delta(\kappa)}(\mathbb{I}(\iota)(E,\delta))\right) : X\left(\iota^!_{\delta(\kappa)}(\mathbb{I}(\iota)(E,\delta))\right) \to X(E,\delta).$$

Since the diagram is in **Set** we could describe the limit very explicitly as the set of compatible sequences. This is useful for checking some properties, but we omit it here due to lack of space.

Equipped with this description of $\forall\kappa$ we are able to show the necessary properties for interpreting the rules of the type theory.

Proposition 5.4.4 (Properties of $\forall\kappa$)**.** *Let Δ be a clock context and $\kappa \in \mathrm{CV}$ a clock not in Δ. The functor $\forall\kappa$ satisfies the following properties.*

1. *The unit η of the adjunction $\iota^* \dashv \forall\kappa$ is a natural isomorphism. Hence ι^* is a full and faithful functor witnessing that $\mathfrak{GR}(\Delta)$ is a full subcategory of $\mathfrak{GR}(\Delta,\kappa)$.*

2. *The functor $\forall\kappa$ preserves all coproducts, but not colimits in general.*

3. *For any object $X \in \mathfrak{GR}(\Delta,\kappa)$ the canonical morphism $c : \forall\kappa.X \to \forall\kappa.(\blacktriangleright^\kappa X)$ defined as $c = \forall\kappa.\left(\mathrm{next}^\kappa_X\right)$ is an isomorphism.*

4. (Beck-Chevalley condition for $\forall\kappa$) *Let $f : \Delta_1 \to \Delta_2$ be a function between two clock contexts, and let $\kappa \notin \Delta_1 \cup \Delta_2$ be a clock.*

   For every $X \in \mathfrak{GR}(\Delta_1,\kappa)$ the presheaves $f^*(\forall\kappa.X)$ and $\forall\kappa.(f+id_\kappa)^*(X)$ are equal *and the canonical morphism*

   $$\forall\kappa.((f+id_\kappa)^*(\varepsilon)) \circ \eta^{f^*(\forall\kappa.X)} \tag{5.4}$$

   from $f^*(\forall\kappa.X)$ to $\forall\kappa.(f+id_\kappa)^*(X)$ is the identity. As a consequence, the functors $\forall\kappa \circ (f+id_\kappa)^*$ and $f^* \circ \forall\kappa$ are equal and further we have

   $$f^*(\eta^X) = \eta^{f^*(X)}. \tag{5.5}$$

5. *Let Δ be a clock context, $\kappa' \in \Delta$, $\kappa \notin \Delta$ and $X \in \mathfrak{GR}(\Delta,\kappa)$ the canonical morphism*

   $$\forall\kappa.(\blacktriangleright^{\kappa'}(\varepsilon)) \circ \eta : \blacktriangleright^{\kappa'}(\forall\kappa.X) \to \forall\kappa.\blacktriangleright^{\kappa'} X$$

   is an isomorphism. ◊

Proof. 1. Using Lemma 5.4.2.(3) the object $\forall\kappa.\iota^*(X)$ at stage (E,δ) is the limit of the constant diagram (5.3). Because the diagram is connected its limit is isomorphic to $X(E,\delta)$ by the unique mediating map, which is by definition the unit η. The second part is a standard fact about adjoint functors [65, Theorem IV.3.1].

2. The reason this property holds is that coproducts are given pointwise and that in **Set** coproducts commute with connected limits.

3. The arrow $\forall\kappa.(\mathrm{next}^\kappa_X)$ at stage $(E,\delta) \in \mathbb{I}(\Delta)$ is by definition the mediating map from the limit of

   $$X\left(\iota^!_1(E,\delta)\right) \longleftarrow X\left(\iota^!_2(E,\delta)\right) \longleftarrow X\left(\iota^!_3(E,\delta)\right) \longleftarrow \cdots$$

   to the limit of

   $$1 \longleftarrow X\left(\iota^!_1(E,\delta)\right) \longleftarrow X\left(\iota^!_2(E,\delta)\right) \longleftarrow X\left(\iota^!_3(E,\delta)\right) \longleftarrow \cdots$$

   so it is an isomorphism.

4. Let $X \in \mathfrak{GR}(\Delta_1, \kappa)$ and $(E, \delta) \in \mathfrak{I}(\Delta_2)$. Let and $\iota : \Delta_1 \to \Delta_1, \kappa$ and $\upsilon : \Delta_2 \to \Delta_2, \kappa$ be the two inclusions. Then

$$f^*(\forall\kappa.X)(E,\delta) = (\forall\kappa.X)(\mathfrak{I}(f)(E,\delta))$$

and by definition $(\forall\kappa.X)(\mathfrak{I}(f)(E,\delta))$ is the limit of the diagram

$$X\left(\iota_1^!(\mathfrak{I}(f)(E,\delta))\right) \longleftarrow X\left(\iota_2^!(\mathfrak{I}(f)(E,\delta))\right) \longleftarrow \cdots$$

By Lemma 5.4.2.(5) this diagram is the same as the diagram

$$X\left(\mathfrak{I}(f+\mathrm{id}_\kappa)(\upsilon_1^!(E,\delta))\right) \longleftarrow X\left(\mathfrak{I}(f+\mathrm{id}_\kappa)(\upsilon_2^!(E,\delta))\right) \longleftarrow \cdots$$

which by definition is the diagram

$$(f+\mathrm{id}_\kappa)^*(X)\left(\upsilon_1^!(E,\delta)\right) \longleftarrow (f+\mathrm{id}_\kappa)^*(X)\left(\upsilon_2^!(E,\delta)\right) \longleftarrow \cdots$$

which is the limit used to define $\forall\kappa.(f+\mathrm{id}_\kappa)^*(X)$ Hence $f^*(\forall\kappa.X)(E,\delta)$ and $\forall\kappa.(f+\mathrm{id}_\kappa)^*(X)$ are limits of the same (not isomorphic, but identical) diagrams, hence they are equal.

The fact that (5.4) is the identity follows by the fact that it is defined using the universal property of limits as follows. Let $(E,\delta) \in \mathfrak{I}(\Delta_2)$. Let π_n^1 be the projection from $\forall\kappa.(f+\mathrm{id}_\kappa)^*(X)(E,\delta)$ to $(f+\mathrm{id}_\kappa)^*(X)(\upsilon_n^!(E,\delta))$ and π_n^2 the projection from $f^*(\forall\kappa.X)(E,\delta)$ to $X\left(\iota_n^!(\mathfrak{I}(f)(E,\delta))\right)$. Note that by the preceding part of the proof these are the same functions, since the limits are over the same diagram. Then by definition of the action of $\forall\kappa$ on morphisms and the definition of η we have for $\overline{x} \in f^*(\forall\kappa.X)(E,\delta)$.

$$\begin{aligned}&\pi_n^1\left((\forall\kappa.((f+\mathrm{id}_\kappa)^*(\varepsilon)))_{(E,\delta)}(\eta_{(E,\delta)}^{f^*(\forall\kappa.X)}(\overline{x}))\right)\\ &=\\ &((f+\mathrm{id}_\kappa)^*(\varepsilon))_{\upsilon_n^!(E,\delta)}\left(\eta_{(E,\delta)}^{f^*(\forall\kappa.X)}\left(\pi_n^2(\overline{x})\right)\right)\end{aligned}$$

By definition of reindexing functors we have

$$((f+\mathrm{id}_\kappa)^*(\varepsilon))_{\upsilon_n^!(E,\delta)} = \varepsilon_{\mathfrak{I}(f+\mathrm{id}_\kappa)(\upsilon_n^!(E,\delta))}.$$

and all projections of

$$\eta_{(E,\delta)}^{f^*(\forall\kappa.X)}\left(\pi_n^2(\overline{x})\right)$$

are the element $\pi_n^2(\overline{x})$ by definition of η. Hence by definition of ε, definition of $\iota_n^!$ and Lemma 5.4.2.(3) we have

$$\begin{aligned}&((f+\mathrm{id}_\kappa)^*(\varepsilon))_{\upsilon_n^!(E,\delta)}\left(\eta_{(E,\delta)}^{f^*(\forall\kappa.X)}\left(\pi_n^2(\overline{x})\right)\right)\\ &=\\ &X\left(\iota_n^!(\mathfrak{I}(f)(E,\delta)) \le \iota_n^!(\mathfrak{I}(f)(E,\delta))\right)(\pi_n^2(\overline{x}))\\ &= \pi_n^2(\overline{x})\end{aligned}$$

Because $\pi_n^2 = \pi_n^1$ the same property holds for the identity function. Thus by uniqueness of mediating morphisms for limits (5.4) must be the identity.

Equality (5.5) is a standard consequence of the Beck-Chevalley condition as follows. Indeed from the fact that (5.4) is the identity on $f^*(\forall\kappa.X)$ on any X we have for any X

$$f^*(\eta^X) = \forall\kappa.((f + \mathrm{id}_\kappa)^*(\varepsilon)) \circ \eta^{f^*(\forall\kappa.\iota^*(X))} \circ f^*(\eta^X)$$

which by naturality of $\eta^{f^*(\forall\kappa.\iota^*(X))}$ and functoriality of $\forall\kappa \circ (f + \mathrm{id}_\kappa)^*$ is

$$= \forall\kappa.((f + \mathrm{id}_\kappa)^*(\varepsilon \circ \iota^*(\eta^X))) \circ \eta^{f^*(X)}$$

and by one of the triangle identities $\varepsilon \circ \iota^*(\eta^X) = \mathrm{id}$, hence

$$= \eta^{f^*(X)}$$

as claimed.

5. Follows by computation and Lemma 5.4.2.(4). Note that to even state it Proposition 5.3.5 is used to get $\iota^* \circ \blacktriangleright^{\kappa'} = \blacktriangleright^{\kappa'} \circ \iota^*$ so we could apply the counit ε. $\mathfrak{QED}$

Extension of clock quantification to slice categories

Let $\Delta_1, \Delta_2 \subseteq^{\mathrm{fin}} \mathrm{CV}$ be two clock contexts and $f : \Delta_1 \to \Delta_2$ a function. For any $X \in \mathfrak{GR}(\Delta_1)$ the functor $f^* : \mathfrak{GR}(\Delta_1) \to \mathfrak{GR}(\Delta_2)$ induces a functor $f_X^* : \mathfrak{GR}(\Delta_1)/X \to \mathfrak{GR}(\Delta_2)/f^*(X)$ on the slice categories in the usual way.

Specializing, let Δ be a finite set of clocks, $\kappa \notin \Delta$ and $\iota : \Delta \to \Delta, \kappa$ the inclusion. The functor $\iota^* : \mathfrak{GR}(\Delta) \to \mathfrak{GR}(\Delta, \kappa)$ induces for each $X \in \mathfrak{GR}(\Delta)$ a functor

$$\iota_X^* : \mathfrak{GR}(\Delta)/X \to \mathfrak{GR}(\Delta, \kappa)/\iota^*(X)$$

between slice categories. Similarly the functor $\forall\kappa : \mathfrak{GR}(\Delta, \kappa) \to \mathfrak{GR}(\Delta)$ induces a functor

$$\mathfrak{GR}(\Delta, \kappa)/\iota^*(X) \to \mathfrak{GR}(\Delta)/(\forall\kappa.\iota^*(X))$$

and because $\forall\kappa \circ \iota^* \cong \mathrm{id}_{\mathfrak{GR}(\Delta)}$ we get an induced functor

$$\forall_X\kappa : \mathfrak{GR}(\Delta, \kappa)/\iota^*(X) \to \mathfrak{GR}(\Delta)/X.$$

Concretely, given an object $\alpha : Y \to \iota^*(X)$ of $\mathfrak{GR}(\Delta, \kappa)/\iota^*(X)$ we have

$$\forall_X\kappa.(\alpha) = \left(\eta^X\right)^{-1} \circ \forall\kappa.(\alpha)$$

and its action on morphisms is the action of $\forall\kappa$ on morphisms. Standard abstract nonsense then shows that this functor is right adjoint to the functor ι_X^*. The units and counits of the adjunction are the unit and the counit of the adjunction $\iota^* \dashv \forall\kappa$.

Proposition 5.4.5 (Properties of $\forall\kappa$ on slices). *Let Δ be a clock context and $\kappa \in \mathrm{CV}$ a clock not in Δ. Let $X \in \mathfrak{GR}(\Delta)$. The functor $\forall_X\kappa$ satisfies the following properties.*

1. *The unit of the adjunction $\iota^*_X \dashv \forall_X\kappa$ is a natural isomorphism. Hence ι^*_X is a full and faithful functor witnessing that $\mathfrak{GR}(\Delta)/X$ is a full subcategory of $\mathfrak{GR}(\Delta,\kappa)/\iota^*(X)$.*

2. *The functor $\forall\kappa_X$ preserves coproducts.*

3. *For any object $Y \in \mathfrak{GR}(\Delta,\kappa)/\iota^*(X)$ the canonical morphism*
$$c : \forall_X\kappa.Y \to \forall_X\kappa(\blacktriangleright^{\kappa}_{Y} X)$$
defined as $c = \forall_X\kappa.\left(\mathrm{next}^{\kappa,X}_{Y}\right)$ is an isomorphism.

4. (Beck-Chevalley condition for $\forall_X\kappa$) *Let Δ_2 be another clock context such that $\kappa \notin \Delta_2$. Let $f : \Delta \to \Delta_2$ be a function between the two clock contexts. For every $Y \in \mathfrak{GR}(\Delta,\kappa)/\iota^*(X)$ the objects*
$$f^*_X(\forall_X\kappa.Y) \qquad \textit{and} \qquad \forall_{f^*(X)}\kappa.(f+id_\kappa)^*_{\iota^*(X)}(Y)$$
are equal *and the canonical morphism*
$$\forall_{f^*(X)}\kappa.((f+id_\kappa)^*_{\iota^*(X)}(\varepsilon)) \circ \eta^{f^*_X(\forall_X\kappa.Y)} \tag{5.6}$$
from $f^*_X(\forall\kappa.Y)$ to $\forall_{f^*(X)}\kappa.(f+id_\kappa)^*_{\iota^*(X)}(Y)$ is the identity.

5. *Let $\kappa' \in \Delta$. Let $Y \in \mathfrak{GR}(\Delta,\kappa)/\iota^*(X)$. The canonical morphism*
$$\forall_X\kappa.(\blacktriangleright^{\kappa'}_X(\varepsilon)) \circ \eta : \blacktriangleright^{\kappa'}_X(\forall_X\kappa.Y) \to \forall_X\kappa.\blacktriangleright^{\kappa'}_X Y$$
is an isomorphism. ◊

Proof. These properties follow from corresponding properties for the functor $\forall\kappa$ stated and proved in Proposition 5.4.4. We exemplify this by proving item 3. Recall that given an object $\alpha : Y \to \iota^*(X)$ in $\mathfrak{GR}(\Delta,\kappa)/\iota^*(X)$, the morphism $\mathrm{next}^{X,\kappa}\, Y$ is the unique dotted mediating morphism in the diagram

$$\begin{tikzcd}
Y \arrow[drr, bend left, "\mathrm{next}^{\kappa}_{Y}"] \arrow[dr, dotted, "\mathrm{next}^{\kappa,X}_{Y}"] \arrow[ddr, bend right, "\alpha"'] & & \\
& \blacktriangleright^{\iota^*(X)}_{\kappa} Y \arrow[r, "i"] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \blacktriangleright^{\kappa} Y \arrow[d, "\blacktriangleright^{\kappa}\alpha"] \\
& \iota^*(X) \arrow[r, "\mathrm{next}^{\kappa}_{\iota^*(X)}"] & \blacktriangleright^{\kappa}\iota^*(X)
\end{tikzcd}$$

Applying the functor $\forall\kappa$, which is a right adjoint, hence preserves pullbacks, to the diagram we have the diagram

$$\begin{tikzcd}
\forall\kappa.Y \arrow[rrd, bend left, "\forall\kappa.\mathrm{next}^{\kappa}_{Y}"] \arrow[rd, dotted, "\forall\kappa.\mathrm{next}^{\kappa,X}_{Y}"] \arrow[rdd, bend right, "\forall\kappa.\alpha"'] & & \\
& \forall\kappa.\blacktriangleright^{\iota^*(X)}_{\kappa} Y \arrow[r, "\forall\kappa.i"] \arrow[d] \arrow[rd, phantom, "\lrcorner", very near start] & \blacktriangleright^{\kappa}\forall\kappa.Y \arrow[d, "\forall\kappa.\blacktriangleright^{\kappa}\alpha"] \\
& \forall\kappa.\iota^*(X) \arrow[r, "\forall\kappa.\mathrm{next}^{\kappa}_{\iota^*(X)}"] & \blacktriangleright^{\kappa}\forall\kappa.\iota^*(X)
\end{tikzcd}$$

Recall that by definition $\forall_X\kappa.\mathrm{next}^{\kappa,X}_{Y}$ is just $\forall\kappa.\mathrm{next}^{\kappa,X}_{Y}$ so we wish to show that this morphism is an isomorphism. By Proposition 5.4.4 the morphisms $\forall\kappa.\mathrm{next}^{\kappa}_{\iota^*(X)}$ and $\forall\kappa.\mathrm{next}^{\kappa}_{Y}$ are isomorphisms. Thus because a pullback of an isomorphism is an isomorphism, the morphism $\forall\kappa.i$ is an isomorphism. Hence so is $\forall\kappa.\mathrm{next}^{\kappa,X}_{Y}$, since from the commutativity of the diagram we have

$$\forall\kappa.\mathrm{next}^{\kappa,X}_{Y} = (\forall\kappa.i)^{-1}\circ\forall\kappa.\mathrm{next}^{\kappa}_{Y}. \qquad \mathfrak{QED}$$

5.5 Universes

We follow previous work [18, 71] and use Hofmann and Streicher's construction of universes in presheaf toposes from Grothendieck universes [49] which we now recall instantiated to our special case. We first recall what a Grothendieck universe is and what a universe in a locally cartesian closed category is.

Definition 5.5.1. A *Grothendieck universe* is a set $\mathbb{U}$ satisfying the following properties.

- $\mathbb{U}$ is a transitive set: if $x \in A$ and $A \in \mathbb{U}$ then $x \in \mathbb{U}$.
- For all $A \in \mathbb{U}$ the power set $\mathcal{P}(A)$ is in $\mathbb{U}$.
- $\mathbb{N} \in \mathbb{U}$[1]
- If $I \in \mathbb{U}$ and $A : I \to \mathbb{U}$ is an I-indexed collection of elements of $\mathbb{U}$ then $\bigcup_{i\in I} A(i) \in \mathbb{U}$.

[1] Often the assumption that $\mathbb{N} \in \mathbb{U}$ is weakened to $\emptyset \in \mathbb{U}$ or even omitted entirely, in which case the empty set is an example of a Grothendieck universe. Since we will need $\mathbb{N} \in \mathbb{U}$ in all our universes we assume all our universes are such.

♦

Using the usual encodings of operations in set theory it is possible to show that $\mathbb{U}$ is closed under small products, small disjoint unions and function spaces. By this we mean if $I \in \mathbb{U}$ and $A : I \to \mathbb{U}$ is an I-indexed collection of elements of $\mathbb{U}$ then

$$\prod_{i \in I} A(i) \in \mathbb{U} \qquad \text{and} \qquad \sum_{i \in I} A(i) \in \mathbb{U}$$

where $\prod$ is the product and $\sum$ is the disjoint union. By being closed under function spaces we mean that if $A, B \in \mathbb{U}$ then the set of functions from A to B is in $\mathbb{U}$.

The existence of such a Grothendieck universe is equivalent to the existence of a strongly inaccessible cardinal [94], so outside of pure ZFC set theory. Existence of a strongly inaccessible cardinal is however a common assumption underlying many results in set theory.

Definition 5.5.2. Let $\mathbb{C}$ be a locally cartesian closed category with coproducts and a natural numbers object $\mathbb{N}$. Let $\mathrm{el} : E \to U$ be a morphism in $\mathbb{C}$. A morphism $f : A \to \Gamma$ is *small* with respect to el, or el-*small*, if there *exists* a morphism $\bar{\mathsf{f}} : \Gamma \to U$ such that f appears as the pullback of el along $\bar{\mathsf{f}}$ as in the following diagram.

$$\begin{tikzcd} A \arrow[r, "\underline{\mathsf{f}}"] \arrow[d, "f"'] \arrow[dr, phantom, "\lrcorner", very near start] & E \arrow[d, "\mathrm{el}"] \\ \Gamma \arrow[r, "\bar{\mathsf{f}}"] & U \end{tikzcd} \tag{5.7}$$

The morphism $\bar{\mathsf{f}}$ is called a *code* of f. An object Γ is small if the unique map $\Gamma \to 1$ is small.

The morphism el is a *universe* if the objects 0, 1, $\mathbb{N}$ are small and the notion of smallness is closed under composition, finite coproducts and small dependent products. ♦

By the "notion of smallness is closed under small dependent products" we mean that if $\alpha : A \to \Gamma$ and $\beta : B \to A$ are el-small then $\Pi(\alpha, \beta) \in \mathbb{C}/\Gamma$ is el-small. Here $\Pi(\alpha, -)$ is the right adjoint to pullback along α.

In general a small map $f : A \to \Gamma$ can have many different codes and for a given code $\bar{\mathsf{f}}$ there can be many morphism $\underline{\mathsf{f}}$ making the diagram (5.7) a pullback. Indeed, if $\underline{\mathsf{f}}$ is such a morphism and $\alpha : A \to A$ is an automorphism satisfying $f \circ \alpha = f$ then replacing $\underline{\mathsf{f}}$ with $\underline{\mathsf{f}} \circ \alpha$ yields another pullback and in general $\underline{\mathsf{f}} \circ \alpha$ will be different from $\underline{\mathsf{f}}$.

Remark 5.5.3. The map el is always small, since it is a pullback of el along the identity. ♦

Example 5.5.4. Let $\mathbb{U}$ be a Grothendieck universe. Let $\mathbb{E}$ be the disjoint union $\sum_{A \in \mathbb{E}} A$ and $\mathrm{el} : \mathbb{E} \to \mathbb{U}$ the first projection. Then el is a universe in **Set**.

An object $A \in$ **Set** is small precisely when it is isomorphic (in **Set**) to an element of $\mathbb{U}$. A morphism $f : A \to B$ is small precisely when for all $b \in B$, the fibre $f^{-1}(b)$ is isomorphic to a set in $\mathbb{U}$. ♦

We now recall the construction of universes in presheaf toposes specialised to our presheaf categories.

Definition 5.5.5 (Hofmann-Streicher [49]). Let $\mathbb{U}$ be a Grothendieck universe and $\Delta \subseteq^{\text{fin}} \mathrm{CV}$. Let the presheaf $V^{\Delta} \in \mathfrak{GR}(\Delta)$ be defined at $(E, \delta) \in \mathrm{I}(\Delta)$ as

$$V^{\Delta}(E, \delta) = \mathbb{U}^{\downarrow(E,\delta)^{\mathrm{op}}}$$

where $\downarrow(E, \delta)$ is the poset on elements

$$\downarrow(E, \delta) = \{(E', \delta') \in \mathrm{I}(\Delta) \mid (E', \delta') \leq (E, \delta)\}$$

with order inherited from $\mathrm{I}(\Delta)$. The set $\mathbb{U}^{\downarrow(E,\delta)^{\mathrm{op}}}$ is the set of presheaves D on $\downarrow(E, \delta)$ such that for all $(E', \delta') \in \downarrow(E, \delta)$ we have $D(E', \delta') \in \mathbb{U}$.

The action of V^{Δ} on morphisms is by precomposition:

$$V^{\Delta}((E_1, \delta_1) \leq (E_2, \delta_2))(D) = D \circ \iota$$

where ι is the inclusion of $\downarrow(E_1, \delta_1)$ to $\downarrow(E_2, \delta_2)$.

The presheaf of elements E^V_{Δ} is defined as

$$E^V_{\Delta}(E, \delta) = \sum_{D \in V^{\Delta}(E,\delta)} D(E, \delta)$$

with restrictions

$$E^V_{\Delta}((E_1, \delta_1) \leq (E_2, \delta_2))(D, x) = (D \circ \iota, D((E_1, \delta_1) \leq (E_2, \delta_2))(x)).$$

The Hofmann-Streicher universe is the first projection $u^{\Delta} : E^V_{\Delta} \to V^{\Delta}$ defined as

$$u^{\Delta}_{E,\delta}(D, x) = D.$$ ♦

Hofmann and Streicher [49] show that u^{Δ} is a universe, so in particular it is closed under dependent products and dependent sums (equivalently composition).

The following proposition will be useful. It implies in particular that all monomorphisms are small.

Proposition 5.5.6 (Hofmann-Streicher [49]). *A morphism $f : A \to \Gamma$ is u^{Δ}-small if and only if for all $(E, \delta) \in \mathrm{I}(\Delta)$ the component $f_{E,\delta} : A(E, \delta) \to \Gamma(E, \delta)$ is small with respect to the universe* el *constructed in Example 5.5.4.* ◊

The universe is closed under $\blacktriangleright^\kappa$

Lemma 5.5.7. *Let $\kappa \in \Delta \subseteq^{\mathrm{fin}} \mathrm{CV}$. There is a morphism $\widehat{\triangleright}^\kappa : \blacktriangleright^\kappa V^\Delta \to V^\Delta$ making the following diagram a pullback. In other words, $\blacktriangleright^\kappa u^\Delta$ is u^Δ-small.*

$$\begin{tikzcd}
\blacktriangleright^\kappa E^V_\Delta \arrow[r] \arrow[d, "\blacktriangleright^\kappa u^\Delta"'] \arrow[dr, phantom, "\lrcorner", very near start] & E^V_\Delta \arrow[d, "u^\Delta"] \\
\blacktriangleright^\kappa V^\Delta \arrow[r, "\widehat{\triangleright}^\kappa"] & V^\Delta
\end{tikzcd}$$

◊

Proof. Using Proposition 5.5.6 it suffices to show that all components of the natural transformation $\blacktriangleright^\kappa u^\Delta$ are small. They are small because they are either components of u^Δ or the identity on the singleton set. $\mathfrak{QED}$

Proposition 5.5.8. *Let $\kappa \in \Delta \subseteq^{\mathrm{fin}} \mathrm{CV}$. If $\alpha : A \to \Gamma$ is u^Δ-small then so is $\blacktriangleright^\kappa_\Gamma \alpha : \blacktriangleright^\kappa_\Gamma A \to \Gamma$ and if $\overline{\alpha}$ is a code for α then*

$$\widehat{\triangleright}^\kappa \circ \mathrm{next}^\kappa_{V^\Delta} \circ \overline{\alpha}$$

is a code for $\blacktriangleright^\kappa_\Gamma \alpha$. ◊

Proof. Let $\overline{\alpha}$ be a code for α. Recall that $\blacktriangleright^\kappa$ preserves pullbacks and consider the following diagram

$$\begin{tikzcd}
\blacktriangleright^\kappa_\Gamma A \arrow[r] \arrow[d, "\blacktriangleright^\kappa_\Gamma \alpha"] \arrow[dr, phantom, "\lrcorner", very near start] & \blacktriangleright^\kappa A \arrow[r, "\blacktriangleright^\kappa \underline{\alpha}"] \arrow[d, "\blacktriangleright^\kappa \alpha"] \arrow[dr, phantom, "\lrcorner", very near start] & \blacktriangleright^\kappa E^V_\Delta \arrow[r] \arrow[d, "\blacktriangleright^\kappa u^\Delta"'] \arrow[dr, phantom, "\lrcorner", very near start] & E^V_\Delta \arrow[d, "u^\Delta"] \\
\Gamma \arrow[r, "\mathrm{next}^\kappa_\Gamma"] & \blacktriangleright^\kappa \Gamma \arrow[r, "\blacktriangleright^\kappa \overline{\alpha}"] & \blacktriangleright^\kappa V^\Delta \arrow[r, "\widehat{\triangleright}^\kappa"] & V^\Delta
\end{tikzcd}$$

where the left square is a pullback by definition of $\blacktriangleright^\kappa_\Gamma$, the middle square is a pullback because $\blacktriangleright^\kappa$ preserves pullbacks and the right square is a pullback by Lemma 5.5.7. Hence by the pullback pasting lemma the outer rectangle is a pullback, and so by definition $\widehat{\triangleright}^\kappa \circ \blacktriangleright^\kappa \overline{\alpha} \circ \mathrm{next}^\kappa_\Gamma$ is a code for $\blacktriangleright^\kappa_\Gamma \alpha$. By naturality of next^κ we have the equality

$$\widehat{\triangleright}^\kappa \circ \blacktriangleright^\kappa \overline{\alpha} \circ \mathrm{next}^\kappa_\Gamma = \widehat{\triangleright}^\kappa \circ \mathrm{next}^\kappa_{V^\Delta} \circ \overline{\alpha}$$

concluding the proof. $\mathfrak{QED}$

The universes are closed under $\forall\kappa$

The last new operation we have is $\forall\kappa$ so we need to show that this also maps small morphisms to small morphisms. However in contrast to $\blacktriangleright^\kappa$, the functor $\forall\kappa$ changes the clock context and so correspondingly we need to relate small maps in two different universes.

We start with a lemma about the universe el from Example 5.5.4.

Lemma 5.5.9. *Let P be a poset whose underlying set is in $\mathbb{U}$. Let $D_1, D_2 : P^{op} \to$ **Set** be two diagrams and $\alpha : D_1 \to D_2$ a natural transformation. If α_p is* el-*small for all $p \in P$ then*

$$\lim_{p \in P} \alpha_p : \lim_{p \in P} D_1(p) \to \lim_{p \in P} D_2(p)$$

is el-*small.* ◊

Proof. Recall that the limit of a diagram in **Set** can be constructed as a set of compatible families. Concretely $\lim_{p \in P} D_1(p)$ is the set

$$\left\{ a \in \prod_{p \in P} D_1(p) \;\middle|\; \forall p \in P, q \leq p, D_1(q \leq p)(a_p) = a_q \right\}$$

and analogously $\lim_{p \in P} D_2(p)$. Let us write β for the function $\lim_{p \in P} \alpha_p$. Concretely it is defined as

$$\beta(a) = p \mapsto \alpha_p(a(p)).$$

We need to show that for each $a \in \lim_{p \in P} D_2(p)$, the fibre $\beta^{-1}(a)$ is isomorphic to an element of $\mathbb{U}$. By definition of β the fibre $\beta^{-1}(a)$ is a subset of the product

$$\prod_{p \in P} \alpha_p^{-1}(a(p)).$$

Because for all $p \in P$ the function α_p is el-small each $\alpha_p^{-1}(a(p))$ is isomorphic to some $A_p \in \mathbb{U}$. Hence $\prod_{p \in P} \alpha_p^{-1}(a(p))$ is isomorphic to $\prod_{p \in P} A_p$ and $\beta^{-1}(a)$ is isomorphic to a subset of $\prod_{p \in P} A_p$. Because $P \in \mathbb{U}$ the product $\prod_{p \in P} A_p$ is also in $\mathbb{U}$. Because $\mathbb{U}$ is transitive and closed under power sets all of the subsets of $\prod_{p \in P} A_p$ are in $\mathbb{U}$, hence in particular $\beta^{-1}(a)$ is isomorphic to an element of $\mathbb{U}$ as needed. 𝔔𝔈𝔇

Lemma 5.5.10. *Let $\Delta \subseteq^{\text{fin}} \mathrm{CV}$ and κ a clock not in Δ. Let $A, \Gamma \in \mathfrak{GR}(\Delta, \kappa)$ and $\alpha : A \to \Gamma$ a morphism. If α is small with respect to $u^{\Delta,\kappa}$ then the morphism $\forall\kappa.\alpha \in \mathfrak{GR}(\Delta)$ is small with respect to u^{Δ}.*

In particular there exists a morphism $\overline{\forall} : \forall\kappa.V^{\Delta,\kappa} \to V^{\Delta}$ making the following diagram a pullback.

$$\begin{tikzcd}
\forall\kappa.E^{V}_{\Delta,\kappa} \arrow[r] \arrow[d, "\forall\kappa.u^{\Delta,\kappa}"'] \arrow[dr, phantom, "\lrcorner", very near start] & E^{V}_{\Delta} \arrow[d, "u^{\Delta}"] \\
\forall\kappa.V^{\Delta,\kappa} \arrow[r, "\overline{\forall}"] & V^{\Delta}
\end{tikzcd}$$

◊

Proof. Recall that for any $X \in \mathfrak{GR}(\Delta,\kappa)$ the diagram $\forall\kappa.X$ at $(E,\delta) \in \mathbb{I}(\Delta)$ is defined to be the limit of the $\mathbb{N}^{op}$ indexed diagram

$$D_X(n) = X(\iota_n^!(E,\delta)).$$

Analogously given a morphism $f : X \to Y$ we have for each $(E,\delta) \in \mathbb{I}(\Delta)$ a natural transformation $D^f : D_X \to D_Y$ defined as

$$D_n^f = f_{\iota_n^!(E,\delta)}$$

and by definition $\forall\kappa.f$ is the morphism $\lim_{n\in\mathbb{N}} D_n^f$.

Hence, if $\alpha : A \to \Gamma$ is small then by Proposition 5.5.6 all of its components are small so in particular all of the components of the natural transformation D^α are small. By Lemma 5.5.9 we have $(\forall\kappa.\alpha)_{E,\delta}$ small for each (E,δ), and so again by Proposition 5.5.6 (but this time for the universe u^Δ in $\mathfrak{GR}(\Delta)$) the morphism $\forall\kappa.\alpha$ is small.

The last claim in the lemma now follows immediately from the fact that $u^{\Delta,\kappa}$ is $u^{\Delta,\kappa}$-small. $\mathfrak{QED}$

Proposition 5.5.11. *Let $\Delta \subseteq^{\text{fin}} \mathrm{CV}$, κ a clock not in Δ and $\iota : \Delta \to \Delta,\kappa$ the inclusion. Let $\Gamma \in \mathfrak{GR}(\Delta)$, $A \in \mathfrak{GR}(\Delta,\kappa)$ and $\alpha : A \to \iota^*(\Gamma)$ a morphism, or, equivalently, an object of $\mathfrak{GR}(\Delta,\kappa)/\iota^*(\Gamma)$.*

If α is small with respect to $u^{\Delta,\kappa}$ then $\forall_\Gamma\kappa.\alpha \in \mathfrak{GR}(\Delta)/\Gamma$ is small with respect to u^Δ. Moreover, if $\overline{\alpha}$ is a code for α then $\overline{\forall} \circ (\forall\kappa.\overline{\alpha}) \circ \eta^\Gamma$ is a code for $\forall_\Gamma.\alpha$. $\lozenge$

Proof. Recall from Section 5.4 (page 198) that $\forall_\Gamma\kappa.\alpha$ is defined to be $\left(\eta^\Gamma\right)^{-1} \circ \forall\kappa.\alpha$ where η is the unit of the adjunction $\iota^* \dashv \forall\kappa$.

Consider the following diagram

$$\begin{tikzcd}
\forall\kappa.A \arrow[r, "\mathrm{id}"] \arrow[d, "\forall_\Gamma\kappa.\alpha"'] \arrow[dr, phantom, "\lrcorner", very near start] & \forall\kappa.A \arrow[r] \arrow[d, "\forall\kappa.\alpha"'] \arrow[dr, phantom, "\lrcorner", very near start] & \forall\kappa.E^V_{\Delta,\kappa} \arrow[r] \arrow[d, "\forall\kappa.u^{\Delta,\kappa}"'] \arrow[dr, phantom, "\lrcorner", very near start] & E^V_\Delta \arrow[d, "u^\Delta"] \\
\Gamma \arrow[r, "\eta^\Gamma"] & \forall\kappa.\iota^*(\Gamma) \arrow[r, "\forall\kappa.\overline{\alpha}"] & \forall\kappa.V^{\Delta,\kappa} \arrow[r, "\overline{\forall}"] & V^\Delta
\end{tikzcd}$$

The right square is a pullback by Lemma 5.5.10 and the middle square is a pullback by the assumption that $\overline{\alpha}$ is a code for α and because $\forall\kappa$ is a right adjoint, so it preserves pullbacks. The left square is easily seen to be a pullback, hence the outer rectangle is a pullback by the pullback pasting lemma. $\mathfrak{QED}$

Preservation of universes by clock substitution

The functors $\mathfrak{GR}(f)$ do not in general preserve the universes we have defined. More precisely by this we mean that it is not true in general that given

$f : \Delta_1 \to \Delta_2$ being $f^*\left(u^{\Delta_1}\right)$-small is equivalent to being u^{Δ_2}-small. Let us see a concrete example. Recall that $\mathfrak{GR}(\emptyset)$ is isomorphic to the category **Set** and $\mathfrak{GR}(\kappa)$ is isomorphic to the topos of trees. The universe $u^{\emptyset}$ is therefore (modulo the isomorphism between **Set** and $\mathfrak{GR}(\emptyset)$) the universe el from Example 5.5.4. Let $\iota : \emptyset \to \{\kappa\}$ be the inclusion and consider when an object $X \in \mathfrak{GR}(\kappa)$ is small with respect to $\iota^*(u^{\emptyset})$. Recall that an object is small by definition if the unique map $X \to 1$ fits into the pullback

$$\begin{tikzcd}
X \arrow[r] \arrow[d] & \iota^*\left(E^V_{\emptyset}\right) \arrow[d, "\iota^*(u^{\emptyset})"] \\
1 \arrow[r, "\overline{\mathrm{X}}"] & \iota^*\left(V^{\emptyset}\right)
\end{tikzcd}$$

for some $\overline{\mathrm{X}}$. If this is the case, it is easy to see that X must be isomorphic to $\iota^*(Y)$ for some $Y \in \mathfrak{GR}(\emptyset)$.

On the other hand, Proposition 5.5.6 tells us that an object X of $\mathfrak{GR}(\kappa)$ is $u^{\{\kappa\}}$-small precisely when $X(n)$ is isomorphic to an element of $\mathbb{U}$ for all $n \in \mathbb{N}$. Since not all of these objects are of the form $\iota^*(Y)$ the notion of smallness for $\iota^*\left(u^{\emptyset}\right)$ and $u^{\{\kappa\}}$ are different.

This example contains the essence of the problem. If $\Delta_1, \Delta_2 \subseteq^{\text{fin}} \mathrm{CV}$ are disjoint, Δ_2 is non-empty and $\iota : \Delta_1 \to \Delta_1 \cup \Delta_2$ is the inclusion then an object $X \in \mathfrak{GR}(\Delta_1 \cup \Delta_2)$ is $\iota^*\left(u^{\Delta_1}\right)$-small precisely when it is isomorphic to $\iota^*(Y)$ for some $Y \in \mathfrak{GR}(\Delta_1)$, so inclusions do not preserve universes.

In contrast *surjective* clock subsitutions preserve universes as stated in the following lemma.

Lemma 5.5.12. *Let $s : \Delta \to \Delta'$ be a* surjective *function between clock contexts Δ and Δ'. There exist* isomorphisms *$c^V_s : s^*\left(V^{\Delta}\right) \to V^{\Delta'}$ and $c^E_s : s^*\left(E^V_{\Delta}\right) \to E^V_{\Delta'}$ such that the diagram*

$$\begin{tikzcd}
s^*\left(E^V_{\Delta}\right) \arrow[r, "c^E_s"] \arrow[d, "s^*(u^{\Delta})"'] & E^V_{\Delta'} \arrow[d, "u^{\Delta'}"] \\
s^*\left(V^{\Delta}\right) \arrow[r, "c^V_s"] & V^{\Delta'}
\end{tikzcd}$$

commutes. In particular the diagram is a pullback.

Moreover, if $t : \Delta'' \to \Delta$ is also surjective then

$$c^V_{s \circ t} = c_s \circ s^*(c^V_t) \tag{5.8}$$

and

$$c^E_{s \circ t} = c_s \circ s^*(c^E_t). \tag{5.9}$$

◊

Proof. From Lemma 5.2.4 we have $\mathbb{I}(s)$ *injective* and from Lemma 5.2.11 we have that $\mathbb{I}(s)$ is a fibration. Thus $\mathbb{I}(s)$ restricted to a function $\downarrow(E,\delta) \to \downarrow\mathbb{I}(s)(E,\delta)$ is a bijection with an order preserving inverse given by the assignment $u((E,\delta),-)$.

Moreover, because the bijection is given by a restriction of a single function $\mathbb{I}(s)$ it is natural in (E,δ). We thus have

$$s^*(V^{\Delta})(E,\delta) = V^{\Delta}(\mathbb{I}(s)(E,\delta)) = \mathbb{U}^{\downarrow\mathbb{I}(s)(E,\delta)^{\mathrm{op}}} \cong \mathbb{U}^{\downarrow(E,\delta)^{\mathrm{op}}} = V^{\Delta'}(E,\delta)$$

where the bijection $\mathbb{U}^{\downarrow\mathbb{I}(s)(E,\delta)^{\mathrm{op}}} \cong \mathbb{U}^{\downarrow(E,\delta)^{\mathrm{op}}}$ is natural in (E,δ). Thus $s^*(V^{\Delta}) \cong V^{\Delta'}$ as presheaves in $\mathfrak{GR}(\Delta')$. Concretely, then, we define c_s^V to be

$$\left(c_s^V\right)_{(E,\delta)}(D) = D \circ \mathbb{I}(s)$$

and its inverse is

$$\left(c_s^V\right)_{(E,\delta)}^{-1}(D) = D \circ u((E,\delta),-)$$

The map c^E is defined analogously. Equalities (5.8) and (5.9) follow directly from (contravariant) functoriality of $\mathbb{I}$. $\mathfrak{QED}$

Remark 5.5.13. Inspection of the proof also shows why for inclusions $\iota : \Delta \to \Delta,\kappa$ the reindexing ι^* does not preserve universes in this way: the function $\mathbb{I}(\iota)$ restricted to $\downarrow(E,\Delta) \to \downarrow\mathbb{I}(\iota)(E,\delta)$ is not a bijection. This is consistent with the situation as it was in Møgelberg's previous model [71] and so following *loc. cit.* we add additional universes in each $\mathfrak{GR}(\Delta)$. ♦

Definition 5.5.14. Let $\Delta' \subseteq \Delta \subseteq^{\mathrm{fin}} \mathrm{CV}$ be clock contexts and $\iota : \Delta' \to \Delta$ the inclusion. We define the universe

$$\mathfrak{u}_{\Delta'}^{\Delta} : \mathcal{E}_{\Delta'}^{\Delta} \to \mathcal{U}_{\Delta'}^{\Delta}$$

as

$$\mathcal{U}_{\Delta'}^{\Delta} = \iota^*\left(V^{\Delta'}\right) \qquad \mathcal{E}_{\Delta'}^{\Delta} = \iota^*\left(E_{\Delta'}^{V}\right) \qquad \mathfrak{u}_{\Delta'}^{\Delta} = \iota^*\left(u^{\Delta'}\right). \qquad ♦$$

Of course we need to justify calling $\mathfrak{u}_{\Delta'}^{\Delta}$ a universe. This is not immediate and relies on ι^* being a locally cartesian closed functor preserving all coproducts. To prove that $\mathfrak{u}_{\Delta'}^{\Delta}$ are universes we will need some general facts about universes in locally cartesian closed categories stated in the next two lemmas.

Universes and generic operations Let $\mathbb{C}$ be a locally cartesian closed category. Let $\mathrm{el} : E \to U$ be a morphism in $\mathbb{C}$. We characterise when the notion of smallness induced by el is closed under composition and dependent products in the following two lemmas. To state the lemmas we need some preliminary definitions. We return back to our specific model in Theorem 5.5.17 on page 5.5.17 below.

Let $\pi_1 : U \times U \to U$ and $\mathrm{el} : E \to U$ be two maps in the slice over U and let $\mathfrak{p}_1 : U_1 \to U$ be the exponential $\mathrm{el} \Rightarrow \pi_1$ over U. Using the internal language we can describe U_1 as the dependent sum

$$U_1 = \sum_{x:U} U^{El(x)}.$$

Define E_1 and $\mathfrak{p}_2$ as the pullback

$$\begin{tikzcd}
E_1 \arrow[r, "w_1"] \arrow[d, "\mathfrak{p}_2"] \arrow[dr, phantom, "\lrcorner", very near start] & E \arrow[d, "\mathrm{el}"] \\
U_1 \arrow[r, "\mathfrak{p}_1"] & U
\end{tikzcd}$$

Using the internal language we may describe E_1 as the dependent sum

$$E_1 = \sum_{y:U_1} El(\pi_1(y))$$

Finally, define E_2 and $\mathfrak{p}_3$ as the pullback

$$\begin{tikzcd}
E_2 \arrow[r, "w_2"] \arrow[d, "\mathfrak{p}_3"] \arrow[dr, phantom, "\lrcorner", very near start] & E \arrow[d, "\mathrm{el}"] \\
E_1 \arrow[r, "\mathfrak{e}"] & U
\end{tikzcd}$$

where $\mathfrak{e} : E_1 \to U$ is defined as

$$\mathfrak{e} = \pi_2 \circ \mathbf{eval}$$

where $\mathbf{eval} : \mathfrak{p}_1 \times_U el \to \pi_1$ is the evaluation map for the local exponential U_1. This makes sense because $\mathfrak{p}_2$ is the product of of $\mathfrak{p}_1$ and el in the slice over U.

Using the internal language E_2 can be described as

$$E_2 = \sum_{z:E_1} El\left((\pi_2(\pi_1(z)))(\pi_2(z))\right)$$

or using pattern-matching notation as

$$E_2 = \sum_{(x,G,y):E_1} El\left(G(y)\right)$$

Lemma 5.5.15. *The morphism $\mathfrak{p}_2 \circ \mathfrak{p}_3$ is U-small if and only if the notion of smallness induced by* el *is closed under composition.* ◊

Proof. The morphisms $\mathfrak{p}_2$ and $\mathfrak{p}_3$ are small by definition and if smallness is closed under composition then their composition is obviously small.

For the converse assume $\mathfrak{p}_2 \circ \mathfrak{p}_3$ is U-small and let $f : B \to A$ and $g : C \to B$ be U-small witnessed by the codes $\overline{\mathrm{f}} : A \to U$ and $\overline{\mathrm{g}} : B \to U$. Because

$$\begin{tikzcd}
B \arrow[r, "\underline{\mathrm{f}}"] \arrow[d, "f"] \arrow[dr, phantom, very near start, "\lrcorner"] & E \arrow[d, "\mathrm{el}"] \\
A \arrow[r, "\overline{\mathrm{f}}"] & U
\end{tikzcd}$$

is a pullback we have $\overline{\mathrm{f}} \circ f = \overline{\mathrm{f}} \times_U \mathrm{el}$.

Define $m : \overline{\mathrm{f}} \times_U \mathrm{el} \to \pi_1$ over U to be the morphism $\left\langle \overline{\mathrm{f}} \circ f, \overline{\mathrm{g}} \right\rangle$. Hence we can take its exponential transpose $\hat{m} : \overline{\mathrm{f}} \to \mathfrak{p}_1$ over U. In particular we have $\hat{m} : A \to U_1$. Now consider the diagram.

$$\begin{tikzcd}
C \arrow[r, "\left\langle \langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1} \circ g, \underline{\mathrm{g}} \right\rangle_{E_2}"] \arrow[d, "g"] & E_2 \arrow[r] \arrow[d, "\mathfrak{p}_3"] & E \arrow[dd, "\mathrm{el}"] \\
B \arrow[r, "\langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1}"] \arrow[d, "f"] & E_1 \arrow[d, "\mathfrak{p}_2"] & \\
A \arrow[r, "\hat{m}"] & U_1 \arrow[r, "\overline{\mathfrak{p}_2 \circ \mathfrak{p}_3}"] & U
\end{tikzcd} \tag{5.10}$$

where $\left\langle \langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1} \circ g, \underline{\mathrm{g}} \right\rangle_{E_2}$ and $\langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1}$ are the maps into the respective pullbacks given by the universal property.

It is obvious that the lower left and the right squares are well-defined and commute. To see that the upper left is well-defined observe that $\langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1}$ is the product of maps $\hat{m} \times_U \mathrm{id}_{\mathrm{el}}$ in the slice over U. Hence

$$\mathfrak{e} \circ \langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1} = \overline{\mathrm{g}} \circ g = \mathrm{el} \circ \underline{\mathrm{g}}$$

so all the morphisms are well-defined. The upper left square thus commutes by definition.

Now observe that both

$$\begin{tikzcd}
B \arrow[r, "\langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1}"] \arrow[d, "f"] \arrow[rr, bend left, "\underline{\mathrm{f}}"] & E_1 \arrow[r, "w_1"] \arrow[d, "\mathfrak{p}_2"] & E \arrow[d, "el"] \\
A \arrow[r, "\hat{m}"] \arrow[rr, bend right, "\overline{\mathrm{f}}"'] & U_1 \arrow[r, "\mathfrak{p}_1"] & U
\end{tikzcd}$$

and

$$\begin{tikzcd}
C \arrow[r, "\left\langle \langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1} \circ g, \underline{\mathrm{g}} \right\rangle_{E_2}"] \arrow[rr, bend left, "\underline{\mathrm{g}}"] \arrow[d, "g"] & E_2 \arrow[r, "w_2"] \arrow[d, "\mathfrak{p}_3"] & E \arrow[d, "el"] \\
B \arrow[r, "\langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1}"] \arrow[rr, bend right, "\overline{\mathrm{g}}"'] & E_1 \arrow[r, "\mathfrak{c}"] & U
\end{tikzcd}$$

commute and since the right and outer ones are pullbacks by assumption, so is the left inner one. Hence by another application of the pullback lemma so is the diagram (5.10) above, meaning that $\overline{\mathfrak{p}_2 \circ \mathfrak{p}_3} \circ \hat{m}$ is a code for $f \circ g$, so $f \circ g$ is small. $\mathfrak{QED}$

Lemma 5.5.16. *The notion of smallness induced by* el *is closed under small dependent products if and only if* $\prod_{\mathfrak{p}_2} \mathfrak{p}_3 : \prod_{\mathfrak{p}_2} E_2 \to U_1$ *is small.* ◊

Proof. If U is closed under small dependent products then $\prod_{\mathfrak{p}_2} \mathfrak{p}_3$ is small because $\mathfrak{p}_2$ and $\mathfrak{p}_3$ are small.

For the converse let $f : B \to A$ and $g : C \to B$ be small and let

$$\begin{tikzcd}
C \arrow[r, "\left\langle \langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1} \circ g, \underline{\mathrm{g}} \right\rangle_{E_2}"] \arrow[d, "g"] & E_2 \arrow[d, "\mathfrak{p}_3"] \\
B \arrow[r, "\langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1}"] \arrow[d, "f"] & E_1 \arrow[d, "\mathfrak{p}_2"] \\
A \arrow[r, "\hat{m}"] & U_1
\end{tikzcd}$$

be the pullback (in particular the bottom square is a pullback) as in the proof of Lemma 5.5.15 above.

Because the bottom square is a pullback we can use the Beck-Chevalley property to get that the canonical natural transformation

$$\mathbf{can} : \hat{m}^* \circ \prod_{\mathfrak{p}_2} \to \prod_{f} \circ \left(\langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1} \right)^*$$

is an isomorphism. In particular the component at $\mathfrak{p}_3$ is an isomorphism

$$\mathbf{can}_{\mathfrak{p}_3} : \hat{m}^* \left(\prod_{\mathfrak{p}_2} (\mathfrak{p}_3) \right) \to \prod_{f} \left(\left(\langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1} \right)^* (\mathfrak{p}_3) \right)$$

Now observe that $\left(\langle \hat{m} \circ f, \underline{\mathrm{f}} \rangle_{E_1}\right)^{*}(\mathfrak{p}_3) = g$ and so we have that the square

$$\begin{tikzcd}
\prod_f B \arrow[r] \arrow[d, "\prod_f g"] & \prod_{\mathfrak{p}_2} E_2 \arrow[d, "\prod_{\mathfrak{p}_2} \mathfrak{p}_3"] \\
A \arrow[r, "\hat{m}"] & U_1
\end{tikzcd}$$

is also a pullback and so because $\prod_{\mathfrak{p}_2} \mathfrak{p}_3$ is small, so is $\prod_f g$ and this is witnessed by a code $\overline{\prod_{\mathfrak{p}_2} \mathfrak{p}_3} \circ \hat{m}$. $\mathfrak{QED}$

The two lemmas lead to the following theorem.

Theorem 5.5.17. *For any $\Delta' \subseteq \Delta \subseteq^{\mathrm{fin}} \mathrm{CV}$ the morphism $\mathfrak{u}^{\Delta}_{\Delta'}$ is a universe in $\mathfrak{GR}(\Delta)$ in the sense of Definition 5.5.2.* ◊

Proof. Since 0, 1 and $\mathbb{N}$ are constant presheaves they are small with respect to $\mathfrak{u}^{\Delta}_{\Delta'}$ as discussed above.

To see that $\mathfrak{u}^{\Delta}_{\Delta'}$ is closed under small dependent products we only need to show (Lemma 5.5.16) that a particular dependent product is small.

Because $u^{\Delta'}$ is a universe (Definition 5.5.5) it is closed under small dependent products. Hence defining $\mathfrak{p}_2$ and $\mathfrak{p}_3$ as on page 207 but using the universe $u^{\Delta'}$ in $\mathfrak{GR}(\Delta')$ we have that $\prod_{\mathfrak{p}_2} p_3$ is $u^{\Delta'}$-small. Because ι^* preserves pullbacks and local exponentials (Theorem 5.2.13) the maps $\mathfrak{p}_2'$ and $\mathfrak{p}_3'$ constructed from $\mathfrak{u}^{\Delta}_{\Delta'}$ are simply $\iota^*(\mathfrak{p}_2)$ and $\iota^*(\mathfrak{p}_3)$.

Because ι^* preserves pullbacks the map $\iota^*\left(\prod_{\mathfrak{p}_2} \mathfrak{p}_3\right)$ is $\mathfrak{u}^{\Delta}_{\Delta'}$-small. Because ι^* preserves dependent products (it is a locally cartesian closed functor, and dependent products can be constructed from local exponentials), we have $\iota^*\left(\prod_{\mathfrak{p}_2} \mathfrak{p}_3\right) \cong \prod_{\iota^*(\mathfrak{p}_2)} \iota^*(\mathfrak{p}_3) \cong \prod_{\mathfrak{p}_2'} \mathfrak{p}_3'$. Finally, being small is closed under isomorphisms, hence $\prod_{\mathfrak{p}_2'} \mathfrak{p}_3'$ is $\mathfrak{u}^{\Delta}_{\Delta'}$-small, as needed.

The case for composition is entirely analogous. $\mathfrak{QED}$

Proposition 5.5.18. *Let $\Delta' \subseteq \Delta \subseteq^{\mathrm{fin}} \mathrm{CV}$ and $\kappa \in \Delta'$. There is a morphism $\widehat{\triangleright}^{\kappa} : \blacktriangleright^{\kappa} \mathcal{U}^{\Delta}_{\Delta'} \to \mathcal{U}^{\Delta}_{\Delta'}$ making the following diagram a pullback.*

$$\begin{tikzcd}
\blacktriangleright^{\kappa} \mathcal{E}^{\Delta}_{\Delta'} \arrow[r] \arrow[d, "\blacktriangleright^{\kappa} \mathfrak{u}^{\Delta}_{\Delta'}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{E}^{\Delta}_{\Delta'} \arrow[d, "\mathfrak{u}^{\Delta}_{\Delta'}"] \\
\blacktriangleright^{\kappa} \mathcal{U}^{\Delta}_{\Delta'} \arrow[r, "\widehat{\triangleright}^{\kappa}"] & \mathcal{U}^{\Delta}_{\Delta'}
\end{tikzcd}$$

As a consequence, if $\alpha : A \to \Gamma$ is $\mathfrak{u}^{\Delta}_{\Delta'}$-small with code $\overline{\alpha}$ then $\blacktriangleright^{\kappa}_{\Gamma} A$ is $\mathfrak{u}^{\Delta}_{\Delta'}$-small with code

$$\widehat{\triangleright}^{\kappa} \circ \mathrm{next}^{\kappa}_{\mathcal{U}^{\Delta}_{\Delta'}} \circ \overline{\alpha}.$$ ◊

Proof. Lemma 5.5.7 provides a $\widehat{\triangleright^\kappa} : \blacktriangleright^\kappa V^{\Delta'} \to V^{\Delta'}$ making the following diagram a pullback.

$$\begin{tikzcd}
\blacktriangleright^\kappa E^V_{\Delta'} \arrow[r] \arrow[d, "\blacktriangleright^\kappa u^{\Delta'}"'] \arrow[dr, phantom, "\lrcorner", very near start] & E^V_{\Delta'} \arrow[d, "u^{\Delta'}"] \\
\blacktriangleright^\kappa V^{\Delta'} \arrow[r, "\widehat{\triangleright^\kappa}"] & V^{\Delta'}
\end{tikzcd}$$

Because $\iota^* : \mathfrak{GR}(\Delta') \to \mathfrak{GR}(\Delta)$ preserves pullbacks the diagram

$$\begin{tikzcd}
\iota^*\left(\blacktriangleright^\kappa E^V_{\Delta'}\right) \arrow[r] \arrow[d, "\iota^*\left(\blacktriangleright^\kappa u^{\Delta'}\right)"'] \arrow[dr, phantom, "\lrcorner", very near start] & \iota^*\left(E^V_{\Delta'}\right) \arrow[d, "\mathfrak{u}^{\Delta'}_{\Delta'}"] \\
\iota^*\left(\blacktriangleright^\kappa V^{\Delta'}\right) \arrow[r, "\iota^*(\widehat{\triangleright^\kappa})"] & \iota^*\left(V^{\Delta'}\right)
\end{tikzcd}$$

is also a pullback. By Proposition 5.3.5 we have, e.g.,

$$\iota^*\left(\blacktriangleright^\kappa u^{\Delta'}\right) = \blacktriangleright^\kappa \mathfrak{u}^{\Delta}_{\Delta'}$$

hence defining the new $\widehat{\triangleright^\kappa}$ to be $\iota^*(\widehat{\triangleright^\kappa})$ proves the first part of the proposition.

The second part follows analogously from Proposition 5.5.8. 𝔔𝔈𝔇

Proposition 5.5.19. *Let $\Delta' \subseteq \Delta \subseteq^{\text{fin}} \mathrm{CV}$ and κ a clock not in Δ. Let $A, \Gamma \in \mathfrak{GR}(\Delta, \kappa)$ and $\alpha : A \to \Gamma$ a morphism. If α is small with respect to $\mathfrak{u}^{\Delta,\kappa}_{\Delta',\kappa}$ then the morphism $\forall\kappa.\alpha \in \mathfrak{GR}(\Delta)$ is small with respect to $\mathcal{U}^{\Delta}_{\Delta'}$.*

In particular there exists a morphism $\overline{\forall} : \forall\kappa.\mathcal{U}^{\Delta,\kappa}_{\Delta',\kappa} \to \mathcal{U}^{\Delta}_{\Delta'}$ making the following diagram a pullback.

$$\begin{tikzcd}
\forall\kappa.\mathcal{E}^{\Delta,\kappa}_{\Delta',\kappa} \arrow[r] \arrow[d, "\forall\kappa.\mathfrak{u}^{\Delta,\kappa}_{\Delta',\kappa}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{E}^{\Delta}_{\Delta'} \arrow[d, "\mathfrak{u}^{\Delta}_{\Delta'}"] \\
\forall\kappa.\mathcal{U}^{\Delta,\kappa}_{\Delta',\kappa} \arrow[r, "\overline{\forall}"] & \mathcal{U}^{\Delta}_{\Delta'}
\end{tikzcd}$$

Let $\upsilon : \Delta \to \Delta, \kappa$ be the inclusion, $\Gamma \in \mathfrak{GR}(\Delta)$, $A \in \mathfrak{GR}(\Delta, \kappa)$ and $\alpha : A \to \iota^*(\Gamma)$ a morphism.

If α is small with respect to $\mathcal{U}^{\Delta,\kappa}_{\Delta',\kappa}$ then $\forall_\Gamma\kappa.\alpha \in \mathfrak{GR}(\Delta)/\Gamma$ is small with respect to $\mathcal{U}^{\Delta}_{\Delta'}$. Moreover, if $\overline{\alpha}$ is a code for α then $\overline{\forall} \circ (\forall\kappa.\overline{\alpha}) \circ \eta^\Gamma$ is a code for $\forall_\Gamma.\alpha$. ◊

Proof. From Lemma 5.5.10 we have that the diagram

$$\begin{tikzcd}
\forall\kappa.E^V_{\Delta',\kappa} \arrow[r] \arrow[d, "\forall\kappa.u^{\Delta',\kappa}"'] \arrow[dr, phantom, "\lrcorner", very near start] & E^V_{\Delta'} \arrow[d, "u^{\Delta'}"] \\
\forall\kappa.V^{\Delta',\kappa} \arrow[r, "\overline{\forall}"] & V^{\Delta'}
\end{tikzcd}$$

is a pullback. Let $\iota : \Delta' \to \Delta$ be the inclusion. Because ι^* preserves pullbacks the following square is likewise a pullback.

$$\begin{tikzcd}
\forall\kappa.\iota^*\left(E^V_{\Delta',\kappa}\right) \arrow[r] \arrow[d, "\iota^*\left(\forall\kappa.u^{\Delta',\kappa}\right)"'] \arrow[dr, phantom, "\lrcorner", very near start] & \iota^*\left(E^V_{\Delta'}\right) \arrow[d, "\iota^*\left(u^{\Delta'}\right)"] \\
\iota^*\left(\forall\kappa.V^{\Delta',\kappa}\right) \arrow[r, "\iota^*(\overline{\forall})"] & \iota^*\left(V^{\Delta'}\right)
\end{tikzcd}$$

By the Beck-Chevalley condition for $\forall\kappa$ (Proposition 5.4.4) we have, e.g.,

$$\iota^*\left(\forall\kappa.u^{\Delta',\kappa}\right) = \forall\kappa.\iota'^*\left(u^{\Delta',\kappa}\right)$$

where $\iota' : \Delta',\kappa \to \Delta,\kappa$ is the inclusion. By definition $\iota'^*\left(u^{\Delta',\kappa}\right) = \mathcal{U}^{\Delta,\kappa}_{\Delta',\kappa}$. Hence defining the new $\overline{\forall}$ as $\iota^*(\overline{\forall})$ proves the first part of the proposition.

The second part follows analogously from Proposition 5.5.11 together with Propositions 5.4.4 and 5.4.5. 𝔔𝔈𝔇

Finally, these additional universes are preserved by clock substitution in the appropriate way.

Proposition 5.5.20. *Let $f : \Delta_1 \to \Delta_2$ be a function between clock contexts Δ_1 and Δ_2. Let $\Delta' \subseteq \Delta_1$ be another clock context and $\mathfrak{u}^{\Delta_1}_{\Delta'} : \mathcal{E}^{\Delta_1}_{\Delta'} \to \mathcal{U}^{\Delta_1}_{\Delta'}$ the universe in $\mathfrak{GR}(\Delta_1)$ from Definition 5.5.14. There exist two* natural isomorphisms $c^{f,\Delta'}_E$ *and $c^{f,\Delta'}_V$ such that the diagram*

Recall that $f[\Delta']$ is the image of the set Δ' under the function f.

$$\begin{tikzcd}
f^*\left(\mathcal{E}^{\Delta_1}_{\Delta'}\right) \arrow[r, "c^{f,\Delta'}_E"] \arrow[d, "f^*\left(\mathfrak{u}^{\Delta_1}_{\Delta'}\right)"'] & \mathcal{E}^{\Delta_2}_{f[\Delta']} \arrow[d, "\mathfrak{u}^{\Delta_2}_{f[\Delta']}"] \\
f^*\left(\mathcal{U}^{\Delta_1}_{\Delta'}\right) \arrow[r, "c^{f,\Delta'}_V"] & \mathcal{U}^{\Delta_2}_{f[\Delta']}
\end{tikzcd}$$

commutes. In particular, $f^*\left(\mathcal{U}^{\Delta_1}_{\Delta'}\right) \cong \mathcal{U}^{\Delta_2}_{f[\Delta']}$.

Moreover, for any $g : \Delta_2 \to \Delta_3$ we have

$$c^{g,f[\Delta']}_V \circ g^*\left(c^{f,\Delta'}_V\right) = c^{g\circ f,\Delta'}_V \tag{5.11}$$

and

$$c^{g,f[\Delta']}_E \circ g^*\left(c^{f,\Delta'}_E\right) = c^{g\circ f,\Delta'}_E. \tag{5.12}$$

◊

Proof. Let ι_1 be the inclusion of Δ' into Δ_1 and ι_2 the inclusion of $f[\Delta']$ into Δ_2. Let $s : \Delta' \to f[\Delta']$ be the restriction of f. By definition s is *surjective* and $f \circ \iota_1 = \iota_2 \circ s$ and so $f^* \circ \iota_1^* = \iota_2^* \circ s^*$. Lemma 5.5.12 gives natural isomorphisms c_s^V and c_s^E such that the diagram on the left

$$\begin{tikzcd}
s^*\left(E^V_{\Delta'}\right) \arrow[r, "c_s^E"] \arrow[d, "s^*\left(u^{\Delta'}\right)"'] & E^V_{f[\Delta']} \arrow[d, "u^{f[\Delta']}"] \\
s^*\left(V^{\Delta'}\right) \arrow[r, "c_s^V"] & V^{f[\Delta']}
\end{tikzcd}
\qquad
\begin{tikzcd}
\iota_2^*\left(s^*\left(E^V_{\Delta'}\right)\right) \arrow[r, "\iota_2^*(c_s^E)"] \arrow[d, "\iota_2^*\left(s^*\left(u^{\Delta'}\right)\right)"'] & \iota_2^*\left(E^V_{f[\Delta']}\right) \arrow[d, "\iota_2^*\left(u^{f[\Delta']}\right)"] \\
\iota_2^*\left(s^*\left(V^{\Delta'}\right)\right) \arrow[r, "\iota_2^*(c_s^V)"] & \iota_2^*\left(V^{f[\Delta']}\right)
\end{tikzcd}$$

commutes. Hence the diagram on the right commutes and the horizontal morphisms are isomorphisms. But notice that, e.g.,

$$\iota_2^*\left(s^*\left(V^{\Delta'}\right)\right) = f^*\left(\iota_1^*\left(V^{\Delta'}\right)\right) = f^*\left(\mathcal{U}^{\Delta_1}_{\Delta'}\right)$$

and by definition $\iota_2^*\left(V^{f[\Delta']}\right) = \mathcal{U}^{\Delta_2}_{f[\Delta']}$. This proves the first part.

The equalities (5.11) and (5.12) follow directly from equalities(5.8) and (5.9) from Lemma 5.5.12. 𝔔𝔈𝔇

Remark 5.5.21. The preceding proposition shows in particular that there is a universe which is preserved by clock substitution. This is the universe $\mathcal{U}^{\emptyset}_{\emptyset}$. It is preserved in the sense that we could define in each category $\mathfrak{GR}(\Delta)$ a single universe $\mathcal{U}^{\Delta}_{\emptyset}$ and these universes would be preserved by clock substitution. In fact, they would even be preserved on the nose.

A question might then appear as to why we need more universes? The answer is that the universes $\mathcal{U}^{\Delta}_{\emptyset}$ are not closed under $\blacktriangleright^{\kappa}$ (in the sense of Proposition 5.5.18). Closure under $\blacktriangleright^{\kappa}$ is however crucial for defining types as fixed points on universes as explained in [18]. ♦

Universe inclusions

The final question is how the universes $\mathfrak{u}^{\Delta}_{\Delta'}$ and $\mathfrak{u}^{\Delta}_{\Delta''}$ are related. The answer is in the following proposition.

Proposition 5.5.22. *Let $\Delta'' \subseteq \Delta \subseteq^{\text{fin}} \mathrm{CV}$ be two clock contexts. There is a monomorphism*

$$\mathfrak{in}^{\Delta}_{\Delta'',\Delta'} : \mathcal{U}^{\Delta}_{\Delta''} \to \mathcal{U}^{\Delta}_{\Delta'}$$

that fits into the following pullback square.

$$\begin{tikzcd}
\mathcal{E}^{\Delta}_{\Delta''} \arrow[r] \arrow[d, "\mathfrak{u}^{\Delta}_{\Delta''}"] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{E}^{\Delta}_{\Delta'} \arrow[d, "\mathfrak{u}^{\Delta}_{\Delta'}"] \\
\mathcal{U}^{\Delta}_{\Delta''} \arrow[r, "\mathfrak{in}^{\Delta}_{\Delta'',\Delta'}"] & \mathcal{U}^{\Delta}_{\Delta'}
\end{tikzcd}$$

If $\Delta''' \subseteq \Delta''$ is a further clock context then also

$$\mathfrak{in}^{\Delta}_{\Delta''',\Delta'} = \mathfrak{in}^{\Delta}_{\Delta'',\Delta'} \circ \mathfrak{in}^{\Delta}_{\Delta''',\Delta''}. \tag{5.13}$$

Finally if $\Delta_1 \subseteq^{\text{fin}} \mathrm{CV}$ and $f : \Delta \to \Delta_1$ is any function then the following diagram commutes.

$$\begin{tikzcd}
f^*\left(\mathcal{U}^{\Delta}_{\Delta''}\right) \arrow[r, "f^*\left(\mathfrak{in}^{\Delta}_{\Delta'',\Delta'}\right)"] \arrow[d, "c^{f,\Delta''}_{V}"] & f^*\left(\mathcal{U}^{\Delta}_{\Delta'}\right) \arrow[d, "c^{f,\Delta'}_{V}"] \\
\mathcal{U}^{\Delta_1}_{f[\Delta'']} \arrow[r, "\mathfrak{in}^{\Delta_1}_{f[\Delta''],f[\Delta']}"] & \mathcal{U}^{\Delta_1}_{f[\Delta']}
\end{tikzcd} \tag{5.14}$$

◊

Proof. Since the components of $\mathfrak{u}^{\Delta'}_{\Delta''} \in \mathfrak{GR}(\Delta')$ are the components of $u^{\Delta''} \in \mathfrak{GR}(\Delta'')$ we have that all components of $\mathfrak{u}^{\Delta'}_{\Delta''}$ are el-small for the universe el from Example 5.5.4. Hence by Proposition 5.5.6 there is a morphism $\mathfrak{in}^{\Delta'}_{\Delta'',\Delta'} : \mathcal{U}^{\Delta'}_{\Delta''} \to \mathcal{U}^{\Delta'}_{\Delta'} = V^{\Delta'}$. Define $\mathfrak{in}^{\Delta}_{\Delta'',\Delta'}$ to be $\iota^*\left(\mathfrak{in}^{\Delta'}_{\Delta'',\Delta'}\right)$ where $\iota : \Delta' \to \Delta$ is the inclusion.

This argument shows that such a morphism exists, however to verify that it is a monomorphism and that (5.13) and (5.14) hold we need a more useful description. The morphism $\mathfrak{in}^{\Delta}_{\Delta'',\Delta'}$ is defined to be

$$\left(\mathfrak{in}^{\Delta}_{\Delta'',\Delta'}\right)_{E,\delta}(D)(E',\delta') = D\left(\mathfrak{I}\left(\iota^{\Delta'}_{\Delta''}\right)(E',\delta')\right)$$

where $\mathfrak{I}\left(\iota^{\Delta'}_{\Delta''}\right) : \Delta'' \to \Delta'$ is the inclusion.

Using this description (5.13) and (5.14) are easy to verify by direct computation. The fact that it is a monomorphism follows from the fact that $\mathfrak{I}\left(\iota^{\Delta'}_{\Delta''}\right)$ is surjective (Lemma 5.2.5). $\mathfrak{QED}$

Using this proposition we have, for instance, that if $\bar{\mathfrak{f}}$ is a $\mathfrak{u}^{\Delta}_{\Delta''}$ code for a map $f : A \to \Gamma$ then $\mathfrak{in}^{\Delta}_{\Delta'',\Delta'} \circ \bar{\mathfrak{f}}$ is a $\mathfrak{u}^{\Delta}_{\Delta'}$ code for f. This follows directly from the pullback pasting lemma.

5.6 Discussion

We have constructed an indexed category $\mathfrak{GR}$. For each $\Delta \subseteq^{\text{fin}} \mathrm{CV}$ the category $\mathfrak{GR}(\Delta)$ supports a model of extensional dependent type theory and for each function $f : \Delta_1 \to \Delta_2$ the functor $\mathfrak{GR}(f)$ preserves the structure needed to model dependent type theory. The indexed category can thus be made into an instance of a PDTT-structure (5.2) (on page 183). However it is not a *split* PDTT-structure. We now discuss what are the problems with splitting.

Since each $\mathfrak{GR}(\Delta)$ is a presheaf topos there are known constructions ([48, 51, 77]) for making $\mathfrak{GR}(\Delta)$ a split closed comprehension category, with strong sums and strong identity types [51], i.e., a model of extensional dependent type theory.

The source of problems is clock weakening. The first indication of this appears in Lemma 5.5.12 and the remark following it which force us to add more universes in each $\mathfrak{GR}(\Delta)$ and universe inclusions (Proposition 5.5.22). A desirable property of universe inclusions is that they commute with all the structure on the nose. For instance, if

$$\hat{\rightarrow}_{\Delta''} : \mathcal{U}^{\Delta}_{\Delta''} \times \mathcal{U}^{\Delta}_{\Delta''} \to \mathcal{U}^{\Delta}_{\Delta''}$$

is the chosen code (recall that to have a split PDTT-structure means in particular to make a choice of all structure) for the exponential operation in the universe $\mathcal{U}^{\Delta}_{\Delta''}$ and $\hat{\rightarrow}_{\Delta'}$ is the chosen code for the exponential operation in $\mathcal{U}^{\Delta}_{\Delta'}$ then we would want to have the equality of codes

$$\hat{\rightarrow}_{\Delta'} \circ \left\langle \mathrm{in}^{\Delta}_{\Delta'',\Delta'} \circ \bar{\mathrm{f}}, \mathrm{in}^{\Delta}_{\Delta'',\Delta'} \circ \bar{\mathrm{g}} \right\rangle = \mathrm{in}^{\Delta}_{\Delta'',\Delta'} \circ \hat{\rightarrow}_{\Delta''} \circ \left\langle \bar{\mathrm{f}}, \bar{\mathrm{g}} \right\rangle$$

for chosen codes $\bar{\mathrm{f}}$ and $\bar{\mathrm{g}}$ of maps f and g, respectively. At present we do not know how to make this choice.

Similarly, we do not know how to make a choice of dependent products in $\mathfrak{GR}(\Delta)$ in such a way that they would be preserved on the nose by clock substitution f^*.

The essence of all the problems is that exponentials (and dependent products) in presheaf categories are constructed using "Kripke" quantification which means that they reflect the indexing category that is used. Hence changing the indexing category, as for example when when moving from $\mathfrak{GR}(\Delta_1)$ to $\mathfrak{GR}(\Delta_2)$, means that the natural choice of exponentials in presheaf categories is very rarely preserved on the nose. We are working with quite specific presheaf categories though, so perhaps the categories can be presented in a better way or alternatively the exponentials and dependent products could be constructed in a different way.

We believe this is a technical, rather than essential, problem with the particular presentation. In particular, without universes, there is a solution to the coherence problem by replacing the categories $\mathfrak{GR}(\Delta)$ by equivalent ones obtained by the Bénabou construction [16] (see also [51, Corollary 5.2.5]) applied to the (fibration obtained from the) indexed category $\mathfrak{GR}$. This then allows us to make choices of structure that are preserved on the nose by functors interpreting clock substitution.

In related work, the model in Chapter 6 is a split PDTT-structure by construction obtained in a substantially simpler way. The reason that clock substitution preserves all the structure on the nose is precisely that, for instance, exponentials do not reflect directly the indexing poset used. Instead, the exponential is a set of functions satisfying some *property* and since clock substitution does not change the underlying sets, but only the property imposed by the indexing poset, it preserves the exponential, since it preserves the property. For technical details on how this works see Chapter 6.

Chapter 6

A Model of Guarded Recursion via Generalised Equilogical Spaces

This chapter is a slightly revised version of

> Aleš Bizjak and Lars Birkedal.
>
> A model of guarded recursion via generalised equilogical spaces.
>
> *???*, 2015.
>
> Under review

which is currently under review.

Abstract

We present a new model, called GuardedEqu, of guarded dependent type theory using generalised equilogical spaces. GuardedEqu models guarded recursive types, which can be used to program with coinductive types and we prove that GuardedEqu ensures that all definable functions on coinductive types, e.g., streams, are continuous with respect to the natural topology. We present a direct, elementary, construction of the new model, which, importantly, is coherent (split) by construction.

6.1 Introduction

Type theories with support for guarded recursive functions and guarded recursive types are useful for programming with coinductive types and also for serving as a meta-theory for constructing sophisticated models of programming languages with effects [22, 91].

In this paper, we present a new model of guarded dependent type theory, based on a generalisation of equilogical spaces [14]. We refer to the new model as GuardedEqu.

In contrast to earlier models of guarded dependent type theory, GuardedEqu ensures that definable functions on coinductive types are suitably continuous. For example, any function definable on the type of streams is continuous with respect to the standard topology on streams. Thus, if f is such a function on streams and xs is a stream, a finite amount of the output $f(xs)$ only depends on a finite amount of the input xs. We prove that an analogous result holds for final coalgebras of arbitrary polynomial functors.

It is well-known that models of dependent type theory can be tricky to construct. A virtue of GuardedEqu is that the model construction is quite elementary and can be presented via a simple generalisation of constructions known from realizability models of type theory. An important feature of GuardedEqu is that it is coherent (split) by construction. A limitation of the model is that it does not include universes.

We now explain how GuardedEqu is related to earlier models of variations of guarded type theory.

Originally a type theory with a single ▶ modality for expressing guardedness was modelled using the category $\mathbf{PSh}(\omega)$, the topos of trees [22]. The model and the type theory allows for the solution of guarded recursive domain equations. It was later realised that guarded recursion can also be used for ensuring that functions producing values of coinductive types are productive in a precise sense. To support such encodings the type theory needs to be extended with the ability to eliminate ▶ in a controlled way. This led Atkey and McBride [11] to generalise ▶ to a family of modalities indexed by clocks, and to support clock quantification for controlled elimination of ▶. Atkey and McBride's development was for a simply typed calculus. They developed a model of their type theory and showed that, e.g., all streams definable in the calculus were were interpreted as actual streams, i.e., non bottom elements. Møgelberg [71] extended their work to a model of dependent type theory with universes. This model was subsequently refined [28] to support clock synchronisation which considerably simplified the calculus. As it currently stands this model is complex, in particular in its split form which is needed to soundly model the rules of guarded dependent type theory [27].

GuardedEqu can be seen as a generalisation of the model by Atkey and McBride to dependent types. The type theory we are considering has, to be useful, certain type isomorphisms [11, 71]. An example is the type isomorphism $\forall\kappa.\mathbb{N} \cong \mathbb{N}$, where $\mathbb{N}$ is the type of natural numbers. In Atkey and McBride's model these were type equalities, but in the presheaf models of guarded dependent type theory [28, 71] the types $\forall\kappa.\mathbb{N}$ and $\mathbb{N}$ are only modelled as canonically[1] isomorphic types. In GuardedEqu these type iso-

[1] Canonical means there is a term definable using just the ordinary introduction rules for

morphisms are again type equalities. This generalises the results of Atkey and McBride to a dependent type theory.

6.2 Guarded Dependent Type Theory

In this section we give a very brief introduction to the syntax of guarded dependent type theory. We refer the reader to [27] for the full set of typing rules, their motivation and more detailed explanation.

Guarded dependent type theory can be seen as a version of polymorphic dependent type theory [51]. It includes two contexts. A context Δ of clock variables $\kappa, \kappa', \cdots$ and a context Γ of ordinary term variables. Types depend on clocks, that is, clocks can appear in types, but clocks are only names in the sense that there are no constructions on clocks themselves. Guarded dependent type theory has the following basic judgements.

$$\begin{array}{l} \Gamma \vdash_\Delta \\ \Gamma \vdash_\Delta A \text{ type} \\ \Gamma \vdash_\Delta t : A \end{array}$$

The judgement $\Gamma \vdash_\Delta$ expresses that the free clocks in Γ are contained in Δ, the judgement $\Gamma \vdash_\Delta A$ type expresses that A is a well-formed type in context $\Gamma \vdash_\Delta$ and the last judgement expresses that t has type A in context $\Gamma \vdash_\Delta$. As usual in dependent type theory, there are also judgements for type and term equality for which we refer to [27]. Clocks are used to distinguish different $\blacktriangleright$ modalities. Thus, for each clock there is a modality $\blacktriangleright^\kappa$ and a term next^κ with the typing judgement

$$\frac{\Gamma \vdash_\Delta t : A}{\Gamma \vdash_\Delta \mathrm{next}^\kappa\, t : \blacktriangleright^\kappa A}\,\kappa \in \Delta$$

Clock weakening is admissible, e.g., there is a derivation of

$$\frac{\Gamma \vdash_\Delta A \text{ type}}{\Gamma \vdash_{\Delta,\kappa} A \text{ type}}$$

for $\kappa \notin \Delta$ and analogously for other judgements. Clock weakening has a right adjoint, which we write as $\forall\kappa$. The introduction rule is

$$\frac{\Gamma \vdash_\Delta \qquad \Gamma \vdash_{\Delta,\kappa} t : A}{\Gamma \vdash_\Delta \Lambda\kappa.t : \forall\kappa.A}$$

with the usual elimination rule for products and β-η rules for judgemental equalities. Thus in the model we shall need to have *polymorphic products* of *kinds* over *types* [51, Chapter 11].

$\forall\kappa$ of type $\mathbb{N} \to \forall\kappa.\mathbb{N}$ and the denotation of this term is an isomorphism.

A profitable way of thinking about guarded dependent type theory (and polymorphic dependent type theory in general) is as follows.

For each clock context Δ we have a core dependent type theory with ordinary type formers Π, Σ and equality types. In Section 6.3 we show how to model this fragment by constructing categories $\mathsf{PEqu}(P)$ and associated split closed comprehension categories with strong coproducts and strong equality. The construction is parametrised by a partially ordered set P. For the model of the core fragment it does not matter what P is. Later on, in Section 6.3, we instantiate P with specific posets which allow us to model the $\blacktriangleright$ modality and clock quantification $\forall\kappa$. Different clock contexts Δ will give rise to different posets P.

Next, to be able to change clock contexts, e.g., to interpret weakening, we need to be able to move between categories $\mathsf{PEqu}(P)$ for different P. Section 6.3 provides the necessary results which ensure that moving between $\mathsf{PEqu}(P)$ and $\mathsf{PEqu}(Q)$ preserves all the structure, e.g., products, coproducts. Section 6.3 then ties it all together into one model of guarded dependent type theory with two-level indexing. One for clock contexts and one for ordinary contexts. Section 6.3 provides a high-level summary of the model construction in the framework of fibrations. In Section 6.4 we prove how final coalgebras for polynomial functors can be obtained via guarded recursive types. And we prove the continuity properties for functions on final coalgebras for polynomial functors mentioned in the Introduction.

GuardedEqu validates all the rules of guarded dependent type theory apart from universes. We do not show soundness of all the rules in this article but only show the main constructions needed. Using these constructions the verification of all the rules is quite straightforward, albeit somewhat tedious to write out in all detail.

Remark 6.2.1. In the most recent formulation of guarded dependent type theory [27], guarded recursive types are defined via fixed points of functions on universes. Since we do not model universes in GuardedEqu, we would need some other facility for defining guarded recursive types syntactically, such as the one used in [22]. Formulating and modeling such guarded recursive types can be done without too much trouble, following [22].[2] Here, however, we do not include such a treatment, since that would not be particularly interesting, given all the other material. Instead we show in Section 6.4 that we can construct final coalgebras of polynomial functors using solutions of guarded type equations and we show that the solution comes equipped

[2] Indeed it is easy to show that $\mathsf{PEqu}(P)$ is enriched in the category of presheaves over P, which is equivalent to the category of sheaves over P equipped with the Alexandrov topology, so fits into the general framework [22] for a well-founded order P. Further, it is easy to see that the usual type constructors, $\rightarrow$, $\times$, $+$ give rise to enriched functors, and thus one can prove an existence theorem for fixed points of contractive functors following [22].

with the correct topology, which allows us to show expected productivity properties of functions defined on final coalgebras. ♦

6.3 GuardedEqu

Modelling core dependent type theory

First we explain some general constructions which do not deal directly with guarded recursion but are used later on. Let P be a partially ordered set. The category $\mathsf{PEqu}(P)$ has as objects pairs $A = (|A|, R_A)$ where

- $|A|$ is an algebraic lattice, i.e., a complete lattice where every element is the supremum of compact elements below it
- R_A is a monotone map from P^{op} to PERs on $|A|$ ordered by subset inclusion. In other words, R_A is a family of partial equivalence relations on $|A|$ indexed by P such that if $p \leq q$ then $R_A(p) \supseteq R_A(q)$.

We will sometimes write $a \approx_A^p a'$ for $(a, a') \in R_A(p)$.

Morphisms from A to B in $\mathsf{PEqu}(P)$ are equivalence classes, with respect to the relation $\sim$ defined below, of continuous *non-expansive* maps $|A| \to |B|$ (i.e., morphisms in AlgLat). The function f is non-expansive if it satisfies for all $p \in P$ the property

$$\forall (a, a') \in R_A(p), (f(a), f(a')) \in R_B(p).$$

The equivalence relation $\sim$ is defined as

$$f \sim g \leftrightarrow \forall p \in P, \forall (a, a') \in R_A(p), (f(a), g(a')) \in R_B(p).$$

Note that $\sim$ is an equivalence relation on non-expansive maps, but only a partial equivalence relation on general continuous maps. Indeed, if we define the relation $\sim$ on all maps then the non-expansive ones are precisely the ones in the domain of $\sim$.

This makes $\mathsf{PEqu}(P)$ into a category. Identities are given by equivalence classes of the identity functions in AlgLat. Composition of $[f] : A \to B$ and $[g] : B \to C$ is given by the equivalence class of $f \circ g$. It can easily be seen that this definition is independent of the choice of representatives f and g.

Remark 6.3.1. If P is the unique poset with one element then $\mathsf{PEqu}(P)$ is the category of partial equilogical spaces [14], equivalent to the category of equilogical spaces. ♦

To interpret dependent types we present the slice categories in a different way, similar to *uniform families* [51] but incorporating the monotonicity requirement.

Analogous to the way that the slice category $\mathbf{PSh}(\mathbb{C})/\Gamma$ of the category of presheaves $\mathbf{PSh}(\mathbb{C})$ over some presheaf Γ is equivalent to the category of presheaves over the category of elements of Γ we will present the slice categories of $\mathsf{PEqu}(P)$ using an auxiliary poset $\int_P \Gamma$. We now define this poset.

Let $\Gamma \in \mathsf{PEqu}(P)$. The poset $\int_P \Gamma$ has as elements pairs $\left(p, [\gamma]_p\right)$ where $p \in P$ and $[\gamma]_p$ is an equivalence class of $\gamma \in |\Gamma|$ with respect to the relation $R_\Gamma(p)$. In particular this means $(\gamma, \gamma) \in R_\Gamma(p)$. We define the order $\leq$ on $\int_P \Gamma$ as $(p, c) \leq (q, c')$ if and only if $p \leq q$ and $c \supseteq c'$. Or equivalently $(p, [\gamma]_p) \leq (q, [\gamma']_q)$ if $p \leq q$ and $(\gamma, \gamma') \in R_\Gamma(p)$.

Lemma 6.3.2. *The set $\int_P \Gamma$ with the order relation $\leq$ defined above is a poset.* ◊

Note that in contrast to the situation with presheaves, it is not the case that the category $\mathsf{PEqu}(P)/\Gamma$ is equivalent to the category $\mathsf{PEqu}(\int_P \Gamma)$. The problem is that the latter category has too few morphisms. However its objects are precisely the ones needed. To get a category equivalent to the slice category we define a new category $\mathsf{Type}_P(\Gamma)$.[3] Its objects are the objects of $\mathsf{PEqu}(\int_P \Gamma)$. A morphism from A to B is an equivalence class of continuous functions $|\Gamma| \to |A| \to |B|$ in AlgLat with respect to the partial equivalence relation $\sim_\Gamma$ which relates f and f' if and only if

$$\forall p \in P, \forall (\gamma, \gamma') \in R_\Gamma(p), \forall (a, a') \in R_A(p, [\gamma]_p), (f\gamma a, g\gamma' a') \in R_B(p, [\gamma]_p).$$

As before such a function f is called *non-expansive* if $f \sim_\Gamma f$, i.e., if f is in the domain of $\sim_\Gamma$. Identity at the object A is given by the equivalence class of the second projection and composition of $[f] : A \to B$ and $[g] : B \to C$ is given by the equivalence class of the continuous function

$$\gamma \mapsto a \mapsto g\gamma(f\gamma a).$$

Remark 6.3.3. Comparing to the situation with presheaves again, the morphisms in $\mathsf{Type}_P(\Gamma)$ have access to an additional parameter Γ (as compared to the morphisms in $\mathsf{PEqu}(\int_P \Gamma)$). The reason this is not needed with presheaf categories is that natural transformations are *indexed* families of functions (satisfying coherence conditions) and elements of Γ are part of the indexing poset. So in essence, the additional parameter Γ is already built into the definition of the category of presheaves over the category of elements. ♦

Theorem 6.3.4. *For any poset P and $\Gamma \in \mathsf{PEqu}(P)$ the category $\mathsf{Type}_P(\Gamma)$ is equivalent to the slice category $\mathsf{PEqu}(P)/\Gamma$.* ◊

Proof. We define two functors $F : \mathsf{Type}_P(\Gamma) \to \mathsf{PEqu}(P)/\Gamma$ and G in the converse direction. The functor F maps

[3] We use the notation $\mathsf{Type}_P(\Gamma)$ because this category will be used to interpret dependent types in context Γ.

- the object A to the object $\pi_A : \Gamma \ltimes A \to \Gamma$ where
$$\begin{aligned} |\Gamma \ltimes A| &= |\Gamma| \times |A| \\ R_{\Gamma \ltimes A}(p) &= \Big\{((\gamma, a), (\gamma', a')) \;\Big|\; (\gamma, \gamma') \in R_\Gamma(p) \wedge (a, a') \in R_A(p, [\gamma]_p)\Big\}. \end{aligned}$$
and π_A is the equivalence class of the first projection $|\Gamma| \times |A| \to |\Gamma|$.

- the morphism $[f]$ to the equivalence class of the morphism
$$(\gamma, a) \mapsto (\gamma, f\gamma a)$$
The definition of the equivalence relation $\sim_\Gamma$ is precisely what is needed to show that such a definition is independent of the choice of representative f.

The functor G maps

- an object $[f] : A \to \Gamma$ of the slice category to the object $f^{-1}(A)$ of the category $\mathsf{Type}_P(\Gamma)$ where
$$\begin{aligned} \left|f^{-1}(A)\right| &= |A| \\ R_{f^{-1}(A)}\big(p, [\gamma]_p\big) &= \{(a, a') \;\big|\; (a, a') \in R_A(p) \wedge (f(a), \gamma) \in R_\Gamma(p)\}. \end{aligned}$$

- a morphism $[\alpha] : [f] \to [g]$ to the equivalence class of the continuous function
$$\gamma \mapsto a \mapsto \alpha a$$

Finally we define two natural isomorphisms $\eta : \mathrm{id} \to F \circ G$ and $\varepsilon : \mathrm{id} \to G \circ F$.

$$\begin{tikzcd} A \arrow[rr, "\eta_f"] \arrow[dr, "{[f]}"'] & & \Gamma \ltimes f^{-1}(A) \arrow[dl, "\pi_{f^{-1}(A)}"] \\ & \Gamma & \end{tikzcd}$$

Define η_f to be the equivalence class of the continuous function
$$a \mapsto (f(a), a).$$
Its inverse is the equivalence class of the second projection $\pi_2 : |\Gamma| \times |A| \to |A|$.

The component of ε and A is given by the equivalence class of
$$\gamma \mapsto a \mapsto (\gamma, a).$$
Its inverse is given by the equivalence class of the continuous function
$$\gamma \mapsto (\gamma', a) \mapsto a.$$
Checking that these are well-defined and that they satisfy the claimed properties is straightforward, albeit somewhat lengthy, unpacking of definitions.

𝔔𝔈𝔇

Reindexing

Given a morphism $[f]:\Gamma_1 \to \Gamma_2$ in $\mathsf{PEqu}(P)$ there is a functor $f^*:\mathsf{Type}_P(\Gamma_2)\to\mathsf{Type}_P(\Gamma_1)$ defined by "precomposition". Given an object $A\in\mathsf{Type}_P(\Gamma_2)$ the object $f^*(A)$ is

$$\begin{aligned}|f^*(A)| &= |A|\\ R_{f^*(A)}\big(p,[\gamma]_p\big) &= R_A\big(p,[f(\gamma)]_p\big).\end{aligned}$$

The functor f^* maps a morphism realised by g to the morphism realised by $g\circ f$. More concretely, a morphism $[g]$ is by definition realised by a continuous function $g:|\Gamma_2|\to|A|\to|B|$. The morphism $f^*([g])$ is realised by the function of type $|\Gamma_1|\to|A|\to|B|$ mapping γ and a to $g(f\gamma)a$.

Lemma 6.3.5. *The operation f^* we have defined is well-defined, i.e., independent of the choice of representative f, and it is a functor.* ◊

Moreover this construction shows that we have a functor from the opposite of the category $\mathsf{PEqu}(P)$ to the category of categories which maps Γ to $\mathsf{Type}_P(\Gamma)$ and $[f]$ to the functor f^*. So we may use the Grothendieck construction to get a fibration $p:\mathsf{UFam}(P)\to\mathsf{PEqu}(P)$. Concretely, the objects of the total category $\mathsf{UFam}(P)$ are pairs (Γ,A) where $\Gamma\in\mathsf{PEqu}(P)$ and $A\in\mathsf{Type}_P(\Gamma)$. Morphisms $(\Gamma_1,A_1)\to(\Gamma_2,A_2)$ are pairs $([f],[g])$ where $[f]:\Gamma_1\to\Gamma_2$ is a morphism in $\mathsf{PEqu}(P)$ and $[g]$ is an equivalence class of continuous functions $|\Gamma_1|\to|A_1|\to|A_2|$ with respect to the relation $\sim_{A_1,A_2}$ which relates g and g' if and only if

$$\forall p\in P,\forall(\gamma,\gamma')\in R_{\Gamma_1}(p),\forall(a,a')\in R_{A_1}(p,[\gamma]_p),(g\gamma a,g'\gamma' a')\in R_{A_2}(p,[f(\gamma)]_p).$$

Since the assignment $f\mapsto f^*$ is a functor the fibration p, which simply projects the first components, is a cloven split fibration. The chosen cartesian lifting of $[f]:\Gamma_1\to p(\Gamma_2,A)$ is

$$([f],[\gamma\mapsto a\mapsto a]):(\Gamma_1,f^*(A))\to(\Gamma_2,A)$$

The functor F defined in the proof of Theorem 6.3.4 can be extended to a comprehension category. Indeed, define the functor $\mathcal{P}:\mathsf{UFam}(P)\to\mathsf{PEqu}(P)^{\to}$ as

$$\begin{aligned}\mathcal{P}(\Gamma,A) &= \pi_A\\ \mathcal{P}([f],[g]) &= ([f],[(\gamma,a)\mapsto(f\gamma,g\gamma a)])\end{aligned}$$

as depicted in the following diagram

$$\begin{tikzcd}
\Gamma_1\ltimes A_1 \arrow[r, "{[(\gamma,a)\mapsto(f(\gamma),g\gamma a)]}"] \arrow[d, "\pi_{A_1}"] & \Gamma_2\ltimes A_2 \arrow[d, "\pi_{A_2}"] \\
\Gamma_1 \arrow[r, "{[f]}"] & \Gamma_2
\end{tikzcd}$$

Theorem 6.3.6. *The diagram*

$$\begin{tikzcd}
\mathsf{UFam}(P) \arrow[r, "\mathcal{P}"] \arrow[d, "p"] & \mathsf{PEqu}(P)^{\to} \arrow[dl, "\mathsf{cod}"] \\
\mathsf{PEqu}(P) &
\end{tikzcd}$$

commutes and p is a full split comprehension category with unit. The terminal object functor $1 : \mathsf{PEqu}(P) \to \mathsf{UFam}(P)$ maps Γ to the object $(\Gamma, 1)$ where

$$\begin{aligned} |1| &= 2^{\emptyset} \\ R_1(p) &= 2^{\emptyset} \times 2^{\emptyset} \end{aligned}$$

and $2^{\emptyset}$ is the power set of the empty set. ◊

Below we will extensively use the fact that the fibre over an object Γ with respect to the fibration p is isomorphic in a trivial way to the category $\mathsf{Type}_P(\Gamma)$. This reduces clutter because we do not have to carry around the first components of objects of the fibre which do not matter, since they are uniquely determined (up to equality).

Dependent products

Theorem 6.3.7. *The comprehension category p has (split) products satisfying (split) Beck-Chevalley condition.* ◊

Proof. Let $\pi_A = \mathcal{P}(\Gamma, A) : \Gamma \ltimes A \to \Gamma$ be a projection. It induces a functor $\pi_A^{\maltese}$ from the slice over Γ to the slice over $\Gamma \ltimes A$ which we need to show has a right adjoint. Representing slices using the categories $\mathsf{Type}_P(-)$ the functor $\pi_A^{\maltese}$ is simply π_A^*. Thus, given an object $B \in \mathsf{Type}_P(\Gamma \ltimes A)$ define the object $\Pi(A, B) \in \mathsf{Type}_P(\Gamma)$ to have the underlying lattice $|\Pi(A, B)|$ the lattice of continuous functions $|A| \to |B|$. The family of relations $R_{\Pi(A,B)}$ is defined at $(p, [\gamma]_p)$ to be

$$\left\{(f, f') \,\middle|\, \forall q \leq p, \forall (a, a') \in R_A(q, [\gamma]_q), (f(a), f'(a')) \in R_B\big(q, [(\gamma, a)]_q\big)\right\}.$$

To show that we get a right adjoint to π_A^* we show that we have a universal morphism

$$\mathsf{ap}_{A,B} : \pi_A^*(\Pi(A, B)) \to B$$

in $\mathsf{Type}_P(\Gamma \ltimes A)$. We define $\mathsf{ap}_{A,B}$ to be the equivalence class of the morphism

$$(\gamma, a) \mapsto f \mapsto f\,a.$$

Note that the relation $R_{\Pi(A,B)}$ states precisely what is needed to show that the function we have defined is non-expansive.

To show that $\mathsf{ap}_{A,B}$ is universal we show that for any C and any morphism $[\varphi] : \pi^*_A(C) \to B$ there is a unique morphism $\mathsf{cur}(\varphi) : C \to \Pi(A,B)$ in $\mathsf{Type}_P(\Gamma)$ making the following diagram commute.

$$\begin{tikzcd}
\pi^*_A(C) \arrow[r, "\pi^*_A(\mathsf{cur}(\varphi))"] \arrow[rd, "{[\varphi]}"'] & \pi^*_A(\Pi(A,B)) \arrow[d, "\mathsf{ap}_{A,B}"] \\
 & B
\end{tikzcd}$$

Define $\mathsf{cur}(\varphi)$ to be the equivalence class of the function

$$\gamma \mapsto c \mapsto (a \mapsto \varphi(\gamma,a)c)$$

of type $|\Gamma| \to |C| \to (|A| \to |B|)$, recalling that $[\varphi]$ is realised by the function φ of type $|\Gamma| \times |A| \to |C| \to |B|$.

Checking that all the stated properties hold is straightforward unpacking of definitions.

Finally, the Beck-Chevalley condition can be verified to hold by simple computations. 𝔔𝔈𝔇

Dependent sums

Theorem 6.3.8. *The comprehension category p has (split)* strong *coproducts satisfying (split) Beck-Chevalley condition.* ◊

Proof. As in the proof of Theorem 6.3.7 given an object $B \in \mathsf{Type}_P(\Gamma \ltimes A)$ define an object $\Sigma(A,B) \in \mathsf{Type}_P(\Gamma)$ as follows. The underlying lattice $|\Sigma(A,B)|$ is the lattice $|A| \times |B|$. The family of relations $R_{\Sigma(A,B)}$ is defined at $(p,[\gamma]_p)$ to be

$$\Big\{((a,b),(a',b')) \;\Big|\; (a,a') \in R_A(p,[\gamma]_p), (b,b') \in R_B\big(p,[(\gamma,a)]_p\big)\Big\}.$$

This definition comes with a morphism $\mathsf{pair}_{A,B} : B \to \pi^*_A(\Sigma(A,B))$ in the fibre over $\Gamma \ltimes A$ which we define to be the equivalence class of the continuous function

$$(\gamma,a) \mapsto b \mapsto (a,b).$$

This morphism satisfies the property that for any C and any $[\varphi] : B \to \pi^*_A(C)$ there exists a unique morphism $\mathsf{unpack}(\varphi) : \Sigma(A,B) \to C$ satisfying

$$\pi^*_A(\mathsf{unpack}(\varphi)) \circ \mathsf{pair}_{A,B} = [\varphi].$$

Indeed, the morphism $\mathsf{unpack}(\varphi)$ is defined as the equivalence class of the continuous function

$$\gamma \mapsto (a,b) \mapsto \varphi(\gamma,a)b.$$

All of the claimed properties are easy and straightforward to check.

With these definitions the assignment $B \mapsto \Sigma(A,B)$ extends to a functor left adjoint to π_A^*. Concretely it maps a morphism $[\varphi] : B \to C$ to the equivalence class of the continuous function

$$\gamma \mapsto (a,b) \mapsto (a, \varphi(\gamma,a)b).$$

Recall the domain functor $\mathsf{dom} : \mathsf{PEqu}(P)^{\to} \to \mathsf{PEqu}(P)$. A comprehension category has *strong* coproducts [51, Definition 10.5.2] if the morphism

$$\mathsf{dom}\Big(\mathcal{P}(\pi_A, \mathsf{pair}_{A,B})\Big)$$

is an isomorphism. In our case this follows by computation. Indeed, the morphism $\mathsf{dom}\Big(\mathcal{P}(\pi_A, \mathsf{pair}_{A,B})\Big)$ is the equivalence class of the continuous function

$$((\gamma,a),b) \mapsto (\gamma,(a,b)). \qquad \mathfrak{QED}$$

Equality

Theorem 6.3.9. *The fibration p has* strong *equality.* ◊

Proof. For any Γ and any $A \in \mathsf{Type}_P(\Gamma)$ there is a morphism in $\mathsf{PEqu}(P)$

$$\delta_A : \Gamma \ltimes A \to \Gamma \ltimes A \ltimes \pi_A^*(A)$$

which is the equivalence class of the continuous function $(\gamma,a) \mapsto ((\gamma,a),a)$.

Recall that a comprehension category has weak equality if $\delta_A^{\maltese}$ has a left adjoint Eq_A for any A and these left adjoints satisfy the Beck-Chevalley condition.

Let us define $\mathsf{Eq}_A : \mathsf{Type}_P(\Gamma \ltimes A) \to \mathsf{Type}_P\Big(\Gamma \ltimes A \ltimes \pi_A^*(A)\Big)$. We proceed as we did in Theorem 6.3.8, by constructing an object $\mathsf{Eq}_A(B)$ and a universal morphism. Given $B \in \mathsf{Type}_P(\Gamma \ltimes A)$ define $\mathsf{Eq}_A(B)$ to have the underlying lattice $|\mathsf{Eq}_A B|$ the lattice $|B|$ and the family of relations $R_{\mathsf{Eq}_A(B)}$ is defined at $(p, [((\gamma,a),a')])$ to be

$$\begin{aligned}&\Big\{(b,b') \,\Big|\, (b,b') \in R_B(p,[(\gamma,a)]), (a,a') \in R_A(p,[\gamma])\Big\}\\ =&\begin{cases} R_B(p,[(\gamma,a)]) & \text{if } (a,a') \in R_A(p,[\gamma]) \\ \emptyset & \text{otherwise}\end{cases}\end{aligned}$$

There is a morphism $\mathsf{refl}_{A,B} : B \to \delta_A^*(\mathsf{Eq}_A B)$ which is the equivalence class of the continuous function

$$(\gamma,a) \mapsto b \mapsto b.$$

This pair satisfies the universal property stating that for any C and any $[\varphi] : B \to \delta_A^*(C)$ there exists a unique morphism $\mathsf{with}(\varphi) : \mathsf{Eq}_A(B) \to C$ satisfying $\delta_A^*(\mathsf{with}(\varphi)) \circ \mathsf{refl}_{A,B} = [\varphi]$. The morphism $\mathsf{with}(\varphi)$ is the equivalence class of the continuous function

$$((\gamma, a), a') \mapsto b \mapsto \varphi(\gamma, a)b.$$

To show that the equality is strong we additionally need to check [51, Definition 10.5.2] that the canonical morphism

$$\kappa : \Gamma \ltimes A \ltimes B \to \Gamma \ltimes A \ltimes \pi_A^*(A) \ltimes \mathsf{Eq}_A B$$

which is the equivalence class of the continuous function

$$((\gamma, a), b) \mapsto (((\gamma, a), a), b)$$

is an isomorphism. This is indeed the case and its inverse is given by the equivalence class of the function

$$(((\gamma, a), a'), b) \mapsto ((\gamma, a), b).$$

$\mathfrak{QED}$

With this we have shown that for any poset P the fibration P is a model of dependent type theory with strong dependent sums and strong, i.e. extensional, equality.

A few words on how the identity type is modelled. A term t of type A in context Γ is modelled as a section of the projection $\mathcal{P}(A)$. Because $\mathcal{P}$ is a comprehension category the diagram

$$\begin{tikzcd}
\Gamma \ltimes A \ltimes \pi_A^*(A) \arrow[r] \arrow[d, "\pi_{\pi_A^*(A)}"] & \Gamma \ltimes A \arrow[d, "\pi_A"] \\
\Gamma \ltimes A \arrow[r, "\pi_A"] & \Gamma
\end{tikzcd}$$

which is the image, $\mathcal{P}(\overline{\pi_A})$, of the cartesian lifting $\overline{\pi_A}$ of π_A, is a pullback diagram. Thus given two terms t and s of type A we get a unique morphism $\langle\langle \mathrm{id}, t\rangle, s\rangle$ making the following diagram commute.

$$\begin{tikzcd}
\Gamma \arrow[drr, bend left, "t"] \arrow[dr, dotted, "{\langle\langle \mathrm{id}, t\rangle, s\rangle}"] \arrow[ddr, bend right, "s"'] & & \\
& \Gamma \ltimes A \ltimes \pi_A^*(A) \arrow[r] \arrow[d, "\pi_{\pi_A^*(A)}"] & \Gamma \ltimes A \arrow[d, "\pi_A"] \\
& \Gamma \ltimes A \arrow[r, "\pi_A"] & \Gamma
\end{tikzcd}$$

The fibration p has a terminal object functor 1. We define the interpretation of the type $\mathsf{Id}(t,s)$ to be the object

$$\langle\langle \mathrm{id}, t\rangle, s\rangle^{\maltese}\left(\mathsf{Eq}_A(1(\Gamma \ltimes A))\right).$$

Concretely in our model the underlying lattice of $[\![\mathsf{Id}(t,s)]\!]$ is the lattice $2^{\emptyset}$. The relation $R_{\mathsf{Id}(t,s)}$ is

$$\begin{aligned} R_{\mathsf{Id}(t,s)}(p,[\gamma]) &= \{(\emptyset,\emptyset) \mid (\pi_2(t(\gamma)), \pi_2(s(\gamma))) \in R_A(p,[\gamma])\} \\ &= \begin{cases} |1| \times |1| & \text{if } (\pi_2(t(\gamma), \pi_2(s(\gamma)))) \in R_A(p,[\gamma]) \\ \emptyset & \text{otherwise} \end{cases} \end{aligned}$$

which is as expected. This also makes it clear why the model supports extensional equality.

Changing the underlying poset

Let P and Q be two posets and $\varphi : Q \to P$ a monotone function. It induces a functor $\varphi^{\dagger} : \mathsf{PEqu}(P) \to \mathsf{PEqu}(Q)$ (notice the reversed order of posets Q and P) by "precomposition" as follows. It maps an object Γ to the object $\varphi^{\dagger}(\Gamma)$ where

$$\begin{aligned} \left|\varphi^{\dagger}(\Gamma)\right| &= |\Gamma| \\ R_{\varphi^{\dagger}(\Gamma)} &= R_{\Gamma} \circ \varphi \end{aligned}$$

where we consider R_{Γ} as a function from P to the set of partial equivalence relations on $|\Gamma|$, so composition with φ is well-defined. Moreover because φ is monotone the composition $R_{\Gamma} \circ \varphi$ retains the monotonicity property. The functor $\varphi^{\dagger}$ maps a morphism realised by f to the equivalence class of the continuous function f. Note however that the equivalence relations are different, consequently $\varphi^{\dagger}$ is in general neither full nor faithful.

Lemma 6.3.10. *The functor $\varphi^{\dagger}$ preserves limits and colimits.* ◊

Further, given $\Gamma \in \mathsf{PEqu}(P)$ there is an induced monotone function φ_{Γ}

$$\begin{aligned} \varphi_{\Gamma} : \int_Q \varphi^{\dagger}(\Gamma) &\to \int_P \Gamma \\ \varphi_{\Gamma}(q, [\gamma]_q) &= \left(\varphi(q), [\gamma]_{\varphi(q)}\right). \end{aligned}$$

Note that by the definition of $\varphi^{\dagger}(\Gamma)$ the sets $[\gamma]_{\varphi(q)}$ and $[\gamma]_q$ are equal.

This function in turn induces a functor

$$\varphi^{\dagger}_{\Gamma} : \mathsf{Type}_P(\Gamma) \to \mathsf{Type}_Q\left(\varphi^{\dagger}(\Gamma)\right)$$

which acts as follows.

objects It maps an object A to the object $(|A|, R_A \circ \varphi_\Gamma)$.

morphisms It maps the morphism realised by f to the equivalence class of f, exactly as in the definition of $\varphi^\dagger$.

We have two types of reindexing. The following lemma states that they commute in the appropriate way.

Lemma 6.3.11. *Let $[f] : \Gamma_1 \to \Gamma_2$ be a morphism in $\mathsf{PEqu}(P)$. The following diagram of functors commutes on the nose*

$$\begin{tikzcd}
\mathsf{Type}_P(\Gamma_2) \arrow[r, "f^*"] \arrow[d, "\varphi^\dagger_{\Gamma_2}"'] & \mathsf{Type}_P(\Gamma_1) \arrow[d, "\varphi^\dagger_{\Gamma_1}"] \\
\mathsf{Type}_Q\left(\varphi^\dagger(\Gamma_2)\right) \arrow[r, "(\varphi^\dagger(f))^*"] & \mathsf{Type}_Q\left(\varphi^\dagger(\Gamma_1)\right)
\end{tikzcd}$$

◊

Proof. Since none of these functors changes the underlying lattices or the realisers those are clearly preserved on the nose if only the relations are preserved. This is easily seen to be the case with the definitions provided. 𝔔𝔈𝔇

Combining these two functors we get a morphism of fibrations

$$\begin{tikzcd}
\mathsf{UFam}(P) \arrow[r, "\varphi^\top"] \arrow[d, "p_P"'] & \mathsf{UFam}(Q) \arrow[d, "p_Q"] \\
\mathsf{PEqu}(P) \arrow[r, "\varphi^\dagger"] & \mathsf{PEqu}(Q)
\end{tikzcd} \tag{6.1}$$

The functor $\varphi^\top$ maps

objects the object (Γ, A) to the object $\left(\varphi^\dagger(\Gamma), \varphi^\dagger_\Gamma(A)\right)$

morphisms the morphism $([f], [g])$ to the morphism $([f]', [g]')$. We use $'$ to highlight the fact that the equivalence relations are different.

Theorem 6.3.12. *The pair of functors just defined is a morphism of split fibrations. It maps the chosen cartesian liftings to the chosen cartesian liftings.* ◊

Proof. The fact that it is a morphism of fibrations follows from Lemma 6.3.11 and the fact that the functors never change the underlying realisers. 𝔔𝔈𝔇

Lemma 6.3.13. *Let $\varphi^{\dagger^{\to}} : \mathsf{PEqu}(P)^{\to} \to \mathsf{PEqu}(Q)^{\to}$ be the functor induced by $\varphi^{\dagger}$ in the natural way and let $\mathcal{P}_P$ and $\mathcal{P}_Q$ be the comprehensions belonging to fibrations p_P and p_Q. Then $\varphi^{\dagger^{\to}} \circ \mathcal{P}_P = \mathcal{P}_Q \circ \varphi^{\top}$. In particular for any $A \in \mathsf{Type}_P(\Gamma)$ we have $\varphi^{\dagger}(\pi_A) = \pi_{\varphi^{\dagger}_{\Gamma} A}$.*

Finally, the diagram of functors

$$\begin{tikzcd}
\mathsf{UFam}(P)_{\Gamma} \arrow[r, "\pi_A^{\maltese}"] \arrow[d, "\varphi^{\top}"] & \mathsf{UFam}(P)_{\Gamma \ltimes A} \arrow[d, "\varphi^{\top}"] \\
\mathsf{UFam}(Q)_{\varphi^{\dagger}(\Gamma)} \arrow[r, "\pi_{\varphi^{\dagger}_{\Gamma} A}^{\maltese}"] & \mathsf{UFam}(Q)_{\varphi^{\dagger}(\Gamma) \ltimes \varphi^{\dagger}_{\Gamma}(A)}
\end{tikzcd}$$

commutes on the nose. Note that we have used the equality $\varphi^{\dagger}(\Gamma) \ltimes \varphi^{\dagger}_{\Gamma}(A) = \varphi^{\dagger}(\Gamma \ltimes A)$ which follows from the first part of the lemma. ◊

Theorem 6.3.14. *If φ is a* fibration *then the morphism of fibrations (6.1) preserves products and coproducts on the nose. This means (cf. [51, Definition 1.9.13])*

- *$\varphi^{\dagger}$ preserves pullbacks on the nose*
- *For every $\Gamma \in \mathsf{PEqu}(P)$ and $A \in \mathsf{Type}_P(\Gamma)$ the canonical natural transformation*

  $$\varphi^{\top} \circ \Pi(A, -) \to \Pi\left(\varphi^{\dagger}_{\Gamma}(A), -\right) \circ \varphi^{\top}$$

  is the identity. This in particular means that we have an equality of objects

  $$\varphi^{\top}(\Pi(A, B)) = \Pi\left(\varphi^{\dagger}_{\Gamma}(A), \varphi^{\top}(B)\right)$$

  for every $B \in \mathsf{Type}_P(\Gamma \ltimes A)$.
- *For every $\Gamma \in \mathsf{PEqu}(P)$ and $A \in \mathsf{Type}_P(\Gamma)$ the canonical natural transformation*

  $$\Sigma\left(\varphi^{\dagger}_{\Gamma}(A), -\right) \circ \varphi^{\top} \to \varphi^{\top} \circ \Sigma(A, -)$$

  is the identity.

◊

Proof. Most of this is simple computation and only requires φ to be monotone. We need the assumption that φ is a fibration to show that products are preserved. To show how this assumption is used we spell out the proof of

$$\varphi^{\top}(\Pi(A, B)) = \Pi\left(\varphi^{\dagger}_{\Gamma}(A), \varphi^{\top}(B)\right).$$

Since $\varphi^\top$ or $\varphi^\dagger_\Gamma$ do not change the underlying lattices they are $|A| \to |B|$ on both sides. To show that the families of relations are the same we show two inclusions.

$\subseteq$ For every $q \in Q$ and $(\gamma, \gamma) \in R_\Gamma(\varphi(q))$ we show

$$R_{\varphi^\top(\Pi(A,B))}(q, [\gamma]) \subseteq R_{\Pi\left(\varphi^\dagger_\Gamma(A), \varphi^\top(B)\right)}(q, [\gamma]).$$

Recall

$$R_{\varphi^\top(\Pi(A,B))}(q, [\gamma]) = R_{\Pi(A,B)}(\varphi(q), [\gamma]). \tag{6.2}$$

Let

$$(f, f') \in R_{\Pi(A,B)}(\varphi(q), [\gamma]), \tag{6.3}$$

$r \le q$ and $(a, a') \in R_{\varphi^\dagger_\Gamma(A)}(r, [\gamma])$. Recall

$$R_{\varphi^\dagger_\Gamma(A)}(r, [\gamma]) = R_A(\varphi(r), [\gamma]).$$

Because φ is monotone $\varphi(r) \le \varphi(q)$ holds and so from assumption (6.3) we get $(fa, f'a') \in R_B(\varphi(r), [(\gamma, a)])$ and by definition of $\varphi^\top$ we have

$$R_{\varphi^\top(B)}(r, [(\gamma, a)]) = R_B(\varphi(r), [(\gamma, a)])$$

which gives

$$(fa, f'a') \in R_{\varphi^\top(B)}(r, [(\gamma, a)])$$

as needed. Note that we have only needed φ to be monotone for this direction.

$\supseteq$ For every $q \in Q$ and $(\gamma, \gamma) \in R_\Gamma(\varphi(q))$ we show

$$R_{\Pi\left(\varphi^\dagger_\Gamma(A), \varphi^\top(B)\right)}(q, [\gamma]) \subseteq R_{\varphi^\top(\Pi(A,B))}(q, [\gamma]).$$

Assume

$$(f, f') \in R_{\Pi\left(\varphi^\dagger_\Gamma(A), \varphi^\top(B)\right)}(q, [\gamma]). \tag{6.4}$$

Recalling (6.2) let $r \le \varphi(q)$ and $(a, a') \in R_A(r, [\gamma])$. We need to show $(fa, f'a') \in R_B(r, [(\gamma, a)])$. Because φ is a fibration there exists a $u(q, r) \in Q$ such that $u(q, r) \le q$ and $\varphi(u(q, r)) = r$. Thus by definition we have $(a, a') \in R_{\varphi^\dagger_\Gamma(A)}(u(q, r), [\gamma])$ and so from (6.4)

$$(fa, f'a') \in R_{\varphi^\top(B)}(u(q, r), [(\gamma, a)]).$$

By definition and the property $\varphi(u(q, r)) = r$ we have

$$R_{\varphi^\top(B)}(u(q, r), [(\gamma, a)]) = R_B(r, [(\gamma, a)])$$

which concludes the proof.

$\mathfrak{QED}$

Remark 6.3.15. We have not needed the full assumption that φ is a fibration, only that the function φ when restricted to $\downarrow q \to \downarrow \varphi(q)$ is *surjective* for any $q \in Q$. This is equivalent to requiring that $\varphi : Q \to P$ is an *open* map when Q and P are equipped with the version of the Alexandrov topology where the opens are lower sets. Indeed, this is immediate from the fact that down sets of the form $\downarrow q$ form a basis of the Alexandrov topology. However if we were to consider universes we would likely also need the additional assumption requiring that the assignment $u(q, -)$ (see Definition 5.2.7 on page 186) is a monotone function (cf. Lemma 5.5.12 on page 206). ♦

Modelling guarded dependent type theory

This section uses the definition of posets $\mathrm{I}(\Delta)$ for a finite set Δ and monotone functions $\mathrm{I}(f)$ for $f : \Delta_1 \to \Delta_2$ and their properties from Section 5.2 (page 184) which we do not repeat here. We define $\mathfrak{GR}(\Delta) = \mathsf{PEqu}(\mathrm{I}(\Delta))$ and for a function $f : \Delta_1 \to \Delta_2$ we define $\mathfrak{GR}(f) : \mathfrak{GR}(\Delta_1) \to \mathfrak{GR}(\Delta_2)$ to be the functor $\mathrm{I}(f)^{\dagger}$ defined in the previous section. Similarly we define $\mathfrak{GR}^{\top}(\Delta) = \mathsf{UFam}(\mathrm{I}(\Delta))$ and $\mathfrak{GR}^{\top}(f) : \mathsf{UFam}(\mathrm{I}(\Delta_1)) \to \mathsf{UFam}(\mathrm{I}(\Delta_2))$ to be the functor $\mathrm{I}(f)^{\top}$ defined in the previous section. We write p_{Δ} for the resulting fibration. For an object $\Gamma \in \mathfrak{GR}(\Delta)$ we define $\mathfrak{GR}_{\Gamma}(\Delta) = \mathsf{Type}_{\mathrm{I}(\Delta)}(\Gamma)$. Note that $\mathfrak{GR}_{\Gamma}(\Delta)$ is isomorphic in a trivial way to the fibre over Γ with respect to the fibration p_{Δ}. Because the diagram (6.1) commutes the functor $\mathfrak{GR}^{\top}(f)$, for any $f : \Delta_1 \to \Delta_2$, restricts to a functor $\mathfrak{GR}_{\Gamma}(\Delta_1) \to \mathfrak{GR}_{\mathfrak{GR}(f)(\Gamma)}(\Delta_2)$. We will write $\mathfrak{GR}_{\Gamma}(f)$ for this functor.

Clock quantification

Let Δ be a finite set of clocks, κ a clock not in Δ and $\iota : \Delta \to \Delta, \kappa$ the inclusion. Let $\Gamma \in \mathfrak{GR}(\Delta)$. We define a functor $\forall\kappa : \mathfrak{GR}_{\mathfrak{GR}(\iota)(\Gamma)}(\Delta, \kappa) \to \mathfrak{GR}_{\Gamma}(\Delta)$ right adjoint to the functor $\mathfrak{GR}_{\Gamma}(\iota)$. It maps an object A to the object $\forall\kappa A$ where

$$|\forall\kappa A| = |A|$$
$$R_{\forall\kappa A}\left((E,\delta), [\gamma]_{E,\delta}\right) = \bigcap_{n \in \mathbb{N}} R_A\left(\iota^{!}_{n}(E,\delta), [\gamma]_{\iota^{!}_{n}(E,\delta)}\right)$$

where $\iota^{!}_{n}$ is defined just before Lemma 5.4.2 on page 194. In particular Lemma 5.4.2 shows that $\iota^{!}_{n}$ satisfies $\mathrm{I}(\iota) \circ \iota^{!}_{n} = \mathrm{id}_{\mathrm{I}(\Delta)}$ which ensures that the sets $[\gamma]_{E,\delta}$ and $[\gamma]_{\iota^{!}_{n}(E,\delta)}$ are in fact equal, so the relation $R_{\forall\kappa A}$ is well-defined and because PERs are closed under intersection it is also a PER.

The functor $\forall\kappa$ maps a morphism realised by f to the morphism realised by f. Note however that, again, the equivalence relations with which the morphisms are constructed are not the same at the source and target, so $\forall\kappa$ is neither full nor faithful in general.

Theorem 6.3.16. *The functor $\forall\kappa$ is right adjoint to $\mathfrak{GR}_\Gamma(\iota)$. It additionally satisfies the following properties.*

- $\forall\kappa \circ \mathfrak{GR}_\Gamma(\iota) = id$
- *The unit of the adjunction is the identity.*
- *The counit of the adjunction at an object A is given by the equivalence class of the continuous function*

  $$\gamma \mapsto a \mapsto a.$$

  Note however that it is not the identity.
- *(The Beck-Chevalley condition for clock substitution) Given $u : \Delta_1 \to \Delta_2$ and $\kappa_1 \notin \Delta_1$ and $\kappa_2 \notin \Delta_2$. Let $\hat{u} : \Delta_1, \kappa_1 \to \Delta_2, \kappa_2$ be the extension of u mapping κ_1 to κ_2. Then*

  $$\mathfrak{GR}_\Gamma(u) \circ \forall\kappa_1 = \forall\kappa_2 \circ \mathfrak{GR}_\Gamma(\hat{u})$$

  and moreover the canonical morphism from left to right is the identity.
- *If $[f] : \Gamma_1 \to \Gamma_2$ is a morphism in $\mathfrak{GR}(\Delta)$ then*

  $$f^* \circ \forall\kappa = \forall\kappa \circ (\mathfrak{GR}(\iota)([f]))^*.$$

  This item shows that substitution in terms commutes with clock quantification as it should.

◊

Proof. The first and second items follow from the property $\mathrm{I}(\iota) \circ \iota_n^! = \mathrm{id}_{\mathrm{I}(\Delta)}$.

The third item follows from the property

$$\iota_{\delta(\kappa)}^!(\mathrm{I}(\iota)(E,\delta)) \geq (E,\delta)$$

proved in Lemma 5.4.2.

The last two items follow from the fact that the underlying lattices never change and a simple computation showing the relations are preserved. $\mathfrak{QED}$

The delay functor

Let Δ be a finite set of clocks, $\kappa \in \Delta$ and $\Gamma \in \mathfrak{GR}(\Delta)$. There is a functor $\blacktriangleright_\Gamma^\kappa : \mathfrak{GR}_\Gamma(\Delta) \to \mathfrak{GR}_\Gamma(\Delta)$. It maps the object A to the object $\blacktriangleright_\Gamma^\kappa(A)$ where

$$\left|\blacktriangleright_\Gamma^\kappa(A)\right| = |A|$$

$$R_{\blacktriangleright_\Gamma^\kappa A}((E,\delta),[\gamma]_{E,\delta}) = \begin{cases} |A| \times |A| & \text{if } \delta(\kappa) = 1 \\ R\left((E,\delta^{-\kappa}),[\gamma]_{(E,\delta^{-\kappa})}\right) & \text{otherwise} \end{cases}$$

where $\delta^{-\kappa}$ is defined in Definition 5.3.2 on page 191. Lemma 5.3.3 then shows in particular that $\delta^{-\kappa}$ satisfies

$$(E, \delta^{-\kappa}) \leq (E, \delta)$$

and the assignment is monotone in (E, δ) which shows that $R_{\blacktriangleright^\kappa A}$ is well-defined. The functor $\blacktriangleright^\kappa_\Gamma$ maps a morphism realised by f to the morphism realised by f.

There is a natural transformation $\mathsf{next}^{\Gamma,\kappa} : \mathsf{id} \to \blacktriangleright^\kappa_\Gamma$ which is realised, at an object A, by the continuous function $\gamma \mapsto a \mapsto a$.

Remark 6.3.17. This is the place where monotonicity of the relations R_X is needed. Without it the morphism next would not exist. Consequently we could not state and prove the fixed point property in Proposition 6.3.20 in general. ♦

Theorem 6.3.18. *The functor* $\blacktriangleright^\kappa_\Gamma$ *we have defined satisfies the following properties.*

- $\forall\kappa \circ \blacktriangleright^\kappa_\Gamma = \forall\kappa$
- *If* $u : \Delta_1 \to \Delta_2$ *and* $\kappa \in \Delta_1$ *then*

$$\blacktriangleright^{f(\kappa)}_{\mathfrak{GR}(f)(\Gamma)} \circ \mathfrak{GR}_\Gamma(u) = \mathfrak{GR}_\Gamma(u) \circ \blacktriangleright^\kappa_\Gamma$$

- *If* $[f] : \Gamma_1 \to \Gamma_2$ *is a morphism in* $\mathfrak{GR}(\Delta)$ *then*

$$f^* \circ \blacktriangleright^\kappa_{\Gamma_1} = \blacktriangleright^\kappa_{\Gamma_2} \circ f^*.$$

The last item shows that substitution for term variables commutes over $\blacktriangleright$ *correctly.* ◊

Remark 6.3.19. It can be shown that for or each κ the functor $\blacktriangleright^\kappa : \mathsf{PEqu}(P) \to \mathsf{PEqu}(P)$ (which corresponds to the functor $\blacktriangleright^\kappa_1$) is an applicative functor in the sense of [69]. In [27] the applicative functor rules are generalised using delayed substitutions in order for $\blacktriangleright$ to behave well with respect to dependent products. The functors $\blacktriangleright^\kappa_\Gamma$ can be shown to validate these rules. The proofs are straightforward, albeit notationally heavy. ♦

Fixed points

Let Δ be a finite set of clocks, $\kappa \in \Delta$, $\Gamma \in \mathfrak{GR}(\Delta)$ and $A \in \mathfrak{GR}_\Gamma(\Delta)$.

Proposition 6.3.20. *Let* $[g] : \blacktriangleright^\kappa_\Gamma A \to A$ *be a morphism. There is a unique morphism* $[f] : 1 \to A$ *such that*

$$[g] \circ \mathsf{next}^{\Gamma,\kappa}_A \circ [f] = [f].$$

◊

Proof. Recall that 1 is the terminal object in $\mathfrak{GR}_\Gamma(\Delta)$. The underlying lattice is $2^\emptyset$ and the relations are total relations on this lattice. To show that a fixed point exists we first construct a continuous function $f : |\Gamma| \to |1| \to |A|$ satisfying $g\gamma(f\gamma\emptyset) = f\gamma\emptyset$ for all $\gamma \in |\Gamma|$. This is easy to do using the fact that taking least fixed points is a continuous operation [87, Proposition 3.14], i.e., we define

$$f\gamma\emptyset = \bigvee_{n=0}^{\infty} (g\gamma)^n(\bot).$$

Using this realiser we have $[g] \circ \mathrm{next}^{\Gamma,\kappa} \circ [f] = [f]$, provided that f is non-expansive, i.e., that f is a realiser. We show this now. We need to show that for any $(E,\delta) \in \mathcal{I}(\Delta)$ and any $(\gamma,\gamma') \in R_\Gamma(E,\delta)$ we have

$$(f\gamma\emptyset, f\gamma'\emptyset) \in R_A((E,\delta),[\gamma]_{E,\delta}). \tag{6.5}$$

We proceed by induction on $\delta(\kappa)$. To be more precise, the statement we are proving by induction is that for any $n \in \mathbb{N}$ and for any (E,δ) satisfying $\delta(\kappa) = n$ the property (6.5) holds.

If $\delta(\kappa) = 1$ then by the definition of $R_{\blacktriangleright_\Gamma^\kappa A}((E,\delta),[\gamma]_{E,\delta})$ we have

$$(f\gamma\emptyset, f\gamma'\emptyset) \in R_{\blacktriangleright_\Gamma^\kappa A}((E,\delta),[\gamma]_{E,\delta})$$

and so, because g is non-expansive,

$$(g\gamma(f\gamma\emptyset), g\gamma'(f\gamma'\emptyset)) \in R_A((E,\delta),[\gamma]_{E,\delta})$$

but, e.g., $g\gamma(f\gamma\emptyset) = f\gamma\emptyset$ so we have (6.5). The induction step should now be clear.

Uniqueness of the constructed fixed point follows by an analogous induction. 𝔔𝔈𝔇

Note that the underlying realisers might have many fixed points, but what we have shown is that all of them are equivalent according to the family of PERs given.

Summary of the model

Theorems 6.3.12 and 6.3.14 together with Lemma 5.2.11 on page 189 ensure that the pair $(\mathfrak{GR}^\top(f), \mathfrak{GR}(f))$ is a morphism of split fibrations preserving products and coproducts on the nose.

We can organise the constructions above into the general framework of fibrations. In particular we construct a PDTT-structure [51, Definition 11.3.1].

The model GuardedEqu can be organised in the following diagram of fibrations and comprehension categories.

$$\begin{tikzcd}
\mathbb{D} \arrow[r, "\mathcal{P}"] \arrow[dr, "q"'] & \mathbb{A}^{\to} \arrow[d, "\mathsf{cod}"] & & & \\
& \mathbb{A} \arrow[drr, "r"] & & \mathbb{E} \arrow[r, "\mathcal{Q}"] \arrow[d, "b"] & \mathbb{B}^{\to} \arrow[dl, "\mathsf{cod}"] \\
& & & \mathbb{B} &
\end{tikzcd}$$

We now explain what all of these are. The base category $\mathbb{B}$ and the category $\mathbb{E}$ are the same. They have as objects finite subsets of the set of clocks CV. A morphism $f : \Delta_1 \to \Delta_2$ is a function $\Delta_2 \to \Delta_1$ (note the reversal). So, briefly, we may describe $\mathbb{B}$ as the opposite of the category of finite sets and functions. The functor b is the identity functor and $\mathcal{Q}$ is defined as follows. We assume we have a function $\mathsf{new}(-)$ that given a finite set of clocks returns a clock not already in that set. The comprehension $\mathcal{Q}$ maps a set Δ to the inclusion ι_Δ from Δ to $\Delta, \mathsf{new}(\Delta)$. It maps a function $u : \Delta \to \Delta'$ to the commutative square

$$\begin{tikzcd}
\Delta \arrow[r, "u"] \arrow[d, "\iota_\Delta"] & \Delta' \arrow[d, "\iota_{\Delta'}"] \\
\Delta, \mathsf{new}(\Delta) \arrow[r, "\hat{u}"] & \Delta', \mathsf{new}(\Delta')
\end{tikzcd}$$

where $\hat{u}$ is the extension of u mapping $\mathsf{new}(\Delta)$ to $\mathsf{new}(\Delta')$. Thus we easily see that $\mathsf{cod} \circ \mathcal{Q} = \mathrm{id}_{\mathbb{B}} = b$.

Next, the category $\mathbb{A}$ has as objects pairs (Δ, Γ) where Δ is a finite set of clocks and Γ is an object of $\mathfrak{GR}(\Delta)$. A morphism $(\Delta_1, \Gamma_1) \to (\Delta_2, \Gamma_2)$ is a pair $(u, [f])$ where u is a function $\Delta_2 \to \Delta_1$, i.e., a morphism in $\mathbb{B}$ from Δ_1 to Δ_2, and $[f]$ is a morphism $\Gamma_1 \to \mathfrak{GR}(u)(\Gamma_2)$ in $\mathfrak{GR}(\Delta_1)$. Equivalently, this is the total category of the Grothendieck construction applied to the functor $\mathfrak{GR}$, the evident projection $r : \mathbb{A} \to \mathbb{B}$ is a split fibration.

Finally, the category $\mathbb{D}$ can be briefly described as the total category of the Grothendieck construction applied to the functor $\mathfrak{GR}^\top(-)$. Concretely, its objects are pairs $(\Delta, (\Gamma, A))$ where Δ is a finite set of clocks and $(\Gamma, A) \in \mathfrak{GR}^\top(\Delta)$. A morphism

$$(\Delta_1, (\Gamma_1, A_1)) \to (\Delta_2, (\Gamma_2, A_2))$$

is pair morphisms $(u, ([f], [g]))$ where u is a function $\Delta_2 \to \Delta_1$ and $([f], [g])$ is a morphism in $\mathfrak{GR}^\top(\Delta_1)$ from (Γ_1, A_1) to $\mathfrak{GR}^\top(u)(\Gamma_2, A_2)$. The functor q maps the pair $(\Delta, (\Gamma, A))$ to the pair (Δ, Γ). Thus we have that $r \circ q$ is the projection associated with the Grothendieck construction. Moreover, q is also a split fibration. Indeed, the cartesian lifting of

$$(u,[f]) : (\Delta_1,\Gamma_1) \to q\,(\Delta_2,(\Gamma_2,A_2))$$

is the morphism

$$(u,([f],[g])) : \left(\Delta_1,\left(\Gamma_1,f^*\left(\mathfrak{GR}_{\Gamma_2}(u)(A_2)\right)\right)\right) \to (\Delta_2,(\Gamma_2,A_2))$$

and $g : \Gamma_1 \to A_2 \to A_2$ is the function

$$\gamma \mapsto a \mapsto a.$$

Recall that $p \in \mathrm{I}(\Delta_1)$ and so $\mathrm{I}(u)(p) \in \mathrm{I}(\Delta_2)$ as required of a PDTT-structure. Note that q can also be seen as arising from the Grothendieck construction mapping the object (Δ,Γ) to the category $\mathsf{Type}_{\mathrm{I}(\Delta)}(\Gamma)$.

And at last, the comprehension $\mathcal{P}$ maps the object $(\Delta,(\Gamma,A))$ to the morphism (object of the arrow category) $(\mathrm{id}_\Delta,\pi_A)$. Hence we see immediately that all the projections are indeed r-vertical as required.

6.4 Continuity

Let $A,B \in \mathfrak{GR}(\emptyset)$ and $[f] : B \to A$ a morphism. Such a morphism defines a polynomial functor $\mathfrak{p}_f$. Abusing notation this functor can be described as

$$\mathfrak{p}_f(X) = \sum_{a:A} X^{f^{-1}(a)}.$$

or more precisely, $\mathfrak{p}_f(X)$ is the total space, i.e., domain, of the exponential π_2^f in the slice over A, where $\pi_2 : X \times A \to A$ is the second projection.

As an example we have the object of streams whose elements are of type A. We take the morphism $[f] : A \to A$ to be the identity. We then have

$$\mathfrak{p}_{\mathrm{id}_A}(X) = \sum_{a:A} X^1 \cong \sum_{a:A} X = A \times X.$$

Alternatively, without abusing notation, recall that the identity on A is the terminal object in the slice category over A. Hence $\pi_2^{\mathrm{id}_A} \cong \pi_2$ as object of the slice over A and the total space of π_2 is precisely $X \times A$. The object of streams is defined to be the final coalgebra of $\mathfrak{p}_{\mathrm{id}_A}$.

A more interesting example is the type of possibly infinite lists of type A. We take the morphism $[f]$ to be the inclusion of A to $1+A$. Then, abusing notation,

$$\mathfrak{p}_f(X) \cong \sum_{x:1} X^{f^{-1}(x)} + \sum_{x:A} X^{f^{-1}(x)} \cong \sum_{x:1} X^0 + \sum_{x:A} X^1 \cong 1 + A \times X$$

so $\mathfrak{P}_f$ indeed describes the shapes of lists. The object of lists is defined to be the initial algebra, if it exists, of this functor and the object of potentially infinite lists is defined to be the final coalgebra of this functor.

Bauer and Birkedal [13] describe the polynomial functor $\mathfrak{P}_f : \mathfrak{GR}(\emptyset) \to \mathfrak{GR}(\emptyset)$ very concretely as follows. It maps an object X to the object

$$\left(|A| \times (|B| \to |X|), R_{\mathfrak{P}_f(X)}\right)$$

where $(a,u) \approx^{\star}_{\mathfrak{P}_f(X)} (a',u')$ if and only if $(a,a') \in R_A(\star)$ and

$$\forall (b,b') \in R_B(\star), (f(b),a) \in R_A(\star) \to (u(b), u'(b')) \in R_X(\star).$$

We have used $\star$ to denote the unique element of $\mathfrak{I}(\emptyset)$. This functor can be lifted to the functor $\mathfrak{P}^{\kappa}_f$ on $\mathfrak{GR}(\kappa)$ for any clock κ by replacing f by $\mathfrak{GR}(\iota)(f)$. Concretely the functor maps the object X of $\mathfrak{GR}(\kappa)$ to the object whose underlying lattice is the same $|A| \times (|B| \to |X|)$. To describe the relations we will use the fact that there is only one equivalence relation on the set $\{\kappa\}$ to represent pairs $(E,\delta) \in \mathfrak{I}(\kappa)$ by a single natural number n. The relation $R_{\mathfrak{P}^{\kappa}_f(X)}$ at n relates (a,u) and (a',u') if and only if $(a,a') \in R_A(\star)$ and

$$\forall (b,b') \in R_B(\star), (f(b),a) \in R_A(\star) \to (u(b), u'(b')) \in R_X(n)$$

The lifting is easily seen to satisfy the property

$$\mathfrak{GR}(\iota) \circ \mathfrak{P}_f = \mathfrak{P}^{\kappa}_f \circ \mathfrak{GR}(\iota).$$

Next, Bauer and Birkedal [13] construct the algebraic lattice

$$M = \prod_{i=0}^{\infty} (|B|^i \to |A|)$$

together with an isomorphism $\varphi : |A| \times (|B| \to M) \to M$ (in AlgLat) defined component-wise (since M is a product, this makes sense). We use $\pi_i : M \to (|B|^i \to |A|)$ to denote projections.

$$\begin{aligned}\pi_0(\varphi(a,u)) &= \star \mapsto a\\ \pi_{i+1}(\varphi(a,u)) &= (b,\vec{b}) \in |B|^{i+1} \mapsto \pi_i(u(b))(\vec{b})\end{aligned}$$

where we have assumed in this case that products associate to the right.

The inverse $\psi : M \to |A| \times (|B| \to M)$ to φ can also be given explicitly as

$$\psi(m) = \left(\pi_0(m)(\star), b \mapsto \left\{\vec{b} \mapsto \pi_{i+1}(m)(b,\vec{b})\right\}_{i=0}^{\infty}\right).$$

We will write $\psi_1 = \pi_1 \circ \psi : M \to |A|$ and $\psi_2 : \pi_2 \circ \psi : M \to (|B| \to M)$.

Next we construct the solution to $\mathfrak{P}^{\kappa}_f(\blacktriangleright^{\kappa} X) \cong X$. The underlying lattice of X is the lattice M constructed above. The family of relations R_X is defined by induction on n using the isomorphisms φ and ψ. We define $m \approx_X^n m'$ if and only if $(\psi_1(m), \psi_1(m')) \in R_A(\star)$ and

$$\forall (b,b') \in R_B(\star), (f(b), \psi_1(m)) \in R_A(\star) \to (\psi_2(m)(b), \psi_2(m')(b')) \in \bigcap_{k<n} R_X(k)$$

or alternatively

$$R_X(1) = \{(m,m') \,\big|\, (\psi_1(m), \psi_1(m')) \in R_A(\star)\}$$

and $m \approx_X^{n+1} m'$ if and only if $(\psi_1(m), \psi_1(m')) \in R_A(\star)$ and

$$\forall (b,b') \in R_B(\star), (f(b), \psi_1(m)) \in R_A(\star) \to (\psi_2(m)(b), \psi_2(m')(b')) \in R_X(n).$$

With these definitions it is easy to see that the functions φ and ψ are non-expansive, and so they give rise to an isomorphism in $\mathfrak{GR}(\kappa)$.

Finally, applying $\forall\kappa$ to the object X we get the relation $R_M = \bigcap_{n=1}^{\infty} R_X(n)$. Observe that the construction of R_X is precisely the chain $\Phi^n(\top)$ where Φ is the operator defined by Bauer and Birkedal [13]. Because Φ commutes with non-empty intersections, i.e.,

$$\Phi\left(\bigcap_{i\in I} R_i\right) = \bigcap_{i\in I} \Phi(R_i)$$

for all inhabited I, we have that R_M defined above is precisely the largest fixed point of Φ by Kleene's fixed point theorem.

Hence (M, R_M) is the final coalgebra of $\mathfrak{P}_f$, as needed.

To recap, what we have shown is that we can construct final coalgebras of polynomial functors using solutions of guarded domain equations.

Finally, we get more than from the presheaf models [28, 71]. In these models M would just be a set with the property that it is the final coalgebra of the functor $\mathfrak{P}_f$ and we get no useful information on functions $M \to M$ definable in the type theory, i.e., in the model functions $M \to M$ are all functions. Using GuardedEqu we get the additional property that every morphism of type $M \to M$ definable in the type theory is realised by a continuous function.

Let us look at a concrete example to see that this property gives useful information. Let A be some *set* considered as an algebraic lattice $A_{\top,\bot}$ by adding two elements $\top$ and $\bot$ with $\top$ being the top element and $\bot$ the bottom and otherwise the order is discrete. We consider A as an object of $\mathfrak{GR}(\emptyset)$ by equipping $A_{\top,\bot}$ with the PER R_A which is the identity relation on A, but *does not* relate $\bot$ or $\top$ to anything, including themselves.

The type of streams is given as the final coalgebra of the polynomial functor $\mathfrak{P}_{\mathrm{id}_{A_{\top,\bot}}}$. This is not a very convenient description so we provide another description of the type of guarded streams of type A. The underlying lattice is the product lattice $\prod_{i=0}^{\infty} A_{\top,\bot}$ and the relations are

$$R_{\mathrm{Str}^{\mathrm{g}}(A)}(n) = \left\{ \left(\{x_i\}_{i=0}^{\infty}, \{y_i\}_{i=0}^{\infty} \right) \,\middle|\, \forall k < n, (x_k, y_k) \in R_A \right\}$$

Note that n starts at 1, but the indexing of the product starts at 0, which is the reason for using the strict order relation $k < n$.

There are two continuous functions $\alpha : \prod_{i=0}^{\infty} A_{\top,\bot} \to M$ and β in the converse direction, where M is the lattice constructed above, but specialised for the functor $\mathfrak{P}_{\mathrm{id}_{A_{\top,\bot}}}$. So

$$M = \prod_{i=0}^{\infty} \left(A^{i}_{\top,\bot} \to A_{\top,\bot} \right).$$

The function α is defined as

$$\alpha(s) = \{\_ \mapsto s_i\}_{i=0}^{\infty}$$

where s_i is the i-th element of the string. The function β is defined by induction. Given $m \in M$ define the stream $\beta(m)$ by induction

$$\begin{aligned} \beta(m)_0 &= \psi_1(m) \\ \beta(m)_{n+1} &= \psi_1\left(\psi_2(m)(\beta(m)_n)\right). \end{aligned}$$

Then it is easy to see that these functions are continuous and that $\beta \circ \alpha = \mathrm{id}$. In contrast $\alpha \circ \beta$ is not the identity. However an easy calculation shows α and β are non-expansive with respect to the equivalence relations given and that $\alpha \circ \beta \sim \mathrm{id}$.

Thus because $\forall\kappa$ is a functor the lattice $\prod_{i=0}^{\infty} A_{\top,\bot}$ together with the partial equivalence relation relating only equal streams where none of the elements are $\top$ or $\bot$ is the final coalgebra for the functor $\mathfrak{P}_{\mathrm{id}_{A_{\top,\bot}}}$.

Using the equivalence between partial equilogical spaces and equilogical spaces and using the fact that Top is a full subcategory of the category of equilogical spaces [14] we quickly see that functions on streams are in bijective correspondence with continuous functions on the topological space $\prod_{i=0}^{\infty} A$, where A is equipped with the discrete topology. Standard topological exercise then shows that these are precisely the functions with the property that the first m elements of an output stream only depend on the first n, for some n, elements of the input stream.

6.5 Discussion

A natural question to ask is what is the relationship between PEqu(P) and presheaves on P valued in PEqu. Clearly if P is the singleton poset these are equivalent. In general, one can show that the category of PEqu-valued presheaves $[P^{\mathrm{op}}, \mathsf{PEqu}]$ is a reflective subcategory of $\mathsf{PEqu}(P)$ and the reflector F is faithful, but in most cases it is not full. In particular if Δ is inhabited and P is $\mathbb{I}(\Delta)$ the reflector is not full. Based on this we *conjecture* that the categories are not equivalent, however we do not have a proof of this. However even if they were equivalent, the presentation with families of PERs is simpler to work with and does not have problems with coherence inherent in the presheaf presentation.

An inspection of the constructions used shows that the use of algebraic lattices as realisers is not essential. For instance it could be replaced by other categories of domains (such as complete pointed partial orders, Scott domains, or countably based algebraic lattices) or we could consider only PERs on a reflexive domain. The important properties are that the category is cartesian closed and that its endomorphisms have fixed points. Section 6.4 requires more, namely the existence of certain countable limits.

One can also model guarded recursive functions using (complete) ordered families of equivalences [44]. An important difference to that approach is that we require no completeness conditions on our families of PERs. The reason is precisely that the underlying category of realisers has fixed points. In contrast, functions between sets do not necessarily have fixed points, so the completeness conditions on ordered families of equivalences in [44] are needed to ensure that suitably contractive functions have fixed points.

Guarded dependent type theory can be thought of in particular as a rich "rule format" for defining functions on coinductive types, *cf.* the work of Rutten [83] who defines a rule format for defining non-expansive stream functions. Since guarded dependent type theory is an extension of dependent type theory with types which allow us to express "non-expansiveness", the rule format is modular in the sense of [70]. Indeed in *loc. cit.* it is shown that, if one defines a function, e.g., from streams to streams using a restricted set of rules then the set of rules allowed in the construction of stream functions can be extended with the newly defined function. This corresponds to the observation that the newly defined function is non-expansive, which in guarded dependent type theory corresponds to a function from *guarded* streams to *guarded* streams. Using clock quantifiers affords more expressiveness and allows us to distinguish in the type theory between functions which can be used in recursive definitions of streams (and stream functions) and functions such as the tail function which can only be used in a much more restricted way. The model constructed in this paper shows that guarded dependent type theory can also be seen as a rule format for defining continous functions on streams and, more generally, on a large class of coinductive types.

Acknowledgement

This research was supported in part by the ModuRes Sapere Aude Advanced Grant from The Danish Council for Independent Research for the Natural Sciences (FNU). Aleš Bizjak was supported in part by a Microsoft Research PhD grant.

Chapter 7

Guarded Dependent Type Theory with Coinductive Types

This chapter is a version of

> Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus Ejlers Møgelberg, and Lars Birkedal.
>
> Guarded dependent type theory with coinductive types.
>
> In *Foundations of Software Science and Computation Structures - 19th International Conference, FoSSaCS 2016*, pages ??–??, 2016.
>
> To appear (Accepted for publication)

extended with an appendix which provides more detailed derivations of examples, showing how the rules of the calculus are used.

Abstract

We present guarded dependent type theory, called gDTT, an extensional dependent type theory with a 'later' modality and clock quantifiers for programming and proving with guarded recursive and coinductive types. The later modality is used to ensure the productivity of recursive definitions in a modular, type based, way. Clock quantifiers are used for controlled elimination of the later modality and for encoding coinductive types using guarded recursive types. Key to the development of gDTT are novel type and term formers involving what we call 'delayed substitutions'. These generalise the applicative functor rules for the later modality considered in earlier work, and are crucial for programming and proving with dependent types. We show soundness of the type theory with respect to a denotational model.

7.1 Introduction

Dependent type theory is useful both for programming, and for proving properties of elements of types. Modern implementations of dependent type theories such as Coq [68], Nuprl [33], Agda [73], and Idris [29], have been used successfully in many projects. However, they offer limited support for programming and proving with *coinductive* types.

One of the key challenges is to ensure that functions on coinductive types are well-defined; that is, equations describing the function have a unique solution. Syntactic guarded recursion [34], as used for example in Coq [45], ensures productivity by requiring that recursive calls be nested directly under a constructor, but it is well known that such syntactic checks exclude many valid definitions, particularly in the presence of higher-order functions.

To address this challenge, a *type-based* approach to guarded recursion, more flexible than syntactic checks, was first suggested by Nakano [72]. A new modality, written ▹ and called 'later' [9], allows us to distinguish between data we have access to now, and data which we will get later. This modality must be used to guard self-reference in type definitions, so for example *guarded streams* of natural numbers are described by the guarded recursive equation

$$\mathrm{Str}^g_{\mathbb{N}} \simeq \mathbb{N} \times \triangleright \mathrm{Str}^g_{\mathbb{N}}$$

asserting that stream heads are available now, but tails only later.

Types defined via guarded recursion with ▹ are not standard coinductive types, as their denotation is defined via models based on the *topos of trees* [22]. More pragmatically, the bare addition of ▹ disallows productive but *acausal* [57] functions such as the 'every other' function that returns every second element of a stream. Atkey and McBride proposed *clock quantifiers* [11] for such functions; these have consequently been extended to dependent types [28, 71], and Møgelberg [71, Theorem 2] has shown that they allow the definition of types whose denotation is precisely that of standard coinductive types interpreted in set-based semantics. As such, they allow us to program with real coinductive types, while retaining productivity guarantees.

In this paper we introduce the extensional guarded dependent type theory gDTT, which provides a framework where guarded recursion can be used not just for programming with coinductive types but also for coinductive reasoning.

As types depend on terms, one of the key challenges in designing gDTT is coping with elements that are only available later, i.e., elements of types of the form ▹A. We do this by generalising the applicative functor structure of ▹ to the dependent setting. Recall the rules for applicative functors [69]:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathrm{next}\, t : \triangleright A} \qquad \frac{\Gamma \vdash f : \triangleright(A \to B) \qquad \Gamma \vdash t : \triangleright A}{\Gamma \vdash f \circledast t : \triangleright B} \tag{7.1}$$

The first rule allows us to make later use of data that we have now. The second allows, for example, functions to be applied recursively to the tails of streams.

Suppose now that f has type $\triangleright(\Pi x : A.B)$, and t has type $\triangleright A$. What should the type of $f \circledast t$ be? Intuitively, t will eventually reduce to some value $\mathrm{next}\, u$, and so the resulting type should be $\triangleright(B[u/x])$, but if t is an open term we may not be able to perform this reduction. This problem occurs in coinductive reasoning: if, e.g., A is $\mathrm{Str}^g_{\mathbb{N}}$, and B a property of streams, in our applications f will be a (guarded) coinduction assumption that we will want to apply to the tail of a stream, which has type $\triangleright \mathrm{Str}^g_{\mathbb{N}}$.

We hence must introduce a new notion, of *delayed substitution*, similar to let-binding, allowing us to give $f \circledast t$ the type

$$\triangleright [x \leftarrow t].B$$

binding x in B. Definitional equality rules then allow us to simplify this type when t has form $\mathrm{next}\, u$, i.e., $\triangleright [x \leftarrow \mathrm{next}\, u].B \equiv \triangleright(B[u/x])$. This construction generalises to bind a list of variables. Delayed substitution is essential to many examples, as shown in Section 7.3, and surprisingly the applicative functor term-former $\circledast$, so central to the standard presentation of applicative functors, turns out to be *definable* via delayed substitutions, as shown in Section 7.2.

Contributions. The contributions of this paper are:

- We introduce the extensional guarded dependent type theory gDTT, and show that it gives a framework for programming and proving with guarded recursive and coinductive types. The key novel feature is the generalisation of the 'later' type-former and 'next' term-former via *delayed substitutions*;
- We prove the soundness of gDTT via a model similar to that used in earlier work on guarded recursive types and clock quantifiers [28, 71].

We focus on the design and soundness of the type theory and restrict attention to an extensional type theory. We postpone a treatment of an intensional version of the theory to future work (see Secs. 7.7 and 7.8).

In addition to the examples included in this paper, we are pleased to note that a preliminary version of gDTT has already proved crucial for formalizing a logical relations adequacy proof of a semantics for PCF using guarded recursive types by Paviotti et. al. [74].

7.2 Guarded Dependent Type Theory

gDTT is a type theory with base types unit $\mathbf{1}$, booleans $\mathbf{B}$, and natural numbers $\mathbf{N}$, along with Π-types, Σ-types, identity types, and universes. For space reasons we omit all definitions that are standard to such a type theory; see e.g. Jacobs [51]. Our universes are à la Tarski, so we distinguish between types and terms, and have terms that represent types; they are called *codes* of types and they can be recognised by their circumflex, e.g., $\widehat{\mathbf{N}}$ is the code of the type $\mathbf{N}$. We have a map El sending codes of types to their corresponding type. We follow standard practice and often omit El in examples, except where it is important to avoid confusion.

We fix a countable set of *clock variables* CV $= \{\kappa_1, \kappa_2, \cdots\}$ and a single *clock constant* κ_0, which will be necessary to define, for example, the function hd in Section 7.5. A *clock* is either a clock variable or the clock constant; they are intuitively temporal dimensions on which types may depend. A *clock context* $\Delta, \Delta', \cdots$ is a finite *set* of *clock variables*. We use the judgement $\vdash_\Delta \kappa$ to express that either κ is a clock variable in the set Δ or κ is the clock constant κ_0. All judgements, summarised in Figure 7.1, are parametrised by clock contexts. Codes of types inhabit *universes* $\mathcal{U}_\Delta$ parametrised by clock contexts similarly. The universe $\mathcal{U}_\Delta$ is only well-formed in clock contexts Δ' where $\Delta \subseteq \Delta'$. Intuitively, $\mathcal{U}_\Delta$ contains codes of types that can vary only along dimensions in Δ. We have *universe inclusions* from $\mathcal{U}_\Delta$ to $\mathcal{U}_{\Delta'}$ whenever $\Delta \subseteq \Delta'$; in the examples we will not write these explicitly. Note that we do not have $\widehat{\mathcal{U}_\Delta} : \mathcal{U}_{\Delta'}$, i.e., these universes do not form a hierarchy. We could additionally have an orthogonal hierarchy of universes, i.e. for each clock context Δ a hierarchy of universes $\mathcal{U}^1_\Delta : \mathcal{U}^2_\Delta : \cdots$.

All judgements are closed under clock weakening and clock substitution. The former means that if, e.g., $\Gamma \vdash_\Delta t : A$ is derivable then, for any clock variable $\kappa \notin \Delta$, the judgement $\Gamma \vdash_{\Delta,\kappa} t : A$ is also derivable. The latter means that if, e.g., $\Gamma \vdash_{\Delta,\kappa} t : A$ is derivable and $\vdash_\Delta \kappa'$ then the judgement $\Gamma[\kappa'/\kappa] \vdash_\Delta t[\kappa'/\kappa] : A[\kappa'/\kappa]$ is also derivable, where clock substitution $[\kappa'/\kappa]$ is defined as obvious.

The rules for guarded recursion can be found in Figs. 7.2 and 7.3; rules for coinductive types are postponed until Section 7.4. Recall the 'later' type former $\triangleright$, which expresses that something will be available at a later time. In gDTT we have $\overset{\kappa}{\triangleright}$ for each clock κ, so we can delay a type along different dimensions. As discussed in the introduction, we generalise the applicative functor structure of each $\overset{\kappa}{\triangleright}$ via *delayed substitutions*, which allow a substitu-

$\vdash_\Delta \kappa$	valid clock	$\Gamma \vdash_\Delta A \equiv B$	type equality
$\Gamma \vdash_\Delta$	well-formed context	$\Gamma \vdash_\Delta t \equiv u : A$	term equality
$\Gamma \vdash_\Delta A$ type	well-formed type	$\vdash_\Delta \xi : \Gamma \overset{\kappa}{\to} \Gamma'$	delayed substitution
$\Gamma \vdash_\Delta t : A$	typing judgment		

Figure 7.1: Judgements in gDTT.

tion to be delayed until its substituent is available. We showed in the introduction how a type with a single delayed substitution $\overset{\kappa}{\triangleright}[x \leftarrow t].A$ should work. However if we have a term f with more than one argument, for example of type $\overset{\kappa}{\triangleright}(\Pi(x:A).\Pi(y:B).C)$, and wish to type an application $f \textcircled{\kappa} t \textcircled{\kappa} u$ (where $\textcircled{\kappa}$ is the applicative functor operation $\circledast$ for clock κ) we may have neither t nor u available now, and so we need sequences of delayed substitutions to define the type $\overset{\kappa}{\triangleright}[x \leftarrow t, y \leftarrow u].C$. Our concrete examples of Section 7.3 will show that this issue arises in practice. We therefore define sequences of delayed substitutions ξ. The new raw types, terms, and delayed substitutions of gDTT are given by the grammar

$$A, B ::= \cdots \mid \overset{\kappa}{\triangleright}\xi.A \qquad t, u ::= \cdots \mid \mathsf{next}^{\kappa}\,\xi.t \mid \widehat{\triangleright}^{\kappa} t \qquad \xi ::= \cdot \mid \xi\,[x \leftarrow t].$$

Note that we just write $\overset{\kappa}{\triangleright}A$ where its delayed substitution is the empty $\cdot$, and that $\overset{\kappa}{\triangleright}\xi.A$ binds the variables substituted for by ξ in A, and similarly for next.

The three rules DS-Emp, DS-Cons, and Tf-$\triangleright$ are used to construct the type $\overset{\kappa}{\triangleright}\xi.A$. These rules formulate how to generalise these types to arbitrarily long delayed substitutions. Once the type formation rule is established, the introduction rule Ty-Next is the natural one.

With delayed substitutions we can *define* $\textcircled{\kappa}$ as

$$f \textcircled{\kappa} t \triangleq \mathsf{next}^{\kappa}\begin{bmatrix} g \leftarrow f \\ x \leftarrow t \end{bmatrix}.g\,x.$$

Using the rules in Figure 7.2 we can derive the following typing judgement for $\textcircled{\kappa}$

$$\frac{\Gamma \vdash_{\Delta} f : \overset{\kappa}{\triangleright}\xi.\Pi(x:A).B \qquad \Gamma \vdash_{\Delta} t : \overset{\kappa}{\triangleright}\xi.A}{\Gamma \vdash_{\Delta} f \textcircled{\kappa} t : \overset{\kappa}{\triangleright}\xi[x \leftarrow t].B}\ \textsc{Ty-}\circledast$$

When a term has the form $\mathsf{next}^{\kappa}\,\xi\,[x \leftarrow \mathsf{next}^{\kappa}\,\xi.u].t$, then we have enough information to perform the substitution in both the term and its type. The rule TmEq-Force applies the substitution by equating the term with the result of an actual substitution, $\mathsf{next}^{\kappa}\,\xi.t[u/x]$. The rule TyEq-Force does the same for its type. Using TmEq-Force we can derive the basic term equality

$$(\mathsf{next}^{\kappa}\,\xi.f) \textcircled{\kappa} (\mathsf{next}^{\kappa}\,\xi.t) \equiv \mathsf{next}^{\kappa}\,\xi.(f\,t).$$

typical of applicative functors [69].

It will often be the case that a delayed substitution is unnecessary, because the variable to be substituted for does not occur free in the type/term. This is what TyEq-$\triangleright$-Weak and TmEq-Next-Weak express, and with these we can justify the simpler typing rule

$$\frac{\Gamma \vdash_{\Delta} f : \overset{\kappa}{\triangleright}\xi.(A \to B) \qquad \Gamma \vdash_{\Delta} t : \overset{\kappa}{\triangleright}\xi.A}{\Gamma \vdash_{\Delta} f \textcircled{\kappa} t : \overset{\kappa}{\triangleright}\xi.B}$$

Universes

$$\frac{\Delta' \subseteq \Delta \qquad \Gamma \vdash_{\Delta}}{\Gamma \vdash_{\Delta} \mathcal{U}_{\Delta'}\ \text{type}}\ \textsc{Univ} \qquad\qquad \frac{\Gamma \vdash_{\Delta} A : \mathcal{U}_{\Delta'}}{\Gamma \vdash_{\Delta} \mathrm{El}(A)\ \text{type}}\ \textsc{El}$$

Delayed substitutions:

$$\frac{\Gamma \vdash_{\Delta} \qquad \vdash_{\Delta} \kappa}{\vdash_{\Delta} \cdot : \Gamma \overset{\kappa}{\rightarrowtriangle} \cdot}\ \text{DS-}\textsc{Emp} \qquad\qquad \frac{\vdash_{\Delta} \xi : \Gamma \overset{\kappa}{\rightarrowtriangle} \Gamma' \qquad \Gamma \vdash_{\Delta} t : \overset{\kappa}{\rhd}\xi.A}{\vdash_{\Delta} \xi\,[x \leftarrow t] : \Gamma \overset{\kappa}{\rightarrowtriangle} \Gamma', x : A}\ \text{DS-}\textsc{Cons}$$

Typing rules:

$$\frac{\Gamma, \Gamma' \vdash_{\Delta} A\ \text{type} \qquad \vdash_{\Delta} \xi : \Gamma \overset{\kappa}{\rightarrowtriangle} \Gamma'}{\Gamma \vdash_{\Delta} \overset{\kappa}{\rhd}\xi.A\ \text{type}}\ \textsc{Tf-}\rhd \qquad\qquad \frac{\vdash_{\Delta'} \kappa \qquad \Gamma \vdash_{\Delta} A : \overset{\kappa}{\rhd}\mathcal{U}_{\Delta'}}{\Gamma \vdash_{\Delta} \widehat{\rhd}^{\kappa} A : \mathcal{U}_{\Delta'}}\ \textsc{Ty-}\widehat{\rhd}$$

$$\frac{\Gamma, \Gamma' \vdash_{\Delta} t : A \qquad \vdash_{\Delta} \xi : \Gamma \overset{\kappa}{\rightarrowtriangle} \Gamma'}{\Gamma \vdash_{\Delta} \mathrm{next}^{\kappa}\, \xi.t : \overset{\kappa}{\rhd}\xi.A}\ \textsc{Ty-Next} \qquad\qquad \frac{\vdash_{\Delta} \kappa \qquad \Gamma, x : \overset{\kappa}{\rhd}A \vdash_{\Delta} t : A}{\Gamma \vdash_{\Delta} \mathrm{fix}^{\kappa}\, x.t : A}\ \textsc{Ty-Fix}$$

Figure 7.2: Overview of the new typing rules involving $\rhd$ and delayed substitutions.

In other words, delayed substitutions on the type are not necessary when we apply a non-dependent function.

Further, we have the applicative functor identity law

$$(\mathrm{next}^{\kappa}\, \xi.\lambda x.x) \circledast_{\kappa} t \equiv t.$$

This follows from the rule TmEq-Next-Var, which allows us to simplify a term $\mathrm{next}^{\kappa}\, \xi\,[y \leftarrow t].y$ to t.

Sometimes it is necessary to switch the order in the delayed substitution. Two substitutions can switch places, as long as they do not depend on each other; this is what TyEq-$\rhd$-Exch and TmEq-Next-Exch express.

Rule TmEq-Next-Comm is not used in the examples of this paper, but it implies the rule $\mathrm{next}^{\kappa}\, \xi\,[x \leftarrow t].\,\mathrm{next}^{\kappa}\, x \equiv \mathrm{next}^{\kappa}\, t$, which is needed in Paviotti's PhD work.

Fixed points and guarded recursive types

In gDTT we have for each clock κ valid in the current clock context a fixed-point combinator fix^{κ}. This differs from a traditional fixed-point combinator in that the type of the recursion variable is not the same as the result type; instead its type is *guarded* with $\overset{\kappa}{\rhd}$. When we define a term using the fixed-point,

Definitional type equalities:

$$\begin{aligned}
\overset{\kappa}{\triangleright}\xi\,[x \leftarrow t].A &\equiv \overset{\kappa}{\triangleright}\xi.A && \text{(TyEq-}\triangleright\text{-Weak)}\\
\overset{\kappa}{\triangleright}\xi\,[x \leftarrow t, y \leftarrow u]\,\xi'.A &\equiv \overset{\kappa}{\triangleright}\xi\,[y \leftarrow u, x \leftarrow t]\,\xi'.A && \text{(TyEq-}\triangleright\text{-Exch)}\\
\overset{\kappa}{\triangleright}\xi\,[x \leftarrow \mathrm{next}^{\kappa}\,\xi.t].A &\equiv \overset{\kappa}{\triangleright}\xi.A[t/x] && \text{(TyEq-Force)}\\
\mathrm{El}(\widehat{\triangleright}^{\kappa}\,(\mathrm{next}^{\kappa}\,\xi.t)) &\equiv \overset{\kappa}{\triangleright}\xi.\,\mathrm{El}(t) && \text{(TyEq-El-}\triangleright\text{)}\\
\mathrm{Id}_{\overset{\kappa}{\triangleright}\xi.A}(\mathrm{next}^{\kappa}\,\xi.t, \mathrm{next}^{\kappa}\,\xi.s) &\equiv \overset{\kappa}{\triangleright}\xi.\mathrm{Id}_A(t,s) && \text{(TyEq-}\triangleright\text{)}
\end{aligned}$$

Definitional term equalities:

$$\begin{aligned}
\mathrm{next}^{\kappa}\,\xi\,[x \leftarrow t].u &\equiv \mathrm{next}^{\kappa}\,\xi.u && \text{(TmEq-Next-Weak)}\\
\mathrm{next}^{\kappa}\,\xi\,[x \leftarrow t].x &\equiv t && \text{(TmEq-Next-Var)}\\
\mathrm{next}^{\kappa}\,\xi\,[x \leftarrow t, y \leftarrow u]\,\xi'.v &\equiv \mathrm{next}^{\kappa}\,\xi\,[y \leftarrow u, x \leftarrow t]\,\xi'.v && \text{(TmEq-Next-Exch)}\\
\mathrm{next}^{\kappa}\,\xi.\,\mathrm{next}^{\kappa}\,\xi'.u &\equiv \mathrm{next}^{\kappa}\,\xi'.\,\mathrm{next}^{\kappa}\,\xi.u && \text{(TmEq-Next-Comm)}\\
\mathrm{next}^{\kappa}\,\xi\,[x \leftarrow \mathrm{next}^{\kappa}\,\xi.t].u &\equiv \mathrm{next}^{\kappa}\,\xi.u[t/x] && \text{(TmEq-Force)}\\
\mathrm{fix}^{\kappa}\,x.t &\equiv t[\mathrm{next}^{\kappa}\,(\mathrm{fix}^{\kappa}\,x.t)/x] && \text{(TmEq-Fix)}
\end{aligned}$$

Figure 7.3: New type and term equalities in gDTT. Rules TyEq-$\triangleright$-Weak and TmEq-Next-Weak require that A and u are well-formed in a context without x. Rules TyEq-$\triangleright$-Exch and TmEq-Next-Exch assume that exchanging x and y is allowed, i.e., that the type of x does not depend on y and vice versa. Likewise, rule TmEq-Next-Comm assumes that exchanging the codomains of ξ and ξ' is allowed and that none of the variables in the codomains of ξ and ξ' appear in the type of u.

we say that it is defined by *guarded recursion*. When the term is intuitively a proof, we say we are proving by *Löb induction* [9].

Guarded recursive types are defined as fixed-points of suitably guarded functions on universes. This is the approach of Birkedal and Møgelberg [18], but the generality of the rules of gDTT allows us to define more interesting dependent guarded recursive types, for example the predicates of Section 7.3.

We first illustrate the technique by defining the (non-dependent) type of guarded streams. Recall from the introduction that we want the type of guarded streams, for clock κ, to satisfy the equation $\mathrm{Str}^{\kappa}_{A} \equiv A \times \overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{A}$.

The type A will be equal to $\mathrm{El}(B)$ for some code B in some universe $\mathcal{U}_{\Delta}$ where the clock variable κ is not in Δ. We then define the *code* S^{κ}_{A} of $\mathrm{Str}^{\kappa}_{A}$ in the universe $\mathcal{U}_{\Delta,\kappa}$ to be $S^{\kappa}_{A} \triangleq \mathrm{fix}^{\kappa}\,X.B\widehat{\times}\widehat{\triangleright}^{\kappa}X$, where $\widehat{\times}$ is the code of the (simple) product type. Via the rules of gDTT we can show $\mathrm{Str}^{\kappa}_{A} \simeq A \times \overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{A}$ as desired.

The head and tail operations, $\mathrm{hd}^{\kappa} : \mathrm{Str}^{\kappa}_{A} \to A$ and $\mathrm{tl}^{\kappa} : \mathrm{Str}^{\kappa}_{A} \to \overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{A}$ are simply the first and the second projections. Conversely, we construct streams by pairing. We use the suggestive cons^{κ} notation which we define as

$$\mathrm{cons}^{\kappa} : A \to \overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{A} \to \mathrm{Str}^{\kappa}_{A} \qquad\qquad \mathrm{cons}^{\kappa} \triangleq \lambda\,(a : A)\left(as : \overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{A}\right).\langle a, as\rangle$$

Defining guarded streams is also done via guarded recursion, for example the stream consisting only of ones is defined as $\mathsf{ones} \triangleq \mathrm{fix}^{\kappa}\, x.\, \mathrm{cons}^{\kappa}\, 1\, x$.

The rule TyEq-El-▷ is essential for defining guarded recursive types as fixed-points on universes, and it can also be used for defining more advanced guarded recursive dependent types such as covectors; see Section 7.3.

Identity types

The type theory gDTT has standard extensional identity types $\mathrm{Id}_{A}(t,u)$ (see, e.g., Jacobs [51]) but with two additional type equivalences necessary for working with guarded dependent types. We write $\mathrm{r}_{A}\, t$ for the reflexivity proof $\mathrm{Id}_{A}(t,t)$. The first type equivalence is the rule TyEq-▷. This rule, which is validated by the model of Section 7.6, may be thought of by analogy to type equivalences often considered in homotopy type theory [92], such as

$$\mathrm{Id}_{A\times B}(\langle s_1, s_2\rangle, \langle t_1, t_2\rangle) \equiv \mathrm{Id}_{A}(s_1, t_1) \times \mathrm{Id}_{B}(s_2, t_2). \tag{7.2}$$

There are two important differences. The first is that (7.2) is (using univalence) a propositional type equality, whereas TyEq-▷ specifies a definitional type equality. This is natural in an extensional type theory. The second difference is that there are terms going in both directions in (7.2), whereas we would have a term of type $\mathrm{Id}_{\overset{\kappa}{\triangleright}\xi.A}(\mathrm{next}^{\kappa}\,\xi.t, \mathrm{next}^{\kappa}\,\xi.u) \to \overset{\kappa}{\triangleright}\xi.\mathrm{Id}_{A}(t,u)$ without the rule TyEq-▷.

The second novel type equality rule, which involves clock quantification, will be presented in Section 7.4.

7.3 Examples

In this section we present some example terms typable in gDTT. Our examples will use a term, which we call $\mathrm{p}\eta$, of type $\Pi(s,t : A\times B).\mathrm{Id}_{A}(\pi_1 t, \pi_1 s) \to \mathrm{Id}_{B}(\pi_2 t, \pi_2 s) \to \mathrm{Id}_{A\times B}(t,s)$. This term is definable in any type theory with a strong (dependent) elimination rule for dependent sums. The second property we will use is that $\mathrm{Str}^{\kappa}_{A} \equiv A \times \overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{A}$. Because hd^{κ} and tl^{κ} are simply first and second projections, $\mathrm{p}\eta$ also has type $\Pi\left(xs, ys : \mathrm{Str}^{\kappa}_{A}\right).\mathrm{Id}_{A}(\mathrm{hd}^{\kappa}\, xs, \mathrm{hd}^{\kappa}\, ys) \to \mathrm{Id}_{\overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{A}}(\mathrm{tl}^{\kappa}\, xs, \mathrm{tl}^{\kappa}\, ys) \to \mathrm{Id}_{\mathrm{Str}^{\kappa}_{A}}(xs, ys)$.

$\mathrm{zipWith}^{\kappa}$ **preserves commutativity.** In gDTT we define the $\mathrm{zipWith}^{\kappa}$ function which has the type $(A \to B \to C) \to \mathrm{Str}^{\kappa}_{A} \to \mathrm{Str}^{\kappa}_{B} \to \mathrm{Str}^{\kappa}_{C}$ by

$$\mathrm{zipWith}^{\kappa} f \triangleq \mathrm{fix}^{\kappa}\, \varphi.\lambda xs, ys.\, \mathrm{cons}^{\kappa} \, (f \, (\mathrm{hd}^{\kappa}\, xs) \, (\mathrm{hd}^{\kappa}\, ys)) \, (\varphi \textcircled{κ} \mathrm{tl}^{\kappa}\, xs \textcircled{κ} \mathrm{tl}^{\kappa}\, ys).$$

We show that commutativity of f implies commutativity of $\mathrm{zipWith}^{\kappa} f$, i.e., that

$$\begin{aligned}&\Pi(f : A \to A \to B).\, (\Pi(x, y : A).\mathrm{Id}_B(f\, x\, y, f\, y\, x)) \to \\ &\Pi\Big(xs, ys : \mathrm{Str}^{\kappa}_{A}\Big).\mathrm{Id}_{\mathrm{Str}^{\kappa}_{B}}(\mathrm{zipWith}^{\kappa} f\, xs\, ys, \mathrm{zipWith}^{\kappa} f\, ys\, xs)\end{aligned}$$

is inhabited. The term that inhabits this type is

$$\lambda f.\lambda c.\, \mathrm{fix}^{\kappa}\, \varphi.\lambda xs, ys.\, \mathrm{p}\eta \; (c \, (\mathrm{hd}^{\kappa}\, xs) \, (\mathrm{hd}^{\kappa}\, ys)) \; (\varphi \textcircled{κ} \mathrm{tl}^{\kappa}\, xs \textcircled{κ} \mathrm{tl}^{\kappa}\, ys).$$

Here, φ has type $\overset{\kappa}{\triangleright}(\Pi\Big(xs, ys : \mathrm{Str}^{\kappa}_{A}\Big).\mathrm{Id}_{\mathrm{Str}^{\kappa}_{B}}(\mathrm{zipWith}^{\kappa} f\, xs\, ys, \mathrm{zipWith}^{\kappa} f\, ys\, xs))$ so to type the term above, we crucially need delayed substitutions.

An example with covectors. The next example is more sophisticated, as it involves programming and proving with a data type that, unlike streams, is dependently typed. Indeed the generalised later, carrying a delayed substitution, is necessary to type even elementary programs. *Covectors* are the potentially infinite version of vectors (lists with length). To define guarded covectors we first need guarded co-natural numbers. The definition in gDTT is $\mathrm{Co}\mathbb{N}^{\kappa} \triangleq \mathrm{El}\Big(\mathrm{fix}^{\kappa}\, X.(\widehat{\mathbf{1}}\widehat{+\triangleright}^{\kappa} X)\Big)$; this type satisfies $\mathrm{Co}\mathbb{N}^{\kappa} \equiv \mathbf{1} + \overset{\kappa}{\triangleright}\mathrm{Co}\mathbb{N}^{\kappa}$. Using $\mathrm{Co}\mathbb{N}^{\kappa}$ we can define the type family of covectors $\mathrm{CoVec}^{\kappa}_{A}\, n \triangleq \mathrm{El}(\widehat{\mathrm{CoVec}}^{\kappa}_{A}\, n)$, where

$$\begin{aligned}\widehat{\mathrm{CoVec}}^{\kappa}_{A} \triangleq \mathrm{fix}^{\kappa}&\Big(\varphi : \overset{\kappa}{\triangleright}(\mathrm{Co}\mathbb{N}^{\kappa} \to \mathcal{U}_{\Delta,\kappa})\Big).\lambda(n : \mathrm{Co}\mathbb{N}^{\kappa}).\, \mathrm{case}\, n\, \mathrm{of} \\ &\mathrm{inl}\, u \Rightarrow \widehat{\mathbf{1}} \\ &\mathrm{inr}\, m \Rightarrow A \widehat{\times\triangleright}^{\kappa} (\varphi \textcircled{κ} m).\end{aligned}$$

We will not distinguish between $\mathrm{CoVec}^{\kappa}_{A}$ and $\widehat{\mathrm{CoVec}}^{\kappa}_{A}$. As an example of covectors, we define ones of type $\Pi(n : \mathrm{Co}\mathbb{N}^{\kappa}).\, \mathrm{CoVec}^{\kappa}_{\mathbb{N}}\, n$ which produces a covector of any length consisting only of ones:

$$\mathsf{ones} \triangleq \mathrm{fix}^{\kappa}\, \varphi.\lambda(n : \mathrm{Co}\mathbb{N}^{\kappa}).\, \mathrm{case}\, n\, \mathrm{of}\, \{\mathrm{inl}\, u \Rightarrow \mathrm{inl}\, \langle\rangle ; \mathrm{inr}\, m \Rightarrow \langle 1, \varphi \textcircled{κ} m\rangle\}.$$

Although this is one of the simplest covector programs one can imagine, it does not type-check without the generalised later with delayed substitutions.

The map function on covectors is defined as

$$\begin{aligned}\mathrm{map} \; &: (A \to B) \to \Pi(n : \mathrm{Co}\mathbb{N}^{\kappa}).\, \mathrm{CoVec}^{\kappa}_{A}\, n \to \mathrm{CoVec}^{\kappa}_{B}\, n \\ \mathrm{map}\, f &\triangleq \mathrm{fix}^{\kappa}\, \varphi.\lambda(n : \mathrm{Co}\mathbb{N}^{\kappa}).\, \mathrm{case}\, n\, \mathrm{of} \\ &\quad \mathrm{inl}\, u \Rightarrow \lambda(x : 1).x \\ &\quad \mathrm{inr}\, m \Rightarrow \lambda\Big(p : A \times \overset{\kappa}{\triangleright}[n \leftarrow m].(\mathrm{CoVec}^{\kappa}_{A}\, n)\Big).\, \langle f\, (\pi_1 p), \varphi \textcircled{κ} m \textcircled{κ} (\pi_2 p)\rangle.\end{aligned}$$

It preserves composition: the following type is inhabited

$$\begin{aligned}&\Pi(f : A \to B)(g : B \to C)(n : \mathrm{Co}\mathbb{N}^{\kappa})(xs : \mathrm{CoVec}^{\kappa}_{A}\, n).\\ &\qquad \mathrm{Id}_{\mathrm{CoVec}^{\kappa}_{C}\, n}(\mathrm{map}\, g\, n\, (\mathrm{map}\, f\, n\, xs), \mathrm{map}\, (g \circ f)\, n\, xs)\end{aligned}$$

by the term

$$\begin{aligned}&\lambda(f : A \to B)(g : B \to C).\, \mathrm{fix}^{\kappa}\, \varphi.\lambda(n : \mathrm{Co}\mathbb{N}^{\kappa}).\, \mathrm{case}\, n\, \mathrm{of}\\ &\quad \mathrm{inl}\, u \Rightarrow \lambda(xs : 1).\mathrm{r}_1\, xs\\ &\quad \mathrm{inr}\, m \Rightarrow \lambda(xs : \mathrm{CoVec}^{\kappa}_{A}(\mathrm{inr}\, m)).\, \mathrm{p}\eta\, (\mathrm{r}_C\, g(f(\pi_1 xs)))\, (\varphi \mathbin{\textcircled{\scriptsize$\kappa$}} m \mathbin{\textcircled{\scriptsize$\kappa$}} \pi_2 xs).\end{aligned}$$

7.4 Coinductive Types

As discussed in the introduction, guarded recursive types on their own disallow productive but acausal function definitions. To capture such functions we need to be able to remove $\overset{\kappa}{\triangleright}$. However such eliminations must be controlled to avoid trivialising $\overset{\kappa}{\triangleright}$. If we had an unrestricted elimination term $\mathrm{elim} : \overset{\kappa}{\triangleright} A \to A$ every type would be inhabited via fix^{κ}, making the type theory inconsistent.

However, we may eliminate $\overset{\kappa}{\triangleright}$ provided that the term does not depend on the clock κ, i.e., the term is typeable in a context where κ does not appear. Intuitively, such contexts have no temporal properties along the κ dimension, so we may progress the computation without violating guardedness. Figure 7.4 extends the system of Figure 7.2 to allow the removal of clocks in such a setting, by introducing *clock quantifiers* $\forall\kappa$ [11, 28, 71]. This is a binding construct with associated term constructor $\Lambda\kappa$, which also binds κ. The elimination term is *clock application*. Application of the term t of type $\forall\kappa.A$ to a clock κ is written as $t[\kappa]$. One may think of $\forall\kappa.A$ as analogous to the type $\forall\alpha.A$ in polymorphic lambda calculus; indeed the basic rules are precisely the same, but we have an additional construct $\mathrm{prev}\,\kappa.t$, called 'previous', to allow removal of the later modality $\overset{\kappa}{\triangleright}$.

Typing this new construct $\mathrm{prev}\,\kappa.t$ is somewhat complicated, as it requires 'advancing' a delayed substitution, which turns it into a context morphism (an actual substitution); see Figure 7.5 for the definition. The judgement $\rho :_{\Delta} \Gamma \to \Gamma'$ expresses that ρ is a context morphism from context $\Gamma \vdash_{\Delta}$ to the context $\Gamma' \vdash_{\Delta}$. We use the notation $\rho[t/x]$ for extending the context morphism by mapping the variable x to the term t. We illustrate this with two concrete examples.

First, we can indeed remove later under a clock quantifier:

$$\mathrm{force} : \forall\kappa.\overset{\kappa}{\triangleright} A \to \forall\kappa.A \qquad\qquad \mathrm{force} \triangleq \lambda x.\, \mathrm{prev}\,\kappa.x[\kappa].$$

The type is correct because advancing the empty delayed substitution in $\overset{\kappa}{\triangleright}$ turns it into the identity substitution ι, and $A\iota \equiv A$. The β and η rules ensure that force is the inverse to the canonical term $\lambda x.\Lambda\kappa.\,\mathsf{next}^{\kappa}\,x[\kappa]$ of type $\forall\kappa.A \to \forall\kappa.\overset{\kappa}{\triangleright}A$.

Second, we may see an example with a non-empty delayed substitution in the term $\mathsf{prev}\,\kappa.\,\mathsf{next}^{\kappa}\,\lambda n.\,\mathsf{succ}\,n \textcircled{κ} \mathsf{next}^{\kappa}\,0$ of type $\forall\kappa.\mathbb{N}$. Recall that $\textcircled{$\kappa$}$ is syntactic sugar and so more precisely the term is

$$\mathsf{prev}\,\kappa.\,\mathsf{next}^{\kappa}\left[\begin{array}{l} f \leftarrow \mathsf{next}^{\kappa}\,\lambda n.\,\mathsf{succ}\,n \\ x \leftarrow \mathsf{next}^{\kappa}\,0 \end{array}\right].f\,x. \tag{7.3}$$

Advancing the delayed substitution turns it into the substitution mapping the variable f to the term $(\mathsf{prev}\,\kappa.\,\mathsf{next}^{\kappa}\,\lambda n.\,\mathsf{succ}\,n)[\kappa]$ and the variable x to the term $(\mathsf{prev}\,\kappa.\,\mathsf{next}^{\kappa}\,0)[\kappa]$. Using the β rule for prev, then the β rule for $\forall\kappa$, this simplifies to the substitution mapping f to $\lambda n.\,\mathsf{succ}\,n$ and x to 0. With this we have that the term (7.3) is equal to $\Lambda\kappa.\,((\lambda n.\,\mathsf{succ}\,n)\,0)$ which is in turn equal to $\Lambda\kappa.1$.

An important property of the term $\mathsf{prev}\,\kappa.t$ is that κ is *bound* in t; hence $\mathsf{prev}\,\kappa.t$ has type $\forall\kappa.A$ instead of just A. This ensures that substitution of terms in types and terms is well-behaved and we do not need the explicit substitutions used, for example, by Clouston et al. [31] where the unary type-former $\square$ was used in place of clocks. This binding structure ensures, for instance, that the introduction rule Ty-Λ closed under substitution in Γ.

The rule TmEq-∀-fresh states that if t has type $\forall\kappa.A$ and the clock κ does not appear in the *type* A, then it does not matter to which clock t is applied, as the resulting term will be the same. In the polymorphic lambda calculus, the corresponding rule for universal quantification over types would be a consequence of relational parametricity.

We further have the construct $\widehat{\forall}$ and the rule Ty-∀-code which witness that the universes are closed under $\forall\kappa$.

To summarise, the new raw types and terms, extending those of Section 7.2, are

$$A, B ::= \cdots \mid \forall\kappa.A \qquad\qquad t, u ::= \cdots \mid \Lambda\kappa.t \mid t[\kappa] \mid \widehat{\forall}\,t \mid \mathsf{prev}\,\kappa.t$$

Finally, we have the equality rule TyEq-∀-Id analogous to the rule TyEq-$\triangleright$. Note that, as in Section 7.2, there is a canonical term of type $\mathrm{Id}_{\forall\kappa.A}(t,s) \to \forall\kappa.\mathrm{Id}_{A}(t[\kappa], s[\kappa])$ but, without this rule, no term in the reverse direction.

Derivable type isomorphisms

The encoding of coinductive types using guarded recursive types crucially uses a family of type isomorphisms commuting $\forall\kappa$ over other type formers [11, 71]. By a type isomorphism $A \cong B$ we mean two well-typed terms f and g of types $f : A \to B$ and $g : B \to A$ such that $f(g\,x) \equiv x$ and $g(f\,x) \equiv x$. The first type isomorphism is $\forall\kappa.A \cong A$ whenever κ is not free in A. The terms $g = \lambda x.\Lambda\kappa.x$ of type $A \to \forall\kappa.A$ and $f = \lambda x.x[\kappa_0]$ of type $A \to \forall\kappa.A$ witness the

$$\frac{\Gamma \vdash_{\Delta} \qquad \Gamma \vdash_{\Delta,\kappa} A \text{ type}}{\Gamma \vdash_{\Delta} \forall\kappa.A \text{ type}}\ \textsc{Tf-}\forall \qquad \frac{\Delta' \subseteq \Delta \qquad \Gamma \vdash_{\Delta} t : \forall\kappa.\mathcal{U}_{\Delta',\kappa}}{\Gamma \vdash_{\Delta} \widehat{\forall}\, t : \mathcal{U}_{\Delta'}}\ \textsc{Ty-}\forall\textsc{-code}$$

$$\frac{\Gamma \vdash_{\Delta} \qquad \Gamma \vdash_{\Delta,\kappa} t : A}{\Gamma \vdash_{\Delta} \Lambda\kappa.t : \forall\kappa.A}\ \textsc{Ty-}\Lambda \qquad \frac{\vdash_{\Delta} \kappa' \qquad \Gamma \vdash_{\Delta} t : \forall\kappa.A}{\Gamma \vdash_{\Delta} t[\kappa'] : A[\kappa'/\kappa]}\ \textsc{Ty-app}$$

$$\frac{\Gamma \vdash_{\Delta} \qquad \Gamma \vdash_{\Delta,\kappa} t : \overset{\kappa}{\triangleright}\xi.A}{\Gamma \vdash_{\Delta} \mathrm{prev}\,\kappa.t : \forall\kappa.(A(\mathrm{adv}^{\kappa}_{\Delta}(\xi)))}\ \textsc{Ty-prev}$$

Figure 7.4: Overview of the new typing rules for coinductive types.

$$\frac{\vdash_{\Delta,\kappa} \cdot : \Gamma \overset{\kappa}{\rightarrowtriangle} \cdot \qquad \Gamma \vdash_{\Delta}}{\mathrm{adv}^{\kappa}_{\Delta}(\cdot) \triangleq \iota :_{\Delta,\kappa} \Gamma \to \Gamma}$$

$$\frac{\vdash_{\Delta,\kappa} \xi[x \leftarrow t] : \Gamma \overset{\kappa}{\rightarrowtriangle} \Gamma', x : A \qquad \Gamma \vdash_{\Delta}}{\mathrm{adv}^{\kappa}_{\Delta}(\xi[x \leftarrow t]) \triangleq \mathrm{adv}^{\kappa}_{\Delta}(\xi)[(\mathrm{prev}\,\kappa.t)[\kappa]/x] :_{\Delta,\kappa} \Gamma \to \Gamma, \Gamma', x : A}$$

Figure 7.5: Advancing a delayed substitution.

isomorphism. Note that we used the clock constant κ_0 in an essential way. The equality $f(g\,x) \equiv x$ follows using only the β rule for clock application. The equality $g(f\,x) \equiv x$ follows using by the rule TmEq-∀-fresh.

The following type isomorphisms follow by using β and η laws for the constructs involved.

- If $\kappa \notin A$ then $\forall\kappa.\Pi(x : A).B \cong \Pi(x : A).\forall\kappa.B$.
- $\forall\kappa.\Sigma\,(x : A)\,B \cong \Sigma\,(y : \forall\kappa.A)\,(\forall\kappa.B[y[\kappa]/x])$.
- $\forall\kappa.A \cong \forall\kappa.\overset{\kappa}{\triangleright}A$.

There is an important additional type isomorphism witnessing that $\forall\kappa$ commutes with binary sums; however unlike the isomorphisms above we require equality reflection to show that the two functions are inverse to each other up to definitional equality. There is a canonical term of type $\forall\kappa.A + \forall\kappa.B \to \forall\kappa.(A+B)$ using just ordinary elimination of coproducts. Using the fact that we encode binary coproducts using Σ-types and universes we can define a term com^{+} of type $\forall\kappa.(A+B) \to \forall\kappa.A + \forall\kappa.B$ which is a inverse to the canonical term. In particular com^{+} satisfies the following two equalities

Definitional type equalities:

$$\frac{\Gamma \vdash_\Delta \qquad \Delta' \subseteq \Delta \qquad \Gamma \vdash_{\Delta,\kappa} t : \mathcal{U}_{\Delta',\kappa}}{\Gamma \vdash_\Delta \mathrm{El}(\widehat{\forall}\Lambda\kappa.t) \equiv \forall\kappa.\,\mathrm{El}(t)}\ \textsc{TyEq-}\forall\textsc{-el}$$

$$\frac{\Gamma \vdash_\Delta \qquad \Gamma \vdash_{\Delta,\kappa} A \text{ type} \qquad \Gamma \vdash_\Delta t : \forall\kappa.A \qquad \Gamma \vdash_\Delta s : \forall\kappa.A}{\Gamma \vdash_\Delta \forall\kappa.\mathrm{Id}_A(t[\kappa], s[\kappa]) \equiv \mathrm{Id}_{\forall\kappa.A}(t,s)}\ \textsc{TyEq-}\forall\textsc{-Id}$$

Definitional term equalities:

$$\frac{\Gamma \vdash_\Delta \qquad \vdash_\Delta \kappa' \qquad \Gamma \vdash_{\Delta,\kappa} t : A}{\Gamma \vdash_\Delta (\Lambda\kappa.t)[\kappa'] \equiv t[\kappa'/\kappa] : A[\kappa'/\kappa]}\ \textsc{TmEq-}\forall\text{-}\beta \qquad \frac{\kappa \notin \Delta \qquad \Gamma \vdash_\Delta t : \forall\kappa.A}{\Gamma \vdash_\Delta \Lambda\kappa.t[\kappa] \equiv t : \forall\kappa.A}\ \textsc{TmEq-}\forall\text{-}\eta$$

$$\frac{\kappa \notin \Delta \qquad \Gamma \vdash_\Delta A \text{ type} \qquad \Gamma \vdash_\Delta t : \forall\kappa.A \qquad \vdash_\Delta \kappa' \qquad \vdash_\Delta \kappa''}{\Gamma \vdash_\Delta t[\kappa'] \equiv t[\kappa''] : A}\ \textsc{TmEq-}\forall\textsc{-fresh}$$

$$\frac{\Gamma \vdash_\Delta \qquad \vdash_{\Delta,\kappa} \xi : \Gamma \overset{\kappa}{\rightarrow} \Gamma' \qquad \Gamma, \Gamma' \vdash_{\Delta,\kappa} t : A}{\Gamma \vdash_\Delta \mathrm{prev}\,\kappa.\,\mathrm{next}^\kappa\,\xi.t \equiv \Lambda\kappa.t(\mathrm{adv}^\kappa_\Delta(\xi)) : \forall\kappa.(A(\mathrm{adv}^\kappa_\Delta(\xi)))}\ \textsc{TmEq-}\text{prev-}\beta$$

$$\frac{\Gamma \vdash_\Delta \qquad \Gamma \vdash_{\Delta,\kappa} t : \overset{\kappa}{\triangleright} A}{\Gamma \vdash_{\Delta,\kappa} \mathrm{next}^\kappa\,((\mathrm{prev}\,\kappa.t)[\kappa]) \equiv t : \overset{\kappa}{\triangleright} A}\ \textsc{TmEq-}\text{prev-}\eta$$

Figure 7.6: Type and term equalities involving clock quantification.

which will be used below.

$$\mathrm{com}^+\,(\Lambda\kappa.\,\mathrm{inl}\,t) \equiv \mathrm{inl}\,\Lambda\kappa.t \qquad \mathrm{com}^+\,(\Lambda\kappa.\,\mathrm{inr}\,t) \equiv \mathrm{inr}\,\Lambda\kappa.t. \tag{7.4}$$

7.5 Example Programs with Coinductive Types

Let A be a type with code $\widehat{A}$ in clock context Δ and κ a fresh clock variable. Let $\mathrm{Str}_A = \forall\kappa.\,\mathrm{Str}^\kappa_A$. We can define head, tail and cons functions

$$\begin{aligned} \mathrm{hd} &: \mathrm{Str}_A \to A & \mathrm{hd} &\triangleq \lambda xs.\,\mathrm{hd}^{\kappa_0}\,(xs[\kappa_0]) \\ \mathrm{tl} &: \mathrm{Str}_A \to \mathrm{Str}_A & \mathrm{tl} &\triangleq \lambda xs.\,\mathrm{prev}\,\kappa.\,\mathrm{tl}^\kappa\,(xs[\kappa]) \\ \mathrm{cons} &: A \to \mathrm{Str}_A \to \mathrm{Str}_A & \mathrm{cons} &\triangleq \lambda x.\lambda xs.\Lambda\kappa.\,\mathrm{cons}^\kappa\,x\,(\mathrm{next}^\kappa\,(xs[\kappa])). \end{aligned}$$

With these we can define the *acausal* 'every other' function eo^κ that removes every second element of the input stream. It is acausal because the second element of the output stream is the third element of the input. Therefore to type the function we need to have the input stream always available, so clock quantification must be used. The function eo^{κ} of type $\mathrm{Str}_A \to \mathrm{Str}^{\kappa}_A$ is defined as

$$\mathrm{eo}^{\kappa} \triangleq \mathrm{fix}^{\kappa}\,\varphi.\lambda\,(xs : \mathrm{Str}_A).\,\mathrm{cons}^{\kappa}(\mathrm{hd}\,xs)\,(\varphi \textcircled{\kappa}\,\mathrm{next}^{\kappa}\,((\mathrm{tl}\,(\mathrm{tl}\,xs)))).$$

The result is a *guarded* stream, but we can easily strengthen it and define eo of type $\mathrm{Str}_A \to \mathrm{Str}_A$ as $\mathrm{eo} \triangleq \lambda xs.\Lambda\kappa.\,\mathrm{eo}^{\kappa}\,xs$.

We can also work with covectors (not just with guarded covectors as we did in Section 7.3). This is a dependent coinductive type indexed by conatural numbers which is the type $\mathrm{Co}\mathbb{N} = \forall\kappa.\,\mathrm{Co}\mathbb{N}^{\kappa}$. It is easy to define $\overline{0}$ and $\overline{\mathrm{succ}}$ as $\overline{0} \triangleq \Lambda\kappa.\,\mathrm{inl}\,\langle\rangle$ and $\overline{\mathrm{succ}} \triangleq \lambda n.\Lambda\kappa.\,\mathrm{inr}\,(\mathrm{next}^{\kappa}\,(n[\kappa]))$. Next, we can define a transport function $\mathrm{com}^{\mathrm{Co}\mathbb{N}}$ of type $\mathrm{com}^{\mathrm{Co}\mathbb{N}} : \mathrm{Co}\mathbb{N} \to 1 + \mathrm{Co}\mathbb{N}$ satisfying

$$\mathrm{com}^{\mathrm{Co}\mathbb{N}}\,\overline{0} \equiv \mathrm{inl}\,\langle\rangle \qquad\qquad \mathrm{com}^{\mathrm{Co}\mathbb{N}}(\overline{\mathrm{succ}}\,n) \equiv \mathrm{inr}\,n. \tag{7.5}$$

This function is used to define the type family of covectors as $\mathrm{CoVec}_A\,n \triangleq \forall\kappa.\,\mathrm{CoVec}^{\kappa}_A\,n$ where $\mathrm{CoVec}^{\kappa}_A : \mathrm{Co}\mathbb{N} \to \mathcal{U}_{\Delta,\kappa}$ is the term

$$\mathrm{fix}^{\kappa}\,\varphi.\lambda\,(n : \mathrm{Co}\mathbb{N}).\,\mathrm{case}\,\mathrm{com}^{\mathrm{Co}\mathbb{N}}\,n\,\mathrm{of}\Big\{\mathrm{inl}\,\_ \Rightarrow \widehat{1};\mathrm{inr}\,n \Rightarrow A\widehat{\times\triangleright}^{\kappa}\,(\varphi \textcircled{\kappa}\,(\mathrm{next}^{\kappa}\,n))\Big\}.$$

Using term equalities (7.4) and (7.5) we can derive the type isomorphisms

$$\begin{aligned} \mathrm{CoVec}_A\,\overline{0} &\equiv \forall\kappa.1 \cong 1 \\ \mathrm{CoVec}_A\,(\overline{\mathrm{succ}}\,n) &\equiv \forall\kappa.\Big(A \times \overset{\kappa}{\triangleright}\Big(\mathrm{CoVec}^{\kappa}_A\,n\Big)\Big) \cong A \times \mathrm{CoVec}_A\,n \end{aligned} \tag{7.6}$$

which are the expected properties of the type of covectors.

A simple function we can define is the tail function

$$\mathrm{tl} : \mathrm{CoVec}_A(\overline{\mathrm{succ}}\,n) \to \mathrm{CoVec}_A \qquad\qquad \mathrm{tl} \triangleq \lambda v.\,\mathrm{prev}\,\kappa.\pi_2\,(v[\kappa]).$$

Note that (7.6) is needed to type tl. The map function of type

$$\mathrm{map} : (A \to B) \to \Pi(n : \mathrm{Co}\mathbb{N}).\,\mathrm{CoVec}_A\,n \to \mathrm{CoVec}_B\,n$$

is defined as $\mathrm{map}\,f \triangleq \lambda n.\lambda xs.\Lambda\kappa.\,\mathrm{map}^{\kappa}\,f\,n\,(xs[\kappa])$ where map^{κ} is

$$\begin{aligned} &\mathrm{map}^{\kappa} : (A \to B) \to \Pi(n : \mathrm{Co}\mathbb{N}).\,\mathrm{CoVec}^{\kappa}_A\,n \to \mathrm{CoVec}^{\kappa}_B\,n \\ &\mathrm{map}^{\kappa} = \lambda f.\,\mathrm{fix}^{\kappa}\,\varphi.\lambda n.\,\mathrm{case}\,\mathrm{com}^{\mathrm{Co}\mathbb{N}}\,n\,\mathrm{of} \\ &\qquad\qquad\qquad \mathrm{inl}\,\_ \Rightarrow \lambda v.v \\ &\qquad\qquad\qquad \mathrm{inr}\,n \Rightarrow \lambda v.\,\langle f\,(\pi_1 v), \varphi \textcircled{\kappa}\,(\mathrm{next}^{\kappa}\,n) \textcircled{\kappa}\,\pi_2(v)\rangle. \end{aligned}$$

Lifting guarded functions

In this section we show how in general we may lift a function on guarded recursive types, such as addition of guarded streams, to a function on coinductive streams. Moreover, we show how to lift proofs of properties, such as the commutativity of addition, from guarded recursive types to coinductive types.

Let Γ be a context in clock context Δ and κ a fresh clock. Suppose A and B are types such that $\Gamma \vdash_{\Delta,\kappa} A$ type and $\Gamma, x : A \vdash_{\Delta,\kappa} B$ type. Finally let f be a function of type $\Gamma \vdash_{\Delta,\kappa} f : \Pi(x : A).B$. We define $\mathfrak{L}(f)$ satisfying the typing judgement $\Gamma \vdash_{\Delta} \mathfrak{L}(f) : \Pi(y : \forall\kappa.A).\forall\kappa.(B[y[\kappa]/x])$ as $\mathfrak{L}(f) \triangleq \lambda y.\Lambda\kappa.f\,(y[\kappa])$.

Now assume that f' is another term of type $\Pi(x : A).B$ (in the same context) and that we have proved $\Gamma \vdash_{\Delta,\kappa} p : \Pi(x : A).\mathrm{Id}_B(f\,x, f'\,x)$. As above we can give the term $\mathfrak{L}(p)$ the type $\Pi(y : \forall\kappa.A).\forall\kappa.\mathrm{Id}_{B[y[\kappa]/x]}(f\,(y[\kappa]), f'(y[\kappa]))$. which by using the type equality TyEq-$\forall$-Id and the η rule for $\forall$ is equal to the type $\Pi(y : \forall\kappa.A).\mathrm{Id}_{\forall\kappa.B[y[\kappa]/x]}(\mathfrak{L}(f)\,y, \mathfrak{L}(f')\,y)$. So we have derived a property of lifted functions $\mathfrak{L}(f)$ and $\mathfrak{L}(f')$ from the properties of the guarded versions f and f'. This is a standard pattern. Using Löb induction we prove a property of a function whose result is a "guarded" type and derive the property for the lifted function.

For example we can lift the zipWith function from guarded streams to coinductive streams and prove that it preserves commutativity, using the result on guarded streams of Section 7.3.

7.6 Soundness

gDTT can be shown to be sound with respect to a denotational model interpreting the type theory. The model is a refinement of [28]. Here we provide some intuition for the semantics of delayed substitutions, we just describe how to interpret the rule

$$\frac{x : A \vdash B\ \text{type} \qquad \vdash t : \triangleright A}{\vdash \triangleright[x \leftarrow t].B\ \text{type}} \tag{7.7}$$

in the case where we only have one clock available.

The subsystem of gDTT with only one clock can be modelled in the category $\mathcal{S}$, known as the topos of trees [22], the presheaf category over the first infinite ordinal ω. The objects X of $\mathcal{S}$ are families of sets $X_1, X_2, \ldots$ indexed by the positive integers, together with families of *restriction functions* $r_i^X : X_{i+1} \to X_i$ indexed similarly. There is a functor $\blacktriangleright : \mathcal{S} \to \mathcal{S}$ which maps an object X to the object

$$\begin{tikzcd} 1 & X_1 \arrow[l, "!"'] & X_2 \arrow[l, "r_1^X"'] & X_3 \arrow[l, "r_2^X"'] & \cdots \arrow[l] \end{tikzcd}$$

where ! is the unique map into the terminal object.

In this model, a closed type A is interpreted as an object of $\mathcal{S}$ and the type $x : A \vdash B$ type is interpreted as an indexed family of sets $B_i(a)$, for a in A_i together with maps $r_i^B(a) \colon B_{i+1}(a) \to B_i(r_i^A(a))$. The term t in (7.7) is interpreted as a morphism $t : 1 \to \triangleright A$ so $t_i(*)$ is an element of A_i (here we write $*$ for the element of 1).

The type $\vdash \triangleright[x \leftarrow t].B$ type is then interpreted as the object X, defined by

$$X_1 = 1 \qquad\qquad X_{i+1} = B_i(t_{i+1}(*)).$$

Notice that the delayed substitution is interpreted by substitution (reindexing) in the model; the change of the index in the model (B_i is reindexed along $t_{i+1}(*)$) corresponds to the delayed substitution in the type theory. Further notice that if B does not depend on x, then the interpretation of

$$\vdash \triangleright[x \leftarrow t].B \text{ type}$$

reduces to the interpretation $\triangleright B$, which is defined to be $\blacktriangleright$ applied to the interpretation of B.

The above can be generalised to work for general contexts and sequences of delayed substitutions, and one can then validate that the judgemental equality rules do indeed hold in this model.

7.7 Related Work

Birkedal et al. [22] introduced dependent type theory with the $\triangleright$ modality, with semantics in the topos of trees. The guardedness requirement was expressed using the syntactic check that every occurrence of a type variable lies beneath a $\triangleright$. This requirement was subsequently refined by Birkedal and Møgelberg [18], who showed that guarded recursive types could be constructed via fixed-points of functions on universes. However, the rules considered in these papers do not allow one to apply terms of type $\triangleright(\Pi(x : A).B)$, as the applicative functor construction $\circledast$ was defined only for simple function spaces. They are therefore less expressive for both programming (consider the covector ones, and function map, of Section 7.3) and proving, noting the extensive use of delayed substitutions in our example proofs. They further do not consider coinductive types, and so are restricted to causal functions.

The extension to coinductive types, and hence acausal functions, is due to Atkey and McBride [11], who introduced *clock quantifiers* into a simply typed setting with guarded recursion. Møgelberg [71] extended this work to dependent types and Bizjak and Møgelberg [28] refined the model further to allow clock synchronisation.

Clouston et al. [31] introduced the logic $Lg\lambda$ to prove properties of terms of the (simply typed) guarded λ-calculus, $g\lambda$. This allowed proofs about coinductive types, but not in the integrated fashion supported by dependent type theories. Moreover it relied on types being “total”, a property that in a dependently typed setting would entail a strong elimination rule for $\triangleright$, which would lead to inconsistency.

Sized types [50] have been combined with copatterns [1] as an alternative type-based approach for modular programming with coinductive types. This work is more mature than ours with respect to implementation and the demonstration of syntactic properties such as normalisation, and so further development of gDTT is essential to enable proper comparison. One advantage of gDTT is that the later modality is useful for examples beyond coinduction, and beyond the utility of sized types, such as the guarded recursive domain equations used to model program logics [91].

7.8 Conclusion and Future Work

We have described the dependent type theory gDTT. The examples we have detailed show that gDTT provides a setting for programming and proving with guarded recursive and coinductive types.

In future work we plan to investigate an intensional version of the type theory and construct a prototype implementation to allow us to experiment with larger examples. Preliminary work has suggested that the path type of cubical type theory [32] interacts better with the new constructs of gDTT than the ordinary Martin-Löf identity type.

Finally, we are investigating whether the generalisation of applicative functors [69] to apply over *dependent* function spaces, via delayed substitutions, might also apply to examples quite unconnected to the later modality.

Acknowledgements. This research was supported in part by the ModuRes Sapere Aude Advanced Grant and DFF-Research Project 1 Grant no. 4002-00442, both from The Danish Council for Independent Research for the Natural Sciences (FNU). Aleš Bizjak was supported in part by a Microsoft Research PhD grant.

7.A Typing Rules

Definitional type equalities:

$$\frac{\Gamma, \Gamma' \vdash_\Delta A \text{ type} \qquad \vdash_\Delta \xi[x \leftarrow t] : \Gamma \xrightarrow{\kappa} \Gamma', x : B}{\Gamma \vdash_\Delta \overset{\kappa}{\triangleright}\xi\,[x \leftarrow t].A \equiv \overset{\kappa}{\triangleright}\xi.A}\ \textsc{TyEq-}\triangleright\textsc{-Weak}$$

$$\frac{\begin{array}{c}\Gamma, \Gamma', x : B, y : C, \Gamma'' \vdash_\Delta A \text{ type}\\ \vdash_\Delta \xi\,[x \leftarrow t, y \leftarrow u]\,\xi' : \Gamma \xrightarrow{\kappa} \Gamma', x : B, y : C, \Gamma'' \qquad x \text{ not free in } C\end{array}}{\Gamma \vdash_\Delta \overset{\kappa}{\triangleright}\xi\,[x \leftarrow t, y \leftarrow u]\,\xi'.A \equiv \overset{\kappa}{\triangleright}\xi\,[y \leftarrow u, x \leftarrow t]\,\xi'.A}\ \textsc{TyEq-}\triangleright\textsc{-Exch}$$

$$\frac{\Gamma \vdash_\Delta \overset{\kappa}{\triangleright}\xi\,[x \leftarrow \mathrm{next}^\kappa\,\xi.t].A \text{ type}}{\Gamma \vdash_\Delta \overset{\kappa}{\triangleright}\xi\,[x \leftarrow \mathrm{next}^\kappa\,\xi.t].A \equiv \overset{\kappa}{\triangleright}\xi.A[t/x]}\ \textsc{TyEq-Force}$$

$$\frac{\Delta' \subseteq \Delta \qquad \vdash_{\Delta'} \kappa \qquad \Gamma, \Gamma' \vdash_\Delta A : \mathcal{U}_{\Delta'} \qquad \vdash_\Delta \xi : \Gamma \xrightarrow{\kappa} \Gamma'}{\Gamma \vdash_\Delta \mathrm{El}(\widehat{\triangleright}^\kappa\,(\mathrm{next}^\kappa\,\xi.A)) \equiv \overset{\kappa}{\triangleright}\xi.\,\mathrm{El}(t)}\ \textsc{TyEq-El-}\triangleright$$

$$\frac{\vdash_\Delta \xi : \Gamma \xrightarrow{\kappa} \Gamma' \qquad \Gamma, \Gamma' \vdash_\Delta t : A \qquad \Gamma, \Gamma' \vdash_\Delta s : A}{\Gamma \vdash_\Delta \mathrm{Id}_{\overset{\kappa}{\triangleright}\xi.A}(\mathrm{next}^\kappa\,\xi.t, \mathrm{next}^\kappa\,\xi.s) \equiv \overset{\kappa}{\triangleright}\xi.\mathrm{Id}_A(t, s)}\ \textsc{TyEq-}\triangleright$$

Definitional term equalities:

$$\frac{\Gamma, \Gamma' \vdash_\Delta u : A \qquad \vdash_\Delta \xi\,[x \leftarrow t] : \Gamma \xrightarrow{\kappa} \Gamma', x : B}{\Gamma \vdash_\Delta \mathrm{next}^\kappa\,\xi\,[x \leftarrow t].u \equiv \mathrm{next}^\kappa\,\xi.u : \overset{\kappa}{\triangleright}\xi.A}\ \textsc{TmEq-Next-Weak}$$

$$\frac{\Gamma \vdash_\Delta t : \overset{\kappa}{\triangleright}\xi.A}{\Gamma \vdash_\Delta \mathrm{next}^\kappa\,\xi\,[x \leftarrow t].x \equiv t : \overset{\kappa}{\triangleright}\xi.A}\ \textsc{TmEq-Next-Var}$$

$$\frac{\begin{array}{c}\Gamma, \Gamma', x : B, y : C, \Gamma'' \vdash_\Delta t : A\\ \vdash_\Delta \xi\,[x \leftarrow t, y \leftarrow u]\,\xi' : \Gamma \xrightarrow{\kappa} \Gamma', x : B, y : C, \Gamma''\\ x \text{ not free in } C\end{array}}{\Gamma \vdash_\Delta \mathrm{next}^\kappa\,\xi\,[x \leftarrow t, y \leftarrow u]\,\xi'.v \equiv \mathrm{next}^\kappa\,\xi\,[y \leftarrow u, x \leftarrow t]\,\xi'.v : \overset{\kappa}{\triangleright}\xi\,[y \leftarrow u, x \leftarrow t]\,\xi'.A}\ \textsc{TmEq-Next-Exch}$$

$$\frac{\Gamma \vdash_\Delta \mathrm{next}^\kappa\,\xi\,[x \leftarrow \mathrm{next}^\kappa\,\xi.t].u : \overset{\kappa}{\triangleright}\xi\,[x \leftarrow \mathrm{next}^\kappa\,\xi.t].A}{\Gamma \vdash_\Delta \mathrm{next}^\kappa\,\xi\,[x \leftarrow \mathrm{next}^\kappa\,\xi.t].u \equiv \mathrm{next}^\kappa\,\xi.u[t/x] : \overset{\kappa}{\triangleright}\xi.A[t/x]}\ \textsc{TmEq-Force}$$

$$\frac{\Gamma \vdash_\Delta \mathrm{fix}^\kappa\,x.t : A}{\Gamma \vdash_\Delta \mathrm{fix}^\kappa\,x.t \equiv t[\mathrm{next}^\kappa\,(\mathrm{fix}^\kappa\,x.t)/x] : A}\ \textsc{TmEq-Fix}$$

7.B Examples

In this section we provide detailed explanations of typing derivations of examples described in Section 7.3.

$\mathrm{zipWith}^{\kappa}$ preserves commutativity

The first proof is the simplest. We will define the standard $\mathrm{zipWith}^{\kappa}$ function on streams and show that if a binary function f is commutative, then so is $\mathrm{zipWith}^{\kappa} f$.

The $\mathrm{zipWith}^{\kappa} : (A \to B \to C) \to \mathrm{Str}^{\kappa}_{A} \to \mathrm{Str}^{\kappa}_{B} \to \mathrm{Str}^{\kappa}_{C}$ is defined by guarded recursion as

$$\begin{aligned}\mathrm{zipWith}^{\kappa} f \triangleq \mathrm{fix}^{\kappa}\, &\varphi.\lambda(xs, ys : \mathrm{Str}^{\kappa}_{A}).\\ &\mathrm{cons}^{\kappa}\,(f\,(\mathrm{hd}^{\kappa}\, xs)\,(\mathrm{hd}^{\kappa}\, ys))\,(\varphi \circledast \mathrm{tl}^{\kappa}\, xs \circledast \mathrm{tl}^{\kappa}\, ys)\end{aligned}$$

Note that none of the new generalised $\triangleright$ rules of gDTT are needed to type this function; this is a function on simple types.

Where we need dependent types is, of course, to state and prove properties. To prove our example, that commutativity of f implies commutativity of $\mathrm{zipWith}^{\kappa} f$, means we must show that the type

$$\begin{aligned}&\Pi(f : A \to A \to B).\,(\Pi(x, y : A).\mathrm{Id}_B(f\,x\,y, f\,y\,x)) \to\\ &\Pi\Big(xs, ys : \mathrm{Str}^{\kappa}_{A}\Big).\mathrm{Id}_{\mathrm{Str}^{\kappa}_{B}}(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys, \mathrm{zipWith}^{\kappa}\, f\, ys\, xs).\end{aligned}$$

is inhabited. We will explain how to construct such a term, and why it is typeable in gDTT. Although this construction might appear complicated at first, the actual proof term that we construct will be as simple as possible.

Let $f : A \to A \to B$ be a function and say we have a term

$$c : \Pi(x, y : A).\mathrm{Id}_B(f\,x\,y, f\,y\,x)$$

witnessing commutativity of f. We now wish to construct a term of type

$$\Pi\Big(xs, ys : \mathrm{Str}^{\kappa}_{A}\Big).\mathrm{Id}_{\mathrm{Str}^{\kappa}_{C}}(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys, \mathrm{zipWith}^{\kappa}\, f\, ys\, xs)$$

We do this by guarded recursion. To this end we assume

$$\varphi : \overset{\kappa}{\triangleright}\Big(\Pi\Big(xs, ys : \mathrm{Str}^{\kappa}_{A}\Big).\mathrm{Id}_{\mathrm{Str}^{\kappa}_{B}}(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys, \mathrm{zipWith}^{\kappa}\, f\, ys\, xs)\Big)$$

and take $xs, ys : \mathrm{Str}^{\kappa}_{A}$. Using c (the proof that f is commutative) we first have $c\,(\mathrm{hd}^{\kappa}\, xs)\,(\mathrm{hd}^{\kappa}\, ys)$ of type

$$\mathrm{Id}_B(f\,(\mathrm{hd}^{\kappa}\, xs)\,(\mathrm{hd}^{\kappa}\, ys), f\,(\mathrm{hd}^{\kappa}\, ys)\,(\mathrm{hd}^{\kappa}\, xs))$$

and because we have by definition of $\mathrm{zipWith}^{\kappa}$

$$\begin{aligned}\mathrm{hd}^{\kappa}\,(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys) &\equiv f\,(\mathrm{hd}^{\kappa}\, xs)\,(\mathrm{hd}^{\kappa}\, ys)\\ \mathrm{hd}^{\kappa}\,(\mathrm{zipWith}^{\kappa}\, f\, ys\, xs) &\equiv f\,(\mathrm{hd}^{\kappa}\, ys)\,(\mathrm{hd}^{\kappa}\, xs)\end{aligned}$$

we see that $c\,(\mathrm{hd}^{\kappa}\, xs)\,(\mathrm{hd}^{\kappa}\, ys)$ has type

$$\mathrm{Id}_B(\mathrm{hd}^{\kappa}\,(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys), \mathrm{hd}^{\kappa}\,(\mathrm{zipWith}^{\kappa}\, f\, ys\, xs)).$$

To show that the tails are equal we use the induction hypothesis φ. The terms $\mathrm{tl}^{\kappa}\, xs$ and $\mathrm{tl}^{\kappa}\, ys$ are of type $\overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{A}$, so we first have $\varphi \circledast \mathrm{tl}^{\kappa}\, xs$ of type

$$\overset{\kappa}{\triangleright}[xs \leftarrow \mathrm{tl}^{\kappa}\, xs].\left(\Pi\left(ys : \mathrm{Str}^{\kappa}_{A}\right).\mathrm{Id}_{\mathrm{Str}^{\kappa}_{C}}\left(\begin{array}{l}\mathrm{zipWith}^{\kappa}\, f\, xs\, ys,\\ \mathrm{zipWith}^{\kappa}\, f\, ys\, xs\end{array}\right)\right)$$

Note the appearance of the generalised $\triangleright$, carrying a delayed substitution. Because the variable xs does not appear in $\overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{A}$ we may apply the weakening rule TmEq-Next-Weak to derive

$$\mathrm{tl}^{\kappa}\, ys : \overset{\kappa}{\triangleright}[xs \leftarrow \mathrm{tl}^{\kappa}\, xs].\,\mathrm{Str}^{\kappa}_{A}$$

Hence we may use the derived applicative rule to have $\varphi \circledast \mathrm{tl}^{\kappa}\, xs \circledast \mathrm{tl}^{\kappa}\, ys$ of type

$$\overset{\kappa}{\triangleright}\left[\begin{array}{l} xs \leftarrow \mathrm{tl}^{\kappa}\, xs\\ ys \leftarrow \mathrm{tl}^{\kappa}\, ys\end{array}\right].\mathrm{Id}_{\mathrm{Str}^{\kappa}_{C}}(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys, \mathrm{zipWith}^{\kappa}\, f\, ys\, xs)$$

and which is definitionally equal to the type

$$\mathrm{Id}_{\overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{C}}\left(\begin{array}{l}\mathrm{next}^{\kappa}\left[\begin{array}{l} xs \leftarrow \mathrm{tl}^{\kappa}\, xs\\ ys \leftarrow \mathrm{tl}^{\kappa}\, ys\end{array}\right].\,\mathrm{zipWith}^{\kappa}\, f\, xs\, ys,\\ \mathrm{next}^{\kappa}\left[\begin{array}{l} xs \leftarrow \mathrm{tl}^{\kappa}\, xs\\ ys \leftarrow \mathrm{tl}^{\kappa}\, ys\end{array}\right].\,\mathrm{zipWith}^{\kappa}\, f\, ys\, xs\end{array}\right).$$

We also compute

$$\begin{aligned}\mathrm{tl}^{\kappa}\,(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys) &\equiv \mathrm{next}^{\kappa}(\mathrm{zipWith}^{\kappa}\, f) \circledast \mathrm{tl}^{\kappa}\, xs \circledast \mathrm{tl}^{\kappa}\, ys\\ &\equiv \mathrm{next}^{\kappa}\left[\begin{array}{l} xs \leftarrow \mathrm{tl}^{\kappa}\, xs\\ ys \leftarrow \mathrm{tl}^{\kappa}\, ys\end{array}\right].(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys)\end{aligned}$$

and

$$\mathrm{tl}^{\kappa}\,(\mathrm{zipWith}^{\kappa}\, f\, ys\, xs) \equiv \mathrm{next}^{\kappa}\left[\begin{array}{l} ys \leftarrow \mathrm{tl}^{\kappa}\, ys\\ zs \leftarrow \mathrm{tl}^{\kappa}\, xs\end{array}\right].(\mathrm{zipWith}^{\kappa}\, f\, ys\, xs).$$

Using the exchange rule TmEq-Next-Exch we have the equality

$$
\begin{aligned}
&\mathrm{next}^{\kappa}\begin{bmatrix} ys \leftarrow \mathrm{tl}^{\kappa}\, ys \\ xs \leftarrow \mathrm{tl}^{\kappa}\, xs \end{bmatrix}.(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys)\\
&\equiv\\
&\mathrm{next}^{\kappa}\begin{bmatrix} xs \leftarrow \mathrm{tl}^{\kappa}\, xs \\ ys \leftarrow \mathrm{tl}^{\kappa}\, ys \end{bmatrix}.(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys).
\end{aligned}
$$

Putting it all together we have shown that the term $\varphi \textcircled{$\kappa$} \mathrm{tl}^{\kappa}\, xs \textcircled{κ} \mathrm{tl}^{\kappa}\, ys$ has type

$$\mathrm{Id}_{\overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{B}}(\mathrm{tl}^{\kappa}\,(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys), \mathrm{tl}^{\kappa}\,(\mathrm{zipWith}^{\kappa}\, f\, ys\, xs))$$

which means that the term

$$\mathrm{fix}^{\kappa}\,\varphi.\lambda\Big(xs, ys : \mathrm{Str}^{\kappa}_{A}\Big).\,\mathrm{p}\eta\;(c\,(\mathrm{hd}^{\kappa}\, xs)\,(\mathrm{hd}^{\kappa}\, ys))\;(\varphi \textcircled{κ} \mathrm{tl}^{\kappa}\, xs \textcircled{κ} \mathrm{tl}^{\kappa}\, ys)$$

has type $\Pi\Big(xs, ys : \mathrm{Str}^{\kappa}_{A}\Big).\mathrm{Id}_{\mathrm{Str}^{\kappa}_{B}}(\mathrm{zipWith}^{\kappa}\, f\, xs\, ys, \mathrm{zipWith}^{\kappa}\, f\, ys\, xs)$.

Notice that the resulting proof term could not be simpler than it is. In particular, we do not have to write delayed substitutions in terms, but only in the intermediate types.

An example with covectors

The next example is more sophisticated, as it will involve programming and proving with a data type that, unlike streams, is dependently typed. In particular, we will see that the generalised later, carrying a delayed substitution, is necessary to type even the most elementary programs.

Covectors are to colists (potentially infinite lists) as vectors are to lists. To define guarded covectors we first need guarded co-natural numbers. This is the type satisfying

$$\mathrm{Co}\mathbb{N}^{\kappa} \equiv \mathbf{1} + \overset{\kappa}{\triangleright}\mathrm{Co}\mathbb{N}^{\kappa}.$$

where binary sums are encoded in the type theory in a standard way. The definition in gDTT is $\mathrm{Co}\mathbb{N}^{\kappa} \triangleq \mathrm{El}\Big(\mathrm{fix}^{\kappa}\,\varphi.(\widehat{\mathbf{1}}\,\widehat{+}\,\widehat{\triangleright}^{\kappa}\varphi)\Big)$.

Using $\mathrm{Co}\mathbb{N}^{\kappa}$ we define the type of covectors of type A, written $\mathrm{CoVec}^{\kappa}_{A}$, as a $\mathrm{Co}\mathbb{N}^{\kappa}$-indexed type satisfying

$$
\begin{aligned}
\mathrm{CoVec}^{\kappa}_{A}(\mathrm{inl}\,\langle\rangle) &\equiv \mathbf{1}\\
\mathrm{CoVec}^{\kappa}_{A}(\mathrm{inr}(\mathrm{next}^{\kappa}\, m)) &\equiv A \times \overset{\kappa}{\triangleright}(\mathrm{CoVec}^{\kappa}_{A}\, m)
\end{aligned}
$$

In gDTT we first define $\widehat{\mathrm{CoVec}}^{\kappa}_{A}$

$$
\begin{aligned}
\widehat{\mathrm{CoVec}}^{\kappa}_{A} \triangleq \mathrm{fix}^{\kappa}\,\varphi.\lambda(n : \mathrm{Co}\mathbb{N}^{\kappa}).\,\mathrm{case}\, n\, \mathrm{of}\quad&\\
\mathrm{inl}\, u \Rightarrow \widehat{\mathbf{1}}&\\
\mathrm{inr}\, m \Rightarrow A\,\widehat{\times}\,\widehat{\triangleright}^{\kappa}(\varphi \textcircled{κ}\, m).&
\end{aligned}
$$

and then $\mathrm{CoVec}^{\kappa}_{A}\,n \triangleq \mathrm{El}(\widehat{\mathrm{CoVec}}^{\kappa}_{A}\,n)$. In the examples we will not distinguish between $\mathrm{CoVec}^{\kappa}_{A}$ and $\widehat{\mathrm{CoVec}}^{\kappa}_{A}$. In the above φ has type $\overset{\kappa}{\triangleright}(\mathrm{Co}\mathbb{N}^{\kappa} \to \mathcal{U}_{\Delta,\kappa})$ and inside the branches, u has type $\mathbf{1}$ and m has type $\overset{\kappa}{\triangleright}\mathrm{Co}\mathbb{N}^{\kappa}$, which is evident from the definition of $\mathrm{Co}\mathbb{N}^{\kappa}$. As an example of covectors, we define ones of type $\Pi(n : \mathrm{Co}\mathbb{N}^{\kappa}).\,\mathrm{CoVec}^{\kappa}_{\mathbb{N}}\,n$ which produces a covector of any length consisting only of ones:

$$\begin{aligned}\mathsf{ones} \triangleq \mathrm{fix}^{\kappa}\,\varphi.\lambda(n : \mathrm{Co}\mathbb{N}^{\kappa}).\,&\mathrm{case}\,n\,\mathrm{of}\\ &\mathrm{inl}\,u \Rightarrow \mathrm{inl}\,\langle\rangle\\ &\mathrm{inr}\,m \Rightarrow \langle 1, \varphi \text{\textcircled{\scriptsize κ}}\, m\rangle.\end{aligned}$$

When checking the type of this program, we need the generalised later. The type of the recursive call is $\overset{\kappa}{\triangleright}(\Pi(n : \mathrm{Co}\mathbb{N}^{\kappa}).\,\mathrm{CoVec}^{\kappa}_{\mathbb{N}}\,n)$, the type of m is $\overset{\kappa}{\triangleright}\mathrm{Co}\mathbb{N}^{\kappa}$, and therefore the type of the subterm $\varphi \text{\textcircled{\scriptsize κ}}\, m$ must be

$$\overset{\kappa}{\triangleright}[n \leftarrow m].\Pi(n : \mathrm{Co}\mathbb{N}^{\kappa}).\,\mathrm{CoVec}^{\kappa}_{\mathbb{N}}\,n.x$$

We now aim to define the function map on covectors and show that it preserves composition. Given two types A and B the map function has type

$$\mathrm{map} : (A \to B) \to \Pi(n : \mathrm{Co}\mathbb{N}^{\kappa}).\,\mathrm{CoVec}^{\kappa}_{A}\,n \to \mathrm{CoVec}^{\kappa}_{B}\,n.$$

and is defined by guarded recursion as

$$\begin{aligned}\mathrm{map}\,f \triangleq\ &\mathrm{fix}^{\kappa}\,\varphi.\lambda(n : \mathrm{Co}\mathbb{N}^{\kappa}).\\ &\mathrm{case}\,n\,\mathrm{of}\\ &\mathrm{inl}\,u \Rightarrow \lambda(x : 1).x\\ &\mathrm{inr}\,m \Rightarrow \begin{array}{l}\lambda\Big(p : A \times \overset{\kappa}{\triangleright}[n \leftarrow m].(\mathrm{CoVec}^{\kappa}_{A}\,n)\Big).\\ \quad\langle f\,(\pi_1 p), \varphi \text{\textcircled{\scriptsize κ}}\, m \text{\textcircled{\scriptsize κ}}\, (\pi_2 p)\rangle\end{array}\end{aligned}$$

Let us see why the definition has the correct type. First, the types of subterms are

$$\begin{aligned}&\varphi : \overset{\kappa}{\triangleright}(\Pi(n : \mathrm{Co}\mathbb{N}^{\kappa}).\,\mathrm{CoVec}^{\kappa}_{A}\,n \to \mathrm{CoVec}^{\kappa}_{B}\,n)\\ &u : \mathbf{1}\\ &m : \overset{\kappa}{\triangleright}\mathrm{Co}\mathbb{N}^{\kappa}\end{aligned}$$

Let $C = \mathrm{CoVec}^{\kappa}_{A}\,n \to \mathrm{CoVec}^{\kappa}_{B}\,n$, and write $C(t)$ for $C[t/n]$. By the definition of $\mathrm{CoVec}^{\kappa}_{A}$ and $\mathrm{CoVec}^{\kappa}_{B}$ we have $C(\mathrm{inl}\,u) \equiv \mathbf{1} \to \mathbf{1}$, and so $\lambda(x : \mathbf{1}).x$ has type $C(\mathrm{inl}\,u)$.

By the definition of $\mathrm{CoVec}^{\kappa}_{A}$ we have

$$\begin{aligned}\mathrm{CoVec}^{\kappa}_{A}(\mathrm{inr}\,m) &\equiv A \times \mathrm{El}\Big(\widehat{\triangleright}^{\kappa}(\mathrm{next}^{\kappa}(\mathrm{CoVec}^{\kappa}_{A}) \text{\textcircled{\scriptsize κ}}\, m)\Big)\\ &\equiv A \times \overset{\kappa}{\triangleright}[n \leftarrow m].\Big(\mathrm{CoVec}^{\kappa}_{A}\,n\Big)\end{aligned}$$

and analogously for $\mathrm{CoVec}^{\kappa}_{B}(\mathrm{inr}\, m)$. Hence the type $C(\mathrm{inr}\, m)$ is convertible to

$$\left(A \times \overset{\kappa}{\triangleright}[n \leftarrow m].\left(\mathrm{CoVec}^{\kappa}_{A}\, n\right)\right) \to \left(B \times \overset{\kappa}{\triangleright}[n \leftarrow m].\left(\mathrm{CoVec}^{\kappa}_{B}\, n\right)\right).$$

Further, using the derived applicative rule we have

$$\varphi \textcircled{\kappa}\, m : \overset{\kappa}{\triangleright}[n \leftarrow m].C(n)$$

and because $\pi_2 p$ in the second branch has type

$$\overset{\kappa}{\triangleright}[n \leftarrow m].(\mathrm{CoVec}^{\kappa}_{A}\, n)$$

we may use the (simple) applicative rule again to get

$$\varphi \textcircled{\kappa}\, m \textcircled{\kappa}\, (\pi_2 p) : \overset{\kappa}{\triangleright}[n \leftarrow m].(\mathrm{CoVec}^{\kappa}_{B}\, n)$$

which allows us to type

$$\lambda\left(p : A \times \overset{\kappa}{\triangleright}[n \leftarrow m].(\mathrm{CoVec}^{\kappa}_{A}\, n)\right).\langle f\,(\pi_1 p), \varphi \textcircled{\kappa}\, m \textcircled{\kappa}\, \pi_2(p)\rangle$$

with type $C(\mathrm{inr}\, m)$. Notice that we have made essential use of the more general applicative rule to apply $\varphi \textcircled{\kappa}\, m$ to $\pi_2 p$. Using the strong (dependent) elimination rule for binary sums we can type the whole case construct with type $C(n)$, which is what we need to give map the desired type.

Now we will show that map so defined satisfies a basic property, namely that it preserves composition in the sense that the type (in the context where we have types A, B and C)

$$\begin{gathered}\Pi(f : A \to B)(g : B \to C)(n : \mathrm{Co}\mathbb{N}^{\kappa})(xs : \mathrm{CoVec}^{\kappa}_{A}\, n).\\ \mathrm{Id}_{\mathrm{CoVec}^{\kappa}_{C}\, n}(\mathrm{map}\, g\, n(\mathrm{map}\, f\, n\, xs), \mathrm{map}(g \circ f)\, n\, xs)\end{gathered} \tag{7.8}$$

is inhabited. The proof is, of course, by Löb induction.

First we record some definitional equalities which follow directly by unfolding the definitions

$$\begin{aligned} \mathrm{map}\, f\,(\mathrm{inl}\, u)\, x &\equiv x\\ \mathrm{map}\, f\,(\mathrm{inr}\, m)\, xs &\equiv \langle f\,(\pi_1 xs), \mathrm{next}^{\kappa}(\mathrm{map}\, f) \textcircled{\kappa}\, m \textcircled{\kappa}\, \pi_2(xs)\rangle\\ &\equiv \langle f\,(\pi_1 xs), \mathrm{next}^{\kappa}\begin{bmatrix} n \leftarrow m\\ ys \leftarrow \pi_2 xs\end{bmatrix}.(\mathrm{map}\, f\, n\, ys)\rangle\end{aligned}$$

and so iterating these two equalities we get

$$\begin{aligned} \mathrm{map}\, g\,(\mathrm{inl}\, u)(\mathrm{map}\, f\,(\mathrm{inl}\, u)\, x) &\equiv x\\ \mathrm{map}\, g\,(\mathrm{inr}\, m)(\mathrm{map}\, f\,(\mathrm{inr}\, m)\, xs) &\equiv \langle g(f\,(\pi_1 xs)), s\rangle\end{aligned}$$

where s is the term

$$\mathrm{next}^{\kappa}\left[\begin{array}{l} n \leftarrow m \\ zs \leftarrow \mathrm{next}^{\kappa}\left[\begin{array}{l} n \leftarrow m \\ ys \leftarrow \pi_2 xs \end{array}\right].(\mathrm{map}\, f\, n\, ys) \end{array}\right].(\mathrm{map}\, g\, n\, zs)$$

which is convertible, by the rule TmEq-Force, to the term

$$\mathrm{next}^{\kappa}\left[\begin{array}{l} n \leftarrow m \\ ys \leftarrow \pi_2 xs \end{array}\right].(\mathrm{map}\, g\, n\, (\mathrm{map}\, f\, n\, ys)).$$

Similarly we have

$$\mathrm{map}(g \circ f)(\mathrm{inl}\, u)\, x \equiv x$$

and $\mathrm{map}(g \circ f)(\mathrm{inr}\, m)\, xs$ convertible to

$$\left\langle g(f(\pi_1 xs)), \mathrm{next}^{\kappa}\left[\begin{array}{l} n \leftarrow m \\ ys \leftarrow \pi_2 xs \end{array}\right].(\mathrm{map}(g \circ f)\, n\, ys)\right\rangle.$$

Now let us get back to proving property (7.8). Take $f : A \to B$, $g : B \to C$ and assume

$$\varphi : \overset{\kappa}{\triangleright}\Pi(n : \mathrm{Co}\mathbb{N}^{\kappa})(xs : \mathrm{CoVec}^{\kappa}_{A}\, n).\mathrm{Id}_{\mathrm{CoVec}^{\kappa}_{C}\, n}(\mathrm{map}\, g\, n (\mathrm{map}\, f\, n\, xs), \mathrm{map}(g \circ f)\, n\, xs)$$

We take $n : \mathrm{Co}\mathbb{N}^{\kappa}$ and write

$$P(n) = \Pi(xs : \mathrm{CoVec}^{\kappa}_{A}\, n).\mathrm{Id}_{\mathrm{CoVec}^{\kappa}_{C}\, n}(\mathrm{map}\, g\, n (\mathrm{map}\, f\, n\, xs), \mathrm{map}(g \circ f)\, n\, xs).$$

Then similarly as in the definition of map and the definitional equalities for map above we compute

$$P(\mathrm{inl}\, u) \equiv \Pi(xs : 1).\mathrm{Id}_{1}(xs, xs)$$

and so we have $\lambda(xs : 1).\mathrm{r}_1\, xs$ of type $P(\mathrm{inl}\, u)$.

The other branch (when $n = \mathrm{inr}\, m$) is of course a bit more complicated. As before we have

$$\mathrm{CoVec}^{\kappa}_{A}(\mathrm{inr}\, m) \equiv A \times \overset{\kappa}{\triangleright}[n \leftarrow m].\, \mathrm{CoVec}^{\kappa}_{A}\, n \tag{7.9}$$

So take xs of type $\mathrm{CoVec}^{\kappa}_{A}(\mathrm{inr}\, m)$. We need to construct a term of type

$$\mathrm{Id}_{\mathrm{CoVec}^{\kappa}_{C}\, n}(\mathrm{map}\, g\, n (\mathrm{map}\, f\, n\, xs), \mathrm{map}(g \circ f)\, n\, xs).$$

First we have $\mathrm{r}_C\, g(f(\pi_1 xs))$ of type $\mathrm{Id}_C(g(f(\pi_1 xs)), g(f(\pi_1 xs)))$. Then because m is of type $\overset{\kappa}{\triangleright}\mathrm{Co}\mathbb{N}^{\kappa}$ we can use the induction hypothesis φ to get $\varphi \textcircled{\kappa}\, m$ of type

$$\overset{\kappa}{\triangleright}[n \leftarrow m].\Pi(xs : \mathrm{CoVec}^{\kappa}_{A}\, n).\mathrm{Id}_{\mathrm{CoVec}^{\kappa}_{C}\, n}(\mathrm{map}\, g\, n (\mathrm{map}\, f\, n\, xs), \mathrm{map}(g \circ f)\, n\, xs).$$

Using (7.9) we have $\pi_2 xs$ of type $\overset{\kappa}{\triangleright}[n \leftarrow m].\mathrm{CoVec}^{\kappa}_{A}\, n$ and so we can use the applicative rule again to give $\varphi \mathbin{\textcircled{\scriptsize κ}} m \mathbin{\textcircled{\scriptsize κ}} \pi_2 xs$ the type

$$\overset{\kappa}{\triangleright}\begin{bmatrix} n \leftarrow m \\ xs \leftarrow \pi_2 xs \end{bmatrix}.\mathrm{Id}_{\mathrm{CoVec}^{\kappa}_{C}\, n}\begin{pmatrix} \mathrm{map}\, g\, n (\mathrm{map}\, f\, n\, xs), \\ \mathrm{map}(g \circ f)\, n\, xs \end{pmatrix}$$

which by the rule TyEq-▹ is the same as

$$\mathrm{Id}_{D}\begin{pmatrix} \mathrm{next}^{\kappa}\begin{bmatrix} n \leftarrow m \\ xs \leftarrow \pi_2 xs \end{bmatrix}.(\mathrm{map}\, g\, n (\mathrm{map}\, f\, n\, xs)), \\ \mathrm{next}^{\kappa}\begin{bmatrix} n \leftarrow m \\ xs \leftarrow \pi_2 xs \end{bmatrix}.(\mathrm{map}(g \circ f)\, n\, xs) \end{pmatrix}$$

where D is the type $\overset{\kappa}{\triangleright}[n \leftarrow m].\mathrm{CoVec}^{\kappa}_{C}\, n$. Thus we can give to the term

$$\lambda(xs : \mathrm{CoVec}^{\kappa}_{A}(\mathrm{inr}\, m)).\, \mathrm{p}\eta\, (\mathrm{r}_C\, g(f(\pi_1 xs)))\, (\varphi \mathbin{\textcircled{\scriptsize κ}} m \mathbin{\textcircled{\scriptsize κ}} \pi_2 xs)$$

the type $P(\mathrm{inr}\, m)$. Using the dependent elimination rule for binary sums we get the final proof of property (7.8) as the term

$$\begin{aligned}
&\lambda(f : A \to B)(g : B \to C).\, \mathrm{fix}^{\kappa}\, \varphi.\lambda(n : \mathrm{Co}\mathbb{N}^{\kappa}). \\
&\mathrm{case}\, n\, \mathrm{of} \\
&\mathrm{inl}\, u \Rightarrow \lambda(xs : 1).\mathrm{r}_1\, xs \\
&\mathrm{inr}\, m \Rightarrow \lambda(xs : \mathrm{CoVec}^{\kappa}_{A}(\mathrm{inr}\, m)).\, \mathrm{p}\eta\, (\mathrm{r}_C\, g(f(\pi_1 xs)))\, (\varphi \mathbin{\textcircled{\scriptsize κ}} m \mathbin{\textcircled{\scriptsize κ}} \pi_2 xs)
\end{aligned}$$

which is as simple as could be expected.

Lifting predicates to streams

Let $P : A \to \mathcal{U}_{\Delta}$ be a predicate on type A and κ a clock variable not in Δ. We can define a lifting of this predicate to a predicate P^{κ} on streams of elements of type A. The idea is that $P^{\kappa} xs$ will hold precisely when P holds for all elements of the stream. However we do not have access to all the element of the stream at the same time. As such we will have $P^{\kappa} xs$ if P holds for the first element of the stream xs now, and P holds for the second element of the stream xs one time step later, and so on. The precise definition uses guarded recursion:

$$\begin{aligned}
&P^{\kappa} : \mathrm{Str}^{\kappa}_{A} \to \mathcal{U}_{\Delta,\kappa} \\
&P^{\kappa} \triangleq \mathrm{fix}^{\kappa}\, \varphi.\lambda\left(xs : \mathrm{Str}^{\kappa}_{A}\right).P\, (\mathrm{hd}^{\kappa}\, xs) \widehat{\times\triangleright}^{\kappa}\, (\varphi \mathbin{\textcircled{\scriptsize κ}} \mathrm{tl}^{\kappa}\, xs).
\end{aligned}$$

In the above term the subterm φ has type $\overset{\kappa}{\triangleright}\left(\mathrm{Str}^{\kappa}_{A} \to \mathcal{U}_{\Delta,\kappa}\right)$ and so because $\mathrm{tl}^{\kappa}\, xs$ has type $\overset{\kappa}{\triangleright}\mathrm{Str}^{\kappa}_{A}$ we may form $\varphi \mathbin{\textcircled{\scriptsize κ}} \mathrm{tl}^{\kappa}\, xs$ of type $\overset{\kappa}{\triangleright}\mathcal{U}_{\Delta,\kappa}$ and so finally $\widehat{\triangleright}^{\kappa}(\varphi \mathbin{\textcircled{\scriptsize κ}} \mathrm{tl}^{\kappa}\, xs)$ has type $\mathcal{U}_{\Delta,\kappa}$ as needed.

To see that this makes sense, we have for a stream $xs : \mathrm{Str}^{\kappa}_{A}$

$$\mathrm{El}\,(P^{\kappa}\,xs) \equiv \mathrm{El}\,(P\,(\mathrm{hd}^{\kappa}\,xs)) \times \mathrm{El}\,(\widehat{\triangleright}^{\kappa}\,(\mathrm{next}^{\kappa}\,P^{\kappa} \textcircled{κ}\,\mathrm{tl}^{\kappa}\,xs)).$$

Using delayed substitution rules we have

$$\mathrm{next}^{\kappa}\,P^{\kappa} \textcircled{κ}\,\mathrm{tl}^{\kappa}\,xs \equiv \mathrm{next}^{\kappa}\,[xs \leftarrow \mathrm{tl}^{\kappa}\,xs].\,(P^{\kappa}\,xs)$$

which gives rise to the type equality

$$\mathrm{El}(\widehat{\triangleright}^{\kappa}\,\mathrm{next}^{\kappa}\,P^{\kappa} \textcircled{κ}\,\mathrm{tl}^{\kappa}\,xs) \equiv \mathrm{El}\,(\widehat{\triangleright}^{\kappa}\,\mathrm{next}^{\kappa}\,[xs \leftarrow \mathrm{tl}^{\kappa}\,xs].\,(P^{\kappa}\,xs)).$$

Finally, the type equality rule TyEq-El-$\triangleright$ gives us

$$\mathrm{El}\,(\widehat{\triangleright}^{\kappa}\,\mathrm{next}^{\kappa}\,[xs \leftarrow \mathrm{tl}^{\kappa}\,xs].\,(P^{\kappa}\,xs)) \equiv \overset{\kappa}{\triangleright}[xs \leftarrow \mathrm{tl}^{\kappa}\,xs].\,\mathrm{El}(P^{\kappa}\,xs).$$

All of these together then give us the type equality

$$\mathrm{El}\,(P^{\kappa}\,xs) \equiv \mathrm{El}(P\,(\mathrm{hd}^{\kappa}\,xs)) \times \overset{\kappa}{\triangleright}[xs \leftarrow \mathrm{tl}^{\kappa}\,xs].\,\mathrm{El}(P^{\kappa}\,xs).$$

And so if $xs = \mathrm{cons}^{\kappa}\,x\,(\mathrm{next}^{\kappa}\,ys)$ we can further simplify, using rule TyEq-Force, to get

$$\overset{\kappa}{\triangleright}[xs \leftarrow \mathrm{next}^{\kappa}\,ys].\,\mathrm{El}(P^{\kappa}\,xs) \equiv \overset{\kappa}{\triangleright}(\mathrm{El}(P^{\kappa}\,xs)[ys/xs]) \equiv \overset{\kappa}{\triangleright}\mathrm{El}(P^{\kappa}\,ys)$$

which then gives $\mathrm{El}(P^{\kappa}xs) \equiv \mathrm{El}\,(P\,x) \times \overset{\kappa}{\triangleright}\mathrm{El}\,(P^{\kappa}\,ys)$ which is in accordance with the motivation given above.

Because P^{κ} is defined by guarded recursion, we prove its properties by Löb induction. In particular, we may prove that if P holds on A then P^{κ} holds on $\mathrm{Str}^{\kappa}_{A}$, i.e., that the type

$$(\Pi(x : A).\,\mathrm{El}\,(P\,x)) \to \left(\Pi\left(xs : \mathrm{Str}^{\kappa}_{A}\right).\,\mathrm{El}\,(P^{\kappa}\,xs)\right)$$

is inhabited (in a context where we have a type A and a predicate P). Take $p : \Pi(x : A).\,\mathrm{El}\,(P\,x)$, and since we are proving by Löb induction we assume the induction hypothesis later

$$\varphi : \overset{\kappa}{\triangleright}\left(\Pi\left(xs : \mathrm{Str}^{\kappa}_{A}\right).\,\mathrm{El}\,(P^{\kappa}\,xs)\right).$$

Let $xs : \mathrm{Str}^{\kappa}_{A}$ be a stream. By definition of P^{κ} we have the type equality

$$\mathrm{El}(P^{\kappa}xs) \equiv \mathrm{El}\,(P\,\mathrm{hd}^{\kappa}\,xs) \times \overset{\kappa}{\triangleright}[xs \leftarrow \mathrm{tl}^{\kappa}\,xs].\,\mathrm{El}\,(P^{\kappa}\,xs)$$

Applying p to $\mathrm{hd}^{\kappa}\,xs$ gives us the first component

$$p(\mathrm{hd}^{\kappa}\,xs) : \mathrm{El}\,(P\,(\mathrm{hd}^{\kappa}\,xs))$$

and applying the induction hypothesis φ we have

$$\varphi \textcircled{κ}\,\mathrm{tl}^{\kappa}\,xs : \overset{\kappa}{\triangleright}[xs \leftarrow \mathrm{tl}^{\kappa}\,xs].\,\mathrm{El}(P^{\kappa}\,xs)$$

Thus combining this with the previous term we have the proof of the lifting property as the term

$$\begin{aligned} &\lambda\,(p : \Pi(x : A).\,\mathrm{El}\,(P\,x)).\\ &\quad \mathrm{fix}^{\kappa}\,\varphi.\lambda\left(xs : \mathrm{Str}^{\kappa}_{A}\right)\langle p\,(\mathrm{hd}^{\kappa}\,xs), \varphi \textcircled{κ}\,\mathrm{tl}^{\kappa}\,xs\rangle. \end{aligned}$$

7.C Example Programs with Coinductive Types

Let A be some small type in clock context Δ and κ, a fresh clock variable. Let $\mathrm{Str}_A = \forall\kappa.\,\mathrm{Str}^{\kappa}_{A}$. We can define head, tail and cons functions

$$\begin{aligned}
&\mathrm{hd} : \mathrm{Str}_A \to A && \mathrm{tl} : \mathrm{Str}_A \to \mathrm{Str}_A \\
&\mathrm{hd} \triangleq \lambda xs.\,\mathrm{hd}^{\kappa_0}\,(xs[\kappa_0]) && \mathrm{tl} \triangleq \lambda xs.\,\mathrm{prev}\,\kappa.\,\mathrm{tl}^{\kappa}\,(xs[\kappa])
\end{aligned}$$

$$\begin{aligned}
&\mathrm{cons} : A \to \mathrm{Str}_A \to \mathrm{Str}_A \\
&\mathrm{cons} \triangleq \lambda x.\lambda xs.\Lambda\kappa.\,\mathrm{cons}^{\kappa}\,x\,(\mathrm{next}^{\kappa}\,(xs[\kappa])).
\end{aligned}$$

With these we can define the *acausal* 'every other' function eo^{κ} that removes every second element of the input stream. This is acausal because the second element of the output stream is the third element of the input. Therefore to type the function we need to have the input stream always available, necessitating the use clock quantification. The function eo^{κ} is

$$\begin{aligned}
&\mathrm{eo}^{\kappa} : \mathrm{Str}_A \to \mathrm{Str}^{\kappa}_{A} \\
&\mathrm{eo}^{\kappa} \triangleq \mathrm{fix}^{\kappa}\,\varphi.\lambda\,(xs : \mathrm{Str}_A). \\
&\qquad\qquad\qquad \mathrm{cons}^{\kappa}(\mathrm{hd}\,xs)\,(\varphi \textcircled{κ}\,\mathrm{next}^{\kappa}\,((\mathrm{tl}\,(\mathrm{tl}\,xs)))).
\end{aligned}$$

i.e., we return the head immediately and then recursively call the function on the stream with the first two elements removed. Note that the result is a *guarded* stream, but we can easily strengthen it and define eo of type $\mathrm{Str}_A \to \mathrm{Str}_A$ as $\mathrm{eo} \triangleq \lambda xs.\Lambda\kappa.\,\mathrm{eo}^{\kappa}\,xs$.

A more interesting type is the type of covectors, which is a refinement of the guarded type of covectors defined in Section 7.3. First we define the type of co-natural numbers $\mathrm{Co}\mathbb{N}$ as

$$\mathrm{Co}\mathbb{N} = \forall\kappa.\,\mathrm{Co}\mathbb{N}^{\kappa}.$$

It is easy to define $\overline{0}$ and $\overline{\mathrm{succ}}$ as

$$\begin{aligned}
&\overline{0} : \mathrm{Co}\mathbb{N} && \overline{\mathrm{succ}} : \mathrm{Co}\mathbb{N} \to \mathrm{Co}\mathbb{N} \\
&\overline{0} \triangleq \Lambda\kappa.\,\mathrm{inl}\,\langle\rangle && \overline{\mathrm{succ}} \triangleq \lambda n.\Lambda\kappa.\,\mathrm{inr}\,(\mathrm{next}^{\kappa}\,(n[\kappa]))^{\cdot}
\end{aligned}$$

Next, we will use type isomorphisms to define a transport function $\mathrm{com}^{\mathrm{Co}\mathbb{N}}$ of type $\mathrm{com}^{\mathrm{Co}\mathbb{N}} : \mathrm{Co}\mathbb{N} \to 1 + \mathrm{Co}\mathbb{N}$ as

$$\begin{aligned}
\mathrm{com}^{\mathrm{Co}\mathbb{N}} \triangleq \lambda n.\,&\mathrm{case}\,\mathrm{com}^{+}\,n\,\mathrm{of} \\
&\mathrm{inl}\,u \Rightarrow \mathrm{inl}\,u[\kappa_0] \\
&\mathrm{inr}\,n \Rightarrow \mathrm{inr}\,\mathrm{prev}\,\kappa.n[\kappa]
\end{aligned}$$

This function satisfies term equalities

$$\mathrm{com}^{\mathrm{Co}\mathbb{N}}\,\overline{0} \equiv \mathrm{inl}\,\langle\rangle \qquad\qquad \mathrm{com}^{\mathrm{Co}\mathbb{N}}(\overline{\mathrm{succ}}\,n) \equiv \mathrm{inr}\,n. \tag{7.10}$$

Using this we can define type of covectors CoVec_A as

$$\mathrm{CoVec}_A\;n \triangleq \forall\kappa.\,\mathrm{CoVec}^{\kappa}_A\;n$$

where $\mathrm{CoVec}^{\kappa}_A : \mathrm{Co}\mathbb{N} \to \mathcal{U}_{\Delta,\kappa}$ is the term

$$\begin{aligned}\mathrm{fix}^{\kappa}\,\varphi.\lambda\,(n : \mathrm{Co}\mathbb{N}).\,\mathrm{case}\,\mathrm{com}^{\mathrm{Co}\mathbb{N}}\,n\,\mathrm{of}\;&\\ &\mathrm{inl}\,\_ \Rightarrow \widehat{1}\\ &\mathrm{inr}\,n \Rightarrow A\widehat{\times\triangleright}^{\kappa}\,(\varphi \circledast^{\kappa} (\mathrm{next}^{\kappa}\,n)).\end{aligned}$$

Notice the use of $\mathrm{com}^{\mathrm{Co}\mathbb{N}}$ to transport n of type $\mathrm{Co}\mathbb{N}$ to a term of type $1{+}\mathrm{Co}\mathbb{N}$ which we can case analyse. To see that this type satisfies the correct type equalities we need some auxiliary term equalities which follow from the way we have defined the terms.

Using term equalities (7.4) and (7.5) we can derive the (almost) expected type equalities

$$\begin{aligned}\mathrm{CoVec}_A\;\overline{0} &\equiv \forall\kappa.1\\ \mathrm{CoVec}_A\,(\overline{\mathrm{succ}}\,n) &\equiv \forall\kappa.\Big(A \times \overset{\kappa}{\triangleright}(\mathrm{CoVec}^{\kappa}\,n)\Big)\end{aligned} \tag{7.11}$$

and using the type isomorphisms we can extend these type equalities to type isomorphisms

$$\begin{aligned}\mathrm{CoVec}_A\;\overline{0} &\cong 1\\ \mathrm{CoVec}_A\,(\overline{\mathrm{succ}}\,n) &\cong A \times \mathrm{CoVec}_A\;n\end{aligned}$$

which are the expected type properties of the covector type.

A simple function we can define is the tail function

$$\begin{aligned}&\mathrm{tl} : \mathrm{CoVec}_A(\overline{\mathrm{succ}}\,n) \to \mathrm{CoVec}_A\\ &\mathrm{tl} \triangleq \lambda v.\,\mathrm{prev}\,\kappa.\pi_2\,(v[\kappa]).\end{aligned}$$

Note that we have used (7.11) to ensure that tl is type correct.

Next, we define the map function on covectors.

$$\begin{aligned}\mathrm{map} &: (A \to B) \to \Pi(n : \mathrm{Co}\mathbb{N}).\,\mathrm{CoVec}_A\,n \to \mathrm{CoVec}_B\,n\\ \mathrm{map}\,f &= \lambda n.\lambda xs.\Lambda\kappa.\,\mathrm{map}^{\kappa}\,f\;n\;(xs[\kappa])\end{aligned}$$

where map^{κ} is the function of type

$$\mathrm{map}^{\kappa} : (A \to B) \to \Pi(n : \mathrm{Co}\mathbb{N}).\,\mathrm{CoVec}^{\kappa}_A\,n \to \mathrm{CoVec}^{\kappa}_B\,n$$

defined as

$$\begin{aligned}\lambda f.\,\mathrm{fix}^{\kappa}\,\varphi.\lambda n.\,&\mathrm{case}\,\mathrm{com}^{\mathrm{Co}\mathbb{N}}\,n\,\mathrm{of}\\ &\mathrm{inl}\,\_\Rightarrow\lambda v.v\\ &\mathrm{inr}\,n\Rightarrow\lambda v.\langle f(\pi_1 v),\varphi\circledast^{\kappa}(\mathrm{next}^{\kappa}\,n)\circledast^{\kappa}\pi_2(v)\rangle.\end{aligned}$$

Let us see that this has the correct type. Let $D_A(x)$ (and analogously $D_B(x)$) be the type

$$D_A(x) \quad\triangleq\quad \begin{aligned}&\mathrm{case}\,x\,\mathrm{of}\\ &\mathrm{inl}\,\_\Rightarrow\widehat{1}\\ &\mathrm{inr}\,n\Rightarrow A\widehat{\times\triangleright}^{\kappa}\left(\left(\mathrm{next}^{\kappa}\,\mathrm{CoVec}_A^{\kappa}\right)\circledast^{\kappa}(\mathrm{next}^{\kappa}\,n)\right).\end{aligned}$$

where x is of type $1+\mathrm{Co}\mathbb{N}$. Using this abbreviation we can write the type of map^{κ} as

$$(A\to B)\to\Pi(n:\mathrm{Co}\mathbb{N}).D_A(\mathrm{com}^{\mathrm{Co}\mathbb{N}}\,n)\to D_B(\mathrm{com}^{\mathrm{Co}\mathbb{N}}\,n).$$

Using this it is straightforward to show, using the dependent elimination rule for sums, as we did in Section 7.3, that map^{κ} has the correct type. Indeed we have $D_A(\mathrm{inl}\,z)\equiv 1$ and $D_A(\mathrm{inr}\,n)\equiv A\times\overset{\kappa}{\triangleright}(\mathrm{CoVec}_A\,n)$.

7.D Type Isomorphisms in Detail

- If $\kappa\notin A$ then $\forall\kappa.A\cong A$. The terms are $\lambda x.x[\kappa_0]$ and $\lambda x.\Lambda\kappa.x$. The rule TmEq-∀-fresh is crucially needed to show that they constitute a type isomorphism.
- If $\kappa\notin A$ then $\forall\kappa.\Pi(x:A).B\cong\Pi(x:A).\forall\kappa.B$. The terms are

  $$\lambda z.\lambda x.\Lambda\kappa.z[\kappa]\,x$$

  of type $\forall\kappa.\Pi(x:A).B\to\Pi(x:A).\forall\kappa.B$ and

  $$\lambda z.\Lambda\kappa.\lambda x.(z\,x)[\kappa]$$

  of type $\Pi(x:A).\forall\kappa.B\to\forall\kappa.\Pi(x:A).B$.
- $\forall\kappa.\Sigma\,(x:A)\,B\cong\Sigma\,(y:\forall\kappa.A)\,(\forall\kappa.B\,[y[\kappa]/x])$. The terms are

  $$\lambda z.\langle\Lambda\kappa.\pi_1\,(z[\kappa]),\Lambda\kappa.\pi_2\,(z[\kappa])\rangle$$

  of type

  $$\forall\kappa.\Sigma\,(x:A)\,B\to\Sigma\,(y:\forall\kappa.A)\,(\forall\kappa.B\,[y[\kappa]/x])$$

  and

  $$\lambda z.\Lambda\kappa.\langle(\pi_1\,z)[\kappa],(\pi_2\,z)[\kappa]\rangle$$

  of the converse type.

- $\forall\kappa.A \cong \forall\kappa.\overset{\kappa}{\triangleright}A$. The terms are

  $$\lambda z.\Lambda\kappa.\,\mathrm{next}^{\kappa}(z[\kappa])$$

  of type $\forall\kappa.A \to \forall\kappa.\overset{\kappa}{\triangleright}A$ and

  $$\lambda z.\,\mathrm{prev}\,\kappa.\,(z[\kappa])$$

  of the converse type. The β and η rules for $\mathrm{prev}\,\kappa.$ ensure that this pair of functions constitutes an isomorphism.

Using these isomorphisms we can construct an additional type isomorphism witnessing that $\forall\kappa$ commutes with binary sums. Recall that we encode binary coproducts using Σ-types and universes in the standard way. Given two codes $\widehat{A}$ and $\widehat{B}$ in some universe $\mathcal{U}_\Delta$ we define

$$\begin{aligned}&\widehat{A}\widehat{+}\widehat{B} : \mathcal{U}_\Delta\\&\widehat{A}\widehat{+}\widehat{B} \triangleq \Sigma\,(b : \mathbf{B})\,\mathrm{if}\,b\,\mathrm{then}\,\widehat{A}\,\mathrm{else}\,\widehat{B}\end{aligned}$$

and we write $A+B$ for $\mathrm{El}\left(\widehat{A}\widehat{+}\widehat{B}\right)$. Suppose that $\Delta' \subseteq \Delta$ and κ is a clock variable not in Δ. Suppose that $\Gamma \vdash_\Delta$ and that we have two codes $\widehat{A}, \widehat{B}$ satisfying

$$\Gamma \vdash_{\Delta,\kappa} \widehat{A} : \mathcal{U}_{\Delta',\kappa} \qquad\qquad \Gamma \vdash_{\Delta,\kappa} \widehat{B} : \mathcal{U}_{\Delta',\kappa}$$

We start with an auxiliary function $\mathrm{com}^{\mathrm{if}}$. Let b be some term of type $\mathbf{B}$. We then define

$$\begin{aligned}\mathrm{com}^{\mathrm{if}}_b &: \forall\kappa.\,\mathrm{El}\left(\mathrm{if}\,b\,\mathrm{then}\,\widehat{A}\,\mathrm{else}\,\widehat{B}\right)\\&\qquad \to \mathrm{El}\left(\mathrm{if}\,b\,\mathrm{then}\,\widehat{\forall}\,\Lambda\kappa.\widehat{A}\,\mathrm{else}\,\widehat{\forall}\,\Lambda\kappa.\widehat{B}\right)\\\mathrm{com}^{\mathrm{if}}_b &\triangleq \mathrm{if}\,b\,\mathrm{then}\,\lambda x.x\,\mathrm{else}\,\lambda x.x\end{aligned}$$

which is typeable due to the strong elimination rule for $\mathbf{B}$.

We now define the function com^+

$$\begin{aligned}&\mathrm{com}^+ : \forall\kappa.(A+B) \to \forall\kappa.A + \forall\kappa.B\\&\mathrm{com}^+ \triangleq \lambda z.\left\langle \pi_1\,(z[\kappa_0]), \mathrm{com}^{\mathrm{if}}_{\pi_1(z[\kappa_0])}\,(\Lambda\kappa.\pi_2\,(z[\kappa]))\right\rangle.\end{aligned}$$

We need to check that the types are well-formed and the function well-typed. The side condition $\Gamma \vdash_\Delta$ ensures that the types are well-formed. To see that the function com^+ is well-typed we consider the types of subterms.

- The term z has type $\forall\kappa.(A+B)$.

- The term $\pi_1\,(z[\kappa_0])$ has type $\mathbf{B}$.

- The term $\Lambda\kappa.\pi_2\,(z[\kappa])$ has type

$$\forall\kappa.\,\mathrm{El}\Big(\mathrm{if}\,\pi_1\,(z[\kappa])\,\mathrm{then}\,\widehat{A}\,\mathrm{else}\,\widehat{B}\Big)$$

- From TmEq-∀-fresh we get $\pi_1(z[\kappa_0]) \equiv \pi_1(z[\kappa])$. Indeed, the term

$$\Lambda\kappa.\pi_1(z[\kappa])$$

  has type $\mathbf{B}$, which does not contain κ, and the required equality follows from TmEq-∀-fresh and the β rule for clock quantification.

- Thus the term $\Lambda\kappa.\pi_2\,(z[\kappa])$ has type

$$\forall\kappa.\,\mathrm{El}\Big(\mathrm{if}\,\pi_1\,(z[\kappa_0])\,\mathrm{then}\,\widehat{A}\,\mathrm{else}\,\widehat{B}\Big)$$

- And so the term

$$\mathrm{com}^{\mathrm{if}}_{\pi_1(z[\kappa_0])}\,\Lambda\kappa.\pi_2\,(z[\kappa])$$

  has type

$$\mathrm{El}\Big(\mathrm{if}\,\pi_1\,(z[\kappa_0])\,\mathrm{then}\,\widehat{\forall}\,\Lambda\kappa.\widehat{A}\,\mathrm{else}\,\widehat{\forall}\,\Lambda\kappa.\widehat{B}\Big)$$

  which is exactly the type needed to typecheck the whole term.

For the term com^+ we can derive the following definitional term equalities.

$$\begin{aligned}\mathrm{com}^+\,(\Lambda\kappa.\,\mathrm{inl}\,t) &\equiv \mathrm{inl}\,\Lambda\kappa.t\\ \mathrm{com}^+\,(\Lambda\kappa.\,\mathrm{inr}\,t) &\equiv \mathrm{inr}\,\Lambda\kappa.t\end{aligned}\tag{7.12}$$

There is also a canonical term of type

$$\forall\kappa.A + \forall\kappa.B \to \forall\kappa.(A+B)$$

defined as

$$\begin{aligned}&\lambda z.\Lambda\kappa.\,\mathrm{case}\,z\,\mathrm{of}\\ &\qquad\mathrm{inl}\,a \Rightarrow \mathrm{inl}\,(a[\kappa])\\ &\qquad\mathrm{inl}\,b \Rightarrow \mathrm{inl}\,(b[\kappa]).\end{aligned}$$

This term is inverse to com^+, although we require equality reflection to show that the two functions are inverses to each other. Without equality reflection we can only prove they are inverses up to propositional equality. The isomorphisms defined previously do not require equality reflection.

Bibliography

[1] Andreas Abel and Brigitte Pientka.
Wellfounded recursion with copatterns: A unified approach to termination and productivity.
In *ICFP*, pages 185–196, 2013.
28, 163, 263

[2] Andreas Abel and Andrea Vezzosi.
A formalized proof of strong normalization for guarded recursive types.
In *APLAS*, pages 140–158, 2014.
153, 163

[3] Gul Agha, Ian A. Mason, Scott F. Smith, and Carolyn L. Talcott.
A foundation for actor computation.
Journal of Functional Programming, 7(1):1–72, 1997.
80

[4] Amal Ahmed.
Semantics of Types for Mutable State.
PhD thesis, Princeton University, 2004.
5, 51

[5] Amal Ahmed.
Step-indexed syntactic logical relations for recursive and quantified types.
In *Proceedings of the 15th European Conference on Programming Languages and Systems*, pages 69–83. Springer-Verlag, 2006.
ISBN 3-540-33095-X, 978-3-540-33095-0.
4, 5, 36

[6] Amal Ahmed.
Step-indexed syntactic logical relations for recursive and quantified types.
Technical report, Harvard University, 2006.
URL http://www.ccs.neu.edu/home/amal/papers/lr-recquant-techrpt.pdf.
5, 87, 125

[7] Pierre America and Jan Rutten.
Solving reflexive domain equations in a category of complete metric spaces.
Journal of Computer and System Sciences, 39(3):343 – 375, 1989.
ISSN 0022-0000.
8

[8] Andrew W. Appel and David McAllester.
An indexed model of recursive types for foundational proof-carrying code.
ACM Transactions on Programming Languages and Systems, 23(5), 2001.
36

[9] Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon.
A very modal model of a modern, major, general type system.
In *POPL*, pages 109–122, 2007.
11, 148, 248, 253

[10] Krzysztof R. Apt and Gordon D. Plotkin.
Countable nondeterminism and random assignment.
Journal of the ACM, 33(4):724–767, 1986.
23, 80

[11] Robert Atkey and Conor McBride.
Productive coprogramming with guarded recursion.
In *ICFP*, pages 197–208, 2013.
19, 24, 25, 149, 152, 163, 169, 180, 181, 186, 220, 248, 256, 257, 262

[12] Steve Awodey.
Category Theory.
Oxford Logic Guides. Ebsco Publishing, 2006.
ISBN 9780191513824.
103

[13] A. Bauer and L. Birkedal.
W-Types and M-Types in Equilogical Spaces.
Manuscript. `http://users-cs.au.dk/birke/papers/w-m-types-in-equ.pdf`, May 1999.
241, 242

[14] Andrej Bauer, Lars Birkedal, and Dana S Scott.
Equilogical spaces.
Theoretical Computer Science, 315(1):35–59, 2004.
25, 220, 223, 243

[15] Michael Beeson.
Recursive models for constructive set theories.
Annals of Mathematical Logic, 23(2–3):127 – 178, 1982.
ISSN 0003-4843.
25

[16] Jean Bénabou.
Fibrations petites et localement petites.
C. R. Acad. Sci. Paris Sér. A-B, 281(21):Ai, A897–A900, 1975.
28, 216

[17] Gavin M. Bierman and Valeria de Paiva.
On an intuitionistic modal logic.
Studia Logica, 65(3):383–416, 2000.
149, 152, 153

[18] Lars Birkedal and Rasmus Ejlers Møgelberg.
Intensional type theory with guarded recursive types qua fixed points on universes.
In *LICS*, pages 213–222, 2013.
181, 200, 214, 253, 262

[19] Lars Birkedal, Jan Schwinghammer, and Kristian Støvring.
A metric model of lambda calculus with guarded recursion.
In *FICS*, pages 19–25, 2010.
19, 157

[20] Lars Birkedal, Kristian Støvring, and Jacob Thamsborg.
The category-theoretic solution of recursive metric-space equations.
Theoretical Computer Science, 411(47):4102 – 4122, 2010.
ISSN 0304-3975.
7, 8, 177

[21] Lars Birkedal, Bernhard Reus, Jan Schwinghammer, Kristian Støvring, Jacob Thamsborg, and Hongseok Yang.
Step-indexed kripke models over recursive worlds.
In *Proceedings of the 38th Symposium on Principles of Programming Languages*, pages 119–132. ACM, 2011.
44

[22] Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring.
First steps in synthetic guarded domain theory: step-indexing in the topos of trees.
Logical Methods in Computer Science, 8(4), 2012.
11, 12, 13, 28, 80, 82, 92, 95, 96, 114, 115, 149, 154, 156, 159, 160, 180, 181, 190, 193, 219, 220, 222, 248, 261, 262

[23] Lars Birkedal, Aleš Bizjak, and Jan Schwinghammer.
Step-indexed relational reasoning for countable nondeterminism.
Logical Methods in Computer Science, 9(4), 2013.
22, 30, 41, 80, 81, 85, 86, 87, 93, 97, 123, 125, 126, 137, 140, 144

[24] Aleš Bizjak and Lars Birkedal.
A model of guarded recursion via generalised equilogical spaces.
???, 2015.
Under review.
24

[25] Aleš Bizjak and Lars Birkedal.
Step-indexed logical relations for probability.
In Andrew Pitts, editor, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pages 279–294. Springer-VS, 2015.
21, 29

[26] Aleš Bizjak, Lars Birkedal, and Marino Miculan.
A model of countable nondeterminism in guarded type theory.
In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi*, volume 8560 of *Lecture Notes in Computer Science*, pages 108–123. Springer International Publishing, 2014.
ISBN 978-3-319-08917-1.
21, 22, 23, 29, 159, 160

[27] Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus Ejlers Møgelberg, and Lars Birkedal.
Guarded dependent type theory with coinductive types.
In *Foundations of Software Science and Computation Structures - 19th International Conference, FoSSaCS 2016*, pages ??–??, 2016.
To appear (Accepted for publication).
24, 25, 30, 220, 221, 222, 237

[28] Aleš Bizjak and Rasmus Ejlers Møgelberg.
A model of guarded recursion with clock synchronisation.
Electronic Notes in Theoretical Computer Science, 319:83 – 101, 2015.
The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).
20, 24, 25, 26, 29, 220, 242, 248, 249, 256, 261, 262

[29] Edwin Brady.
Idris, a general-purpose dependently typed programming language: Design and implementation.
J. Funct. Programming, 23(5):552–593, 2013.
248

[30] Ranald Clouston and Rajeev Goré.
Sequent calculus in the topos of trees.
In *FoSSaCS*, 2015.
159

[31] Ranald Clouston, Aleš Bizjak, Hans Bugge Grathwohl, and Lars Birkedal.
Programming and reasoning with guarded recursion for coinductive typeso.
In *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015*, pages 407–421, 2015.
21, 23, 29, 257, 262

[32] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg.
Cubical type theory: a constructive interpretation of the univalence axiom.
Under submission, available at http://www.math.ias.edu/~amortberg/papers/cubicaltt.pdf, 2015.
263

[33] R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith.
Implementing Mathematics with the Nuprl Proof Development System.
Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.
ISBN 0-13-451832-2.
248

[34] Thierry Coquand.
Infinite objects in type theory.
In *TYPES*, pages 62–78, 1993.
3, 148, 248

[35] Raphaëlle Crubillé and Ugo Dal Lago.
On probabilistic applicative bisimulation and call-by-value λ-calculi.
In Zhong Shao, editor, *Programming Languages and Systems*, volume 8410 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 2014.
ISBN 978-3-642-54832-1.
22, 36, 37

[36] Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti.
On coinductive equivalences for higher-order probabilistic functional programs.
In *Proceedings of 41st Symposium on Principles of Programming Languages*, pages 297–308. ACM, 2014.

22, 36, 37

[37] Vincent Danos and Russell S. Harmer.
Probabilistic game semantics.
ACM Transactions on Computational Logic, 3(3), 2002.
36

[38] Pietro Di Gianantonio, Furio Honsell, and Gordon D. Plotkin.
Uncountable limits and the lambda calculus.
Nordic Journal of Computing, 2(2):126–145, 1995.
23, 80

[39] Derek Dreyer, Amal Ahmed, and Lars Birkedal.
Logical step-indexed logical relations.
Logical Methods in Computer Science, 7(2), 2011.
4, 40, 43, 80, 93

[40] Derek Dreyer, Georg Neis, and Lars Birkedal.
The impact of higher-order state and control effects on local relational reasoning.
Journal of Functional Programming, 22(4-5 special issue):477–528, 2012.
doi: http://dx.doi.org/10.1017/S0956796812000287.
4, 51

[41] Thomas Ehrhard, Michele Pagani, and Christine Tasson.
The computational meaning of probabilistic coherence spaces.
In *Proceedings of the 26th IEEE Symposium on Logic in Computer Science*, pages 87–96. IEEE, 2011.
ISBN 978-0-7695-4412-0.
36

[42] Thomas Ehrhard, Christine Tasson, and Michele Pagani.
Probabilistic coherence spaces are fully abstract for probabilistic pcf.
In *Proceedings of 41st Symposium on Principles of Programming Languages*, pages 309–320. ACM, 2014.
ISBN 978-1-4503-2544-8.
36

[43] Jörg Endrullis, Clemens Grabmayer, and Dimitri Hendriks.
Mix-automatic sequences.
Fields Workshop on Combinatorics on Words, contributed talk., 2013.
148

[44] Pietro Di Gianantonio and Marino Miculan.
A unifying approach to recursive and co-recursive definitions.

In *Types for Proofs and Programs, Second International Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24-28, 2002, Selected Papers*, pages 148–161, 2002.
244

[45] Eduarde Giménez.
Codifying guarded definitions with recursive schemes.
In *TYPES*, pages 39–59, 1995.
148, 248

[46] Jean-Yves Girard, Paul Taylor, and Yves Lafont.
Proofs and Types.
Cambridge University Press, New York, NY, USA, 1989.
ISBN 0-521-37181-3.
181

[47] Martin Hofmann.
On the interpretation of type theory in locally cartesian closed categories.
In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic*, volume 933 of *Lecture Notes in Computer Science*, pages 427–441. Springer Berlin Heidelberg, 1995.
27

[48] Martin Hofmann.
Syntax and semantics of dependent types.
In Andrew M. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, pages 79–130. Cambridge University Press, 1997.
ISBN 9780511526619.
Cambridge Books Online.
27, 215

[49] Martin Hofmann and Thomas Streicher.
Lifting Grothendieck universes.
Unpublished, 1999.
URL www.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf.
200, 202

[50] John Hughes, Lars Pareto, and Amr Sabry.
Proving the correctness of reactive systems using sized types.
In *POPL*, pages 410–423, 1996.
28, 163, 263

[51] Bart Jacobs.
Categorical Logic and Type Theory.

Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.
14, 24, 25, 27, 28, 182, 183, 186, 215, 216, 221, 223, 229, 230, 233, 238, 250, 254

[52] Patricia Johann, Alex Simpson, and Janis Voigtländer.
A generic operational metatheory for algebraic effects.
In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science*, pages 209–218. IEEE, 2010.
36

[53] Peter T. Johnstone.
Sketches of an elephant: a topos theory compendium. Vol. 1, volume 43 of *Oxford Logic Guides*.
The Clarendon Press Oxford University Press, New York, 2002.
ISBN 0-19-853425-6.
100, 101, 102, 186

[54] Claire Jones and Gordon Plotkin.
A probabilistic powerdomain of evaluations.
In *Proceedings of the 4th Symposium on Logic in Computer Science*, pages 186–195. IEEE, 1989.
36

[55] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer.
Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning.
In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 637–650, New York, NY, USA, 2015. ACM.
ISBN 978-1-4503-3300-9.
4, 14

[56] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky.
The simplicial model of univalent foundations.
Preprint, `http://arxiv.org/abs/1211.2851`, 2012.
27

[57] Neelakantan R. Krishnaswami and Nick Benton.
Ultrametric semantics of reactive programs.
In *LICS*, pages 257–266, 2011.
149, 163, 248

[58] Ugo Dal Lago and Margherita Zorzi.
Probabilistic operational semantics for the lambda calculus.

RAIRO - Theoretical Informatics and Applications, 46, 2012.
36

[59] J. Lambek and P.J. Scott.
Introduction to Higher-Order Categorical Logic.
Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1988.
ISBN 9780521356534.
11, 95

[60] Søren B. Lassen.
Relational Reasoning about Functions and Nondeterminism.
PhD thesis, University of Aarhus, 1998.
80

[61] Søren B. Lassen and Andrew Moran.
Unique fixed point induction for McCarthy's amb.
In *Mathematical Foundations of Computer Science*, pages 198–208, 1999.

[62] Søren B. Lassen and Corin Pitcher.
Similarity and bisimilarity for countable non-determinism and higher-order functions.
Electronic Notes in Theoretical Computer Science, 10, 1997.

[63] Paul Blain Levy.
Infinitary Howe's method.
In *Coalgebraic Methods in Computer Science*, pages 85–104, 2006.
80

[64] Peter Lefanu Lumsdaine and Michael A. Warren.
The local universes model: An overlooked coherence construction for dependent type theories.
ACM Transactions on Computation Logic, 16(3):23:1–23:31, July 2015.
ISSN 1529-3785.
27

[65] Saunders MacLane.
Categories for the Working Mathematician.
Graduate Texts in Mathematics. Springer New York, second edition, 1998.
ISBN 9780387984032.
196

[66] Saunders MacLane and Ieke Moerdijk.
Sheaves in Geometry and Logic: A First Introduction to Topos Theory.
Mathematical Sciences Research Institute Publications. Springer New York, 1992.

ISBN 9780387977102.
11, 94, 95, 100, 101, 112, 185, 194

[67] Ian A. Mason and Carolyn L. Talcott.
Equivalence in functional languages with effects.
Journal of Functional Programming, 1(3):287–327, 1991.
93, 125

[68] The Coq development team.
The Coq proof assistant reference manual.
LogiCal Project, 2004.
URL http://coq.inria.fr.
Version 8.0.
3, 14, 180, 248

[69] Conor McBride and Ross Paterson.
Applicative programming with effects.
J. Funct. Programming, 18(1):1–13, 2008.
150, 237, 248, 251, 263

[70] Stefan Milius, Lawrence S. Moss, and Daniel Schwencke.
Abstract GSOS rules and a modular treatment of recursive definitions.
LMCS, 9(3), 2013.
163, 244

[71] Rasmus Ejlers Møgelberg.
A type theory for productive coprogramming via guarded recursion.
In *CSL-LICS*, 2014.
24, 25, 156, 163, 180, 181, 186, 200, 207, 220, 242, 248, 249, 256, 257, 262

[72] Hiroshi Nakano.
A modality for recursion.
In *LICS*, pages 255–266, 2000.
19, 148, 163, 180, 248

[73] Ulf Norell.
Towards a practical programming language based on dependent type theory.
PhD thesis, Chalmers University of Technology, 2007.
180, 248

[74] Marco Paviotti, Rasmus Ejlers Møgelberg, and Lars Birkedal.
A model of PCF in guarded type theory.
In *MFPS*, 2015.
249

[75] Benjamin C. Pierce.
Types and Programming Languages.
MIT Press, Cambridge, MA, USA, 2002.
ISBN 0-262-16209-1.
4

[76] Andrew M. Pitts.
Parametric polymorphism and operational equivalence.
Mathematical Structures in Computer Science, 10(3), 2000.
36

[77] Andrew M. Pitts.
Categorical logic.
In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures*, chapter 2, pages 39–128. Oxford University Press, 2000.
ISBN 0-19-853781-6.
215

[78] Andrew M. Pitts.
Typed operational reasoning.
In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 7. MIT Press, 2005.
4, 42, 43, 47

[79] Andrew M. Pitts.
Step-indexed biorthogonality: a tutorial example.
In Amal Ahmed, Nick Benton, Lars Birkedal, and Martin Hofmann, editors, *Modelling, Controlling and Reasoning About State*, number 10351 in Dagstuhl Seminar Proceedings, 2010.
36

[80] Dag Prawitz.
Natural Deduction: A Proof-Theoretical Study.
Dover Publ., 1965.
152

[81] Norman Ramsey and Avi Pfeffer.
Stochastic lambda calculus and monads of probability distributions.
In *Proceedings of the 29th Symposium on Principles of Programming Languages*, pages 154–165. ACM, 2002.
36

[82] Gonzalo E. Reyes and Houman Zolfaghari.
Bi-heyting algebras, toposes and modalities.
Journal of Philosophical Logic, 25(1):25–43, 1996.

ISSN 0022-3611.
97, 98, 105, 109

[83] J. J. M. M. Rutten.
Behavioural differential equations: A coinductive calculus of streams, automata, and power series.
Theor. Comput. Sci., 308(1–3):1–53, 2003.
ISSN 0304-3975.
23, 149, 162, 244

[84] David Sabel and Manfred Schmidt-Schauß.
A call-by-need lambda calculus with locally bottom-avoiding choice: context lemma and correctness of transformations.
Mathematical Structures in Computer Science, 18(3):501–553, 2008.
80

[85] N. Saheb-Djahromi.
Cpo's of measures for nondeterminism.
Theoretical Computer Science, 12(1), 1980.
36

[86] Davide Sangiorgi and Valeria Vignudelli.
Environmental bisimulations for probabilistic higher-order languages.
In *Proceedings of POPL'16*, 2016.
To appear.
26

[87] Dana Scott.
Continuous lattices.
In F.W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lecture Notes in Mathematics*, pages 97–136. Springer Berlin Heidelberg, 1972.
ISBN 978-3-540-05920-2.
238

[88] Paula G Severi and Fer-Jan J de Vries.
Pure type systems with corecursion on streams: from finite to infinitary normalisation.
In *ICFP*, pages 141–152, 2012.
163

[89] Filip Sieczkowski, Aleš Bizjak, and Lars Birkedal.
ModuRes: A Coq library for modular reasoning about concurrent higher-order imperative programming languages.
In *Interactive Theorem Proving*, volume 9236 of *Lecture Notes in Computer Science*, pages 375–390. Springer International Publishing, 2015.
30

[90] Michael B Smyth and Gordon D Plotkin.
The category-theoretic solution of recursive domain equations.
SIAM Journal on Computing, 11(4):761–783, 1982.
6, 7

[91] Kasper Svendsen and Lars Birkedal.
Impredicative concurrent abstract predicates.
In Zhong Shao, editor, *Programming Languages and Systems*, volume 8410 of *Lecture Notes in Computer Science*, pages 149–168. Springer Berlin Heidelberg, 2014.
ISBN 978-3-642-54832-1.
4, 28, 219, 263

[92] The Univalent Foundations Program.
Homotopy Type Theory: Univalent Foundations of Mathematics.
http://homotopytypetheory.org/book, Institute for Advanced Study, 2013.
254

[93] Aaron Turon, Derek Dreyer, and Lars Birkedal.
Unifying refinement and hoare-style reasoning in a logic for higher-order concurrency.
In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pages 377–390, New York, NY, USA, 2013. ACM.
ISBN 978-1-4503-2326-0.
4, 10

[94] N. H. Williams.
On grothendieck universes.
Compositio Mathematica, 21(1):1–3, 1969.
201