Uniform Operational Semantics
And combined probabilistic/non-deterministic choice

Aliaume Lopez
M1 student in Computer Science,
École Normale Supérieure Paris-Saclay,
Université Paris-Saclay, France
aliaume.lopez@ens-paris-saclay.fr

Under the supervision of Pr. Alex Simpson
Faculty of Mathematics and Physics
University of Ljubljana, Slovenia

From the 20/02/2017 to the 07/07/2017

Abstract

The goal of this internship was to give a systematic study of contextual equivalence in the presence of algebraic effects in a uniform, compact and syntactic way, in the continuation of pre-existing work [13]. To reach this objective, we extended a call-by-value PCF with a signature of algebraic effects, and gave an abstract equality between the contextual equivalence for this language and a usable logical relation, independently of the signature itself. Generic meta theorems were then proven using this abstract equivalence. To justify the usefulness of this approach, a direct link with denotational semantics and the work of Plotkin and Power [22] was developed, and signature of mixed non-determinism and probabilities was studied in-depth.

I am grateful to Alex Simpson, who allowed me to have an amazing internship in his lab
I would also like to thank Andrej Bauer for the Slovene support and his great sense of humor
Many thanks to Philip, Niels, Bret, Riccardo, Théo and Pierre-Marie
with whom I discovered Ljubljana's bars
But this could not have been possible without the help of Jean Goubault-Larrecq …

1 Preliminaries

1.1 University of Ljubljana (FMF)

My internship took place in Ljubljana the capital city of Slovenia. The Faculty of Mathematics and Physics (FMF) of the University of Ljubljana is the main university in Slovenia for mathematical studies. Located in the southern part of the town, it is accessible by bus and bike, and is at a walking distance from the center of Ljubljana.

1.2 Research environment

My research supervisor was Alex Simpson who is currently a "Professor of Computer Science" at the Faculty of Mathematics and Physics (FMF) of Ljubljana. Even if the FMF is not a faculty of computer science, Alex Simpson is well known in the theoretical area of this discipline and presented two talks during the CALCO/MFPS conference in Ljubljana on June 2017. This year he was supervising Niels Vorneeveld, a PhD student, working on effects inside programming languages. The university was able to give me a desk from the first day, and when Théo Winterhalter — another ENS Paris-Saclay student — left Ljubljana at the end of his internship, I was able to move to the office of the PhD students.

The group of PhD students, containing both theoretical computer science and complex analysis research, was very friendly. Once or twice a week, they would go to the climbing gym, and on a regular basis having beers in the town center. Every working day we had meals with Andrej Bauer and Alex Simpson, two teachers and supervisors of some PhD students of the group at the local restaurant.

I attended the weekly seminars on "Foundations of mathematics and theoretical computer science" with the rest of the group. I gave one seminar myself in a joint seminar with Niels, where we presented the recent work of Dal Lago, Gavazzo and Levy [8] on abstract bisimulations for a parametrized class of programming languages. I also attended with Niels Voorneveld and Philipp Haselwarter a reading course given by Dr. Mtija Pretnar for PhD students on operational semantics. The objective of this course was to understand a paper by Andrew Pitts [21], which was tightly related to the technical tools I was using in my research.

I would also like to mention the presence of Pierre-Marie Pédrot, a former student from ENS Lyon who was also part of this cohesive group, while not being a PhD student anymore (post-doc).

On top of this ideal setting for research, Ljubljana was hosting the seventh Conference on Algebra and Coalgebra in Computer Science (CALCO) and Mathematical Foundations of Programming Semantics (MFPS) the same week in the building of the FMF. This was a wonderful week because I had the opportunity to see the ongoing work in the area I was working on, meet and befriend researchers in this domain of computer science. The complete list of participants and the abstracts of the talks are available online at `http://coalg.org/mfps-calco2017/`.

1.3 Research subject

Even before it started, the internship was tied to an article published in 2010 by Alex Simpson and two other authors [13]. The paper itself was a step towards having a "General Operational Metatheory" for calculi with effects, a project that can be linked to the theory of algebraic effects of Plotkin and Power, and from which some technicalities were borrowed, for instance from Adequacy for Algebraic Effects [22].

The goal of the internship was very simple: adapt the previous results from a call-by-name to a call-by-value setting. It was justified by the fact that it is very unnatural to consider effects in a programming language where the evaluation order of said effects cannot be determined easily. For instance, consider the two following programs:

(i) A function that returns its input with probability one half, and returns one with probability one half: `function x -> pr (x,1)`

(ii) A function that is the identity with probability one half, and the constant function equal to one with probability one half: `pr (function x -> x, function x -> 1)`

They represent two functions that give the same result $\mathrm{pr}(x, 1)$ when applied to the same input x. This implies that in a call-by-name evaluation setting they are considered equivalent [13]. However, it is natural to distinguish the two in a regular programming language: the program (i) represents a function that is going to toss a coin *every time it is called* whereas the program (ii) represents a *unique* toss of coin to decide to behave as the constant function equal to 1 or as the identity function.

In addition to this main goal, I was given multiple other tasks. The first one was to tie the results to the denotational setting, in order to make a clear connection between the two approaches to contextual equivalence. Indeed, a parallel approach to contextual equivalence is to consider a denotational interpretation: a lot of work has been done to link the two, leading to well-known properties such as *full-abstraction* [23]. It is clear that being able to reuse results from this field can be useful. The second one was to consider a non-trivial and well-studied combination of two effects: non-determinism and probability. This combination is known to be challenging [17] and being able to consider it inside our restricted setting is a concrete proof of its expressive power.

The restriction in the number of pages prevents the inclusion of all the proofs, and therefore a notation $\star$, $\star\star$, $\star\star\star$, $\star\star\star\star$ is used to estimate the difficulty/time-spent on the different results. This allows to highlight the problematic parts and differentiate them from the "routine" results without including the full proofs. However this internship has given me the opportunity to prepare a paper that is going to be submitted to FOSSACS in 2018, a draft containing the proofs can be found in appendix.

2 Introduction

Contextual equivalence in the style of Morris has imposed itself as a very simple and powerful way to express what an equivalence on programs should be. In the presence of a typed calculus is generally defined as follows:

> Two terms M and M' of type τ are contextually equivalent if and only if for all context[1] C such that $C[M]$ and $C[M']$ are terms of a *simple* type, the observations that can be made on $C[M]$ are the same as the one that can be made on $C[M']$.

The idea is simple, instead of defining a relation on terms directly, one defines *observations* that can be made on terms of a simple type. For instance, the classical contextual equivalence for PCF is defined using natural numbers as the "simple type" and *termination* as the only observation [23]. This way, two complex programs are equivalent if and only if they give the same observations when plugged into a bigger one whose behaviour is easy to define. This equivalence is of interest because it precisely captures when one program can be substituted by another, which gives a lot of information on the language itself, but is also necessary when doing optimisation: replacing a piece of code by a supposedly faster one *without changing the meaning of the overall program*.

However this operational notion of equivalence is rarely usable as-is [19]. For this reason, many other forms of equivalences have been developed in the past years: bisimulations and their refinements (environmental bisimulations, bisimulations up-to) [15], game semantics [2], denotational interpretation in domains [26], higher order logic on programs [12] and logical relations [21]. Unsurprisingly one can observe extreme variations on the complexity of such methods when changing the class of effects studied. This has to do with the fact that most of the operational semantics are deeply

[1] A context is a program with one hole

tied with the effects of the language, and adding or removing effects changes the overall shape of the semantics. For denotational interpretation, the problem is less visible because it is transposed into domain theoretic constructions, such as powerdomains, distributive laws, and solving domain equations.

The objective is to apply one of the former method that are useful to study contextual equivalence of terms, but apply it *uniformly* across a wide class of languages. This allows, for instance, to *extend* a language that is being studied with new effects, without having to redo all the previous work on contextual equivalence. Following the work done by Patricia Johann, Alex Simpson and Janis Voigtländer [13], we choose to use a *logical relation* to capture the contextual equivalence. The class of languages that is studied is a simply typed lambda-calculus with a parameter Σ, a list of the *effect* constructions that are added. Using this, we derive theorems that apply *uniformly* across this set of languages. Examples of languages that can be found in this set are numerous: non-determinism, probabilities, input-output, and exceptions.

However, the work done by Johann et al. takes as basic language a *call-by-name* lambda-calculus. This does not correspond to the large majority of existing calculus and real programming languages that are in *call-by-value*. This is an issue because the two following programs are considered contextually equivalent in a *call-by-name* setting, while obviously[2] are not in a *call-by-value* one: (a) A program of type Nat $\to$ Nat that does not terminate (b) A program of type Nat $\to$ Nat that does not terminate when applied to a value . This example shows that the contextual equivalence is not the same in both cases, and theorems have to be adapted.

The report is answering the following goals:

I. **Adapt** the paper from Johann et al. to a *call-by-value* setting. This is the first part of this report, and consists in several stages:
   (a) The definition of the parametrized class of languages (syntax, type system, etc.)
   (b) The definition of a small-step semantics for the *base* language (ignoring the effects). In order to avoid having to define a semantics for the effects, the operational semantics is building *trees* where nodes are (uninterpreted) effects, and leaves are values of the base language
   (c) The definition of contextual equivalence in this setting. The definition is parametrized both by Σ (the list of effects) and a preorder $\sqsubseteq_b$ on trees of natural numbers with nodes labelled by elements of Σ
   (d) The construction of the logical relation, and the proof that the logical relation captures the contextual equivalence, everything still parametrized by Σ and $\sqsubseteq_b$, justifying the *uniformity* property of the proof.

II. **Continue** the work done by Plotkin and Power [22] by using their results to link our setting to the denotational approach of contextual equivalence. This is done by first relating a denotational interpretation of our language to the preorders $\sqsubseteq_b$ on trees that can be built, and then using this result to prove that both approaches to contextual equivalence coincide

III. **Compare** different ways of using the global method on specific examples: angelic non-determinism, demonic non-determinism and probabilistic choice. The idea is to compare the different ways of defining the preorder $\sqsubseteq_b$, and prove that they coincide. The definitions of $\sqsubseteq_b$ can be obtained in several ways: (i) using a denotational interpretation (ii) freely generating it from an inequational theory (iii) defining it operationally on trees

IV. **Study** the concrete example of a programming language with both probabilistic and non-deterministic choice, thus extending the previous comparison work to a non-trivial case. This will demonstrate the applicability of the general method, and is interesting because this is a combination of effects regularly encountered in denotational models [27] [10] [14] and the study of concurrent programming languages [17] [18]

[2] One is a non terminating term, and the other one is a terminating one

3 Parametrized class of languages

In order to derive *generic* theorems and have *uniform* results across different languages, we are going to consider a parametrized class of languages. The studied class of languages is based on PCF [23] in a call-by-value setting parametrized by extra operations called *effects*, which is a setting very similar to the one studied by Plotikn and Power [22]. The main motivation is to find result similar to the ones we already have in a call-by-name setting [13].

The parameter for this class of languages is Σ, a set of symbols with a finite arity. Refined call-by-value [8] is used instead of regular call-by-value to help separating effect evaluation and regular term evaluation in the language. However the results apply equally to both languages because there is a direct translation from one to the other that can be seen in Figure 2. Moreover, the parameter Σ is enough to consider a lot of different effects: non-deterministic choice, probabilistic choice, input-output, global state storing natural numbers, exceptions [13].

The type system is fairly simple, with only natural numbers and functions from one type to another type as it can be seen in Figure 1. The type inference rules for this language can be seen in Figure 3, no rule should be surprising but it is worth mentioning that already the inference rules are making a distinction between *computations* (terms constructed with the rules for M in Figure 1) and *values* (terms constructed with V in Figure 1). Moreover, natural numbers are encoded in *unary* using the usual Z and S constructions (zero and successor).

This setting is purposely restricted, and several improvements can be added without technical issue. For instance, more complex types such as sum, products and even type polymorphism can be studied in this context. In fact, logical relations excel in proving parametricity results [28]. In the spirit of simplicity and to allow comparison with the work on bisimulations by Ugo Dal Lago, Francesco Gavazzo and Paul Blain Lévi [8] we take the same kind of effect signature as they do. Technically, it means that compared to the paper from Johann et al. [13] it lacks three of the four effect constructions, but as noticed in the said paper, all of the constructions share the same pattern of proof, so that they actually treated only one of the four cases in their proofs.

It is worth mentioning that this is a setting very similar to the one of the Eff programming language [4], where the programmer can create *new* effects and describe their semantics withing the language itself: the parameter Σ is this case an *open* enumeration of effects, and can be extended by the user.

$$
\begin{aligned}
\tau &:= \mathsf{Nat} \mid \tau \to \tau \\
V &:= x \mid \lambda x : \tau . M \mid \mathsf{Z} \mid \mathrm{S}\, V \\
M &:= \mathrm{return}\, V \mid VV \mid \mathrm{fix}\, V \\
&\quad \mid \mathrm{case}\, V \,\mathrm{of}\, \mathsf{Z} \Rightarrow M; \mathrm{S}(x) \Rightarrow M \\
&\quad \mid \mathrm{let}\, x : \tau \Leftarrow M \,\mathrm{in}\, M \\
&\quad \mid \sigma(\underbrace{M, \ldots, M}_{n}) \quad \text{where } (\sigma, n) \in \Sigma
\end{aligned}
$$

Figure 1: Refined Call-By-Value PCF with effects

$$
\begin{aligned}
(\lambda x : \tau . N)M &\leftarrow \mathrm{let}\, x : \tau \Leftarrow M \,\mathrm{in}\, N \\
MN &\to \mathrm{let}\, f : \sigma \to \tau \Leftarrow M \,\mathrm{in}\, \mathrm{let}\, x : \sigma \Leftarrow N \,\mathrm{in}\, fx
\end{aligned}
$$

Figure 2: Translation between refined and regular call-by-value

$$\frac{}{\Gamma, x : \tau \vdash_V x : \tau} \qquad \frac{\Gamma \vdash_V V : \tau}{\Gamma \vdash_C \mathrm{return}\, V : \tau} \qquad \frac{\Gamma, x : \tau \vdash_C M : \tau'}{\Gamma \vdash_V \lambda x : \tau . M : \tau \to \tau'} \qquad \frac{}{\Gamma \vdash_V \mathrm{Z} : \mathrm{Nat}}$$

$$\frac{\Gamma \vdash_V V : \mathrm{Nat}}{\Gamma \vdash_V \mathrm{S}\, V : \mathrm{Nat}} \qquad \frac{\Gamma \vdash_V V : (\tau \to \tau') \to \tau \to \tau'}{\Gamma \vdash_C \mathrm{fix}\, V : \tau \to \tau'} \qquad \frac{\Gamma \vdash_V V : \tau \to \tau' \qquad \Gamma \vdash_V W : \tau}{\Gamma \vdash_C VW : \tau'}$$

$$\frac{\Gamma \vdash_V V : \mathrm{Nat} \qquad \Gamma \vdash_C M_1 : \tau \qquad \Gamma, x : \mathrm{Nat} \vdash_C M_2 : \tau}{\Gamma \vdash_C \mathrm{case}\, V \,\mathrm{of}\, \mathrm{Z} \Rightarrow M_1; \mathrm{S}(x) \Rightarrow M_2 : \tau} \qquad \frac{\Gamma \vdash_C M : \tau \qquad \Gamma, x : \tau \vdash_C N : \tau'}{\Gamma \vdash_C \mathrm{let}\, x : \tau \Leftarrow M \,\mathrm{in}\, N : \tau'}$$

$$\frac{(\sigma, n) \in \Sigma \qquad \forall 1 \leq i \leq n, \quad \Gamma \vdash_C M_i : \tau}{\Gamma \vdash_C \sigma(M_1, \ldots, M_n) : \tau}$$

Figure 3: Inference rules for typing

Example 1 (Signature for combined probabilities and non-determinism)**.** In the case of combining non-determinism and probabilities, one can consider a fair coin toss pr of arity two, with a demonic choice operator or of arity two. The signature of the language is therefore $\Sigma = \{(\mathrm{pr}, 2), (\mathrm{or}, 2)\}$.

It can be shown that the set of values of type Nat is isomorphic to $\mathbb{N}$ and therefore we will identify the two sets using $\underline{n}$ to denote the value corresponding to n.

3.1 Uniform Small-Step Semantics

The first step to define contextual equivalence is to define an operational semantics. Because we are considering a *class* of languages we are going to have a two steps approach: first interpret the core language independently of the parameter Σ and only then *refine* this semantics to give one to the effects. The small-step operational semantics is given using stack and frames [13]. The link between this small-step semantics and its big-step counterpart is made both in [22] and in [21]. This method can be traced back to [3] (1998) page 184 and [9].

$$E := \mathrm{let}\, x : \tau \Leftarrow \Box \,\mathrm{in}\, M$$
$$S := Id \mid S \circ E$$

Figure 4: Stacks and Frames

The idea of such an operational semantics is to explicitly refer to the *evaluation stack* in order to simplify the evaluation process. When evaluating a term, it is sometimes needed to remember the context in which the evaluation is taking place: this is exactly what the stack is, the list of all evaluations that have been suspended so far. However, we are going to see that stacks are not just a convenient tool to encode evaluation, and their introduction is crucial to the next developments. In a refined call-by-value calculus stacks and frames are very simple as it can be seen in Figure 4, but few cases does not imply little expressive power, and in fact any term with one free variable can be turned into a stack. A type system for stacks is also defined Figure 5, where a stack expecting a term of type τ and outputting a term of type τ' is given the type $\tau \multimap \tau'$.

Because stacks are just a `let`-binding with one hole, we can define application of a stack to a *computation* which returns a computation obtained by substitution as defined in Figure 6. We say that a pair (S, M) of a stack and a term is well typed when $S : \sigma \multimap \tau$ and $M : \sigma$, this will be consistent with the safety Lemma 1.

Now that evaluation stacks are defined, we can use them to define the operational semantics of our language. First of all, on simple computations, we can define a term reduction as in Figure 7,

then given this basic reduction, we can use stacks to generalise the evaluation process to more terms as seen in Figure 8.

$$\frac{x : \tau \vdash_C M : \tau'}{\vdash \mathrm{let}\, x : \tau \Leftarrow \Box \,\mathrm{in}\, M : \tau \multimap \tau'} \qquad \frac{\vdash E : \tau \multimap \tau' \qquad \vdash S : \tau' \multimap \tau''}{\vdash S \circ E : \tau \multimap \tau''}$$

$$\frac{}{\vdash Id : \tau \multimap \tau}$$

Figure 5: Typing of Stacks and Frames

$$\begin{aligned} Id\{M\} &= M \\ (S \circ E)\{M\} &= S\{E\{M\}\} \\ (\mathrm{let}\, x : \tau \Leftarrow \Box \,\mathrm{in}\, N\{M\}) &= \mathrm{let}\, x : \tau \Leftarrow M \,\mathrm{in}\, N \end{aligned}$$

Figure 6: Stack application

$$\begin{array}{lcl} (\lambda x : \tau.M)V & \rightsquigarrow & M[x := V] \\ \mathrm{case}\,\mathrm{Z}\,\mathrm{of}\ \mathrm{Z} \Rightarrow M_1; \mathrm{S}(x) \Rightarrow M_2 & \rightsquigarrow & M_1 \\ \mathrm{case}\,\mathrm{S}(V)\,\mathrm{of}\ \mathrm{Z} \Rightarrow M_1; \mathrm{S}(x) \Rightarrow M_2 & \rightsquigarrow & M_2[x := V] \\ \mathrm{let}\, x : \tau \Leftarrow \mathrm{return}\, V \,\mathrm{in}\, M & \rightsquigarrow & M[x := V] \\ \mathrm{fix}\, V & \rightsquigarrow & V(\lambda x.\, \mathrm{let}\, g \Leftarrow (\mathrm{fix}\, V) \,\mathrm{in}\, gx) \end{array}$$

Figure 7: Term reduction

$$\begin{array}{lcll} (S, E\{M\}) & \rightarrowtail & (S \circ E, M) & \\ (S \circ E, \mathrm{return}\, V) & \rightarrowtail & (S, E\{\mathrm{return}\, V\}) & \\ (S, M) & \rightarrowtail & (S, M') & \text{when } M \rightsquigarrow M' \end{array}$$

Figure 8: Evaluation steps

Lemma 1 (Safety $\star$)**.** *Typing of stacks and frame is consistent with the typing of terms and stack application.*

$$\left(\vdash S : \tau \multimap \tau'\right) \wedge \left(\vdash_C M : \tau\right) \implies \left(\vdash_C S\{M\} : \tau'\right)$$

Lemma 2 (Safety $\star$)**.** *If a pair (S, M) is well-typed then evaluation preserves this property and the type of $S\{M\}$ is preserved during the evaluation process*

Proof. Lemma 1 is proven using case analysis on S and Lemma 2 using case analysis on S and M. □

Example 2 (Reduction for combination of non-determinism and probabilities)**.** We can consider the following term in our language with signature $\Sigma = \{\mathrm{pr}, \mathrm{or}\}$:

$$M = (\lambda x : \mathrm{Nat}.\, \mathrm{pr}(x, \underline{1}))\underline{0}$$

This term corresponds to applying $\underline{0}$ to a function that tosses a coin and return either the input given to the function or $\underline{1}$. It can be shown that (Id, M) reduces to $(Id, \mathrm{pr}(\underline{0}, \underline{1}))$ as one could expect. However there is no more reduction possible afterwards: the evaluation is stuck, because there is no rule to evaluate the effect of a coin toss.

3.2 Computation Trees

We now have a definition of an operational semantics for the parametrized language, but it lacks one thing: any evaluation that encounters an effect is stuck, because there is no rule to deal with them. The technical tool used to keep the semantics uniform across the class of languages is the notion of computation tree [22] [13].

The idea is that any effect has a finite amount of arguments defined by its arity in Σ, because we do not know what the effect is going to do with the different arguments, we are going to compute all of them, and build a tree whose nodes are uninterpreted effects, and leaves are syntactical values of our calculus. One problem arises because of the fixed point operator: some trees can be infinite, and some computation are not terminating.

To deal with both issues, trees of type τ are going to have an ωCPPO structure, meaning that there is a preorder $\sqsubseteq$ on trees, that least-upper bounds of increasing sequences of trees can be taken, and that there is a bottom tree $\bot$ under any other tree. We are in fact building the free Σ-continuous algebra over the set of values of type τ [1].

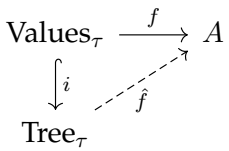
Definition 1 (Trees over a set)**.** A computation tree over a set X is the free continuous Σ-algebra over X. A continuous Σ-algebra is an ωCPPO equipped with continuous functions of appropriate arity for each operation symbol in Σ; a morphism of such algebras is a strict continuous function preserving the operations; the free continuous Σ-algebra functor is the left adjoint to the forgetful functor from the category of continuous Σ-algebras to that of sets.

The order on Tree_X is the following one: a tree t is an approximation of a tree t' if and only if t' can be obtained by replacing some of the leaves labelled by $\bot$ in t by arbitrary trees. The picture to have in mind is that $t \sqsubseteq t'$ when t is a *prefix* of t' with $\bot$ leaves where the tree t' should continue.

Definition 2 (Computation Tree)**.** Given τ a type, a computation tree of type τ is an element of Tree_τ which is a shorthand for the trees over the set of values of type τ. Because Tree_τ is the free continuous Σ-algebra over values of type τ we have the following universal property.

For every function $f : \mathrm{Values}_\tau \to A$ where A is a continuous Σ-algebra, there exists a unique morphism $\hat{f} : \mathrm{Tree}_\tau \to A$ such that:

$$f = \hat{f} \circ i$$



We can already see the usefulness of this definition when looking at the construction of substitution on trees: given abstractly, in full generality and guaranteed to be continuous without any hassle.

Definition 3 (Substitution)**.** Let $t \in \mathrm{Tree}_X$ and $\sigma : X \to \mathrm{Tree}_Y$. Using the universal property, there exists a unique lift $\hat{\sigma} : \mathrm{Tree}_X \to \mathrm{Tree}_Y$, which is the substitution. We write $t\sigma$ as a shorthand for $\hat{\sigma}t$.

Using this new construction it is now possible to continue our definition of the operational semantics without interpreting effects by building an infinite tree labeled by the effects encountered during evaluation. Because of the injection of values of type τ into Tree_τ, and to avoid unnecessary

clutter in the equations, we are going to treat this injection as an inclusion and write V instead of $i(V)$.

Definition 4 (Computation tree)**.** Given a well-typed pair (S, M) and an integer n we can compute the tree $|S, M|_n$ by induction on n following the rules in Figure 9. The sequence $|S, M|_n$ is ascending in n given a fixed pair (S, M) and we write $|S, M|$ for its least upper bound.

$$\begin{array}{lll} |S, M|_{n+1} & = |S', M'|_n & (S, M) \rightarrowtail (S', M') \\ |Id, \mathrm{return}\, V|_{n+1} & = V & \\ |S, \sigma(M_1, \ldots, M_l)|_{n+1} & = \sigma(|S, M_1|_n, \ldots, |S, M_l|_n) & \\ |S, M|_0 & = \bot & \end{array}$$

Figure 9: Computation tree construction

Example 3 (Non termination)**.** The following term does not terminate, and no effects occurs during the evaluation of this term:

$$\Omega_\tau = \mathrm{let}\, g : \mathsf{Nat} \to \tau \Leftarrow \mathrm{fix}(\lambda f : \mathsf{Nat} \to \tau.\, \mathrm{return}\, f)\, \mathrm{in}\, g\underline{0}$$

It is therefore possible to conclude that the computation tree associated to this term is always the least element of Tree_τ, independently of the stack $\vdash S : \tau \to \tau'$:

$$\forall S : \sigma \multimap \tau, |S, \Omega_\sigma| = \bot$$

One key result about this tree construction is the relationship between substitution on trees and application of stacks. To compute a computation tree for the pair (S, M) it suffices to compute the computation tree for (Id, M) and then substitute leaves with the computation tree obtained by $(S, \mathrm{return}\, V)$ where V is the corresponding value of the leaf.

Lemma 3 (Stack commutation $\star$)**.** *Let $S : \tau \multimap \tau'$ and $M : \tau$, we always have:*

$$|S, M| = |Id, M|\sigma_S$$

Where $\sigma_S(V) = |S, \mathrm{return}\, V|$.

Proof. The proof is done by considering the finite trees $|S, M|_n$, using induction on n and case analysis on the constructions. Afterwards it suffices to consider the least upper bound of $|S, M|_n$ to conclude. □

The last thing to check with our construction is that approximation on trees is indeed capturing the construction of infinite trees obtained by fixed points in the language. This equivalence is precisely given by the Theorem 1 stating that the least upper bound of the semantics of the approximations is exactly the semantics of the fixed point. An equivalent result also exists in call-by-name [13], note that because of the evaluation strategy, the theorem is stated on trees over values of type Nat to ensure full evaluation of the fixed-point. Unrolling fixed points is a very common operation [22] and this approximation result is a basic one in papers defining logical relations [19] [21].

Theorem 1 (Unrolling $\star\star$)**.** *Let $\vdash S : (\sigma \to \tau) \multimap \mathsf{Nat}$ be a stack and $\vdash_V V : (\sigma \to \tau) \to \sigma \to \tau$ be a value term.*

$$\bigsqcup_{n \geq 0} |S, \mathrm{unroll}_n\, V| = |S, \mathrm{fix}\, V|$$

$$\begin{aligned}\mathrm{unroll}_0\, V &= \Omega_{\sigma\to\tau}\\ \mathrm{unroll}_{n+1}\, V &= V(\lambda x:\sigma.\,\mathrm{let}\, g:\sigma\to\tau \Leftarrow (\mathrm{unroll}_n\, V)\,\mathrm{in}\, gx)\end{aligned}$$

Figure 10: Unrolling fixed point

Proof. The proof is quite tedious. First of all we replace all trees by their finite approximations, and then prove by induction that for all finite approximation on one side, there is a finite approximation on the other side that is above it. □

There is now an operational semantics for the language, respecting effects and capturing approximations. Up to this point, everything has been done uniformly over all signatures Σ. Given a computation term M of type τ, one can compute $|Id, M|$ (abbreviated $|M|$) which is a tree labelled with effects and has *values* of type τ as leafs.

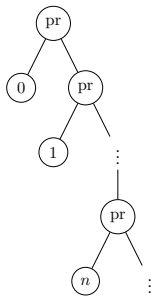
The next goal is to define contextual equivalence for a specific signature Σ. This is usually done by fixing a relation on computation terms of type Nat. However we can separate the semantics of the effects contained in this relation from the semantics of the ground language: it suffices to give a relation on *trees* of natural numbers to have one over computation terms of type Nat. Indeed, given a relation on $\mathrm{Tree}_{\mathrm{Nat}}$ it suffices to use the tree computation $|-|$ to transform it into a relation on computation terms of type Nat. The information required is therefore a simple relation over $\mathrm{Tree}_{\mathrm{Nat}}$.

Example 4 (Infinite probabilistic tree)**.** Let us consider the following program:

$$M = \mathrm{fix}\,(\lambda f:\mathrm{Nat}\to\mathrm{Nat}.\lambda x:\mathrm{Nat}.\,\mathrm{pr}(\mathrm{return}\, x, f(Sx)))\,\underline{0}$$

This program corresponds to a recursive call to a function that uses probabilistic choice to either return its argument n, or call itself with the value $n+1$. Even if the formal semantics of the `pr` effect is not yet defined, the intended meaning of such a program is to output 0 with probability one-half, 1 with probability one-fourth, etc. A hypothetical equivalent OCaml code and the associated computation tree $|M|$ can can be seen below.

```
let rec f x =
    match randomCoinToss () with
      | Heads -> return x
      | Tails -> return (f (x+1))
in
f 0
```



4 Contextual Preorder

Instead of simply studying the contextual *equivalence* relation, one can consider a contextual *preorder*: instead of stating when two programs can be interchanged, it helps understanding if a program can be "approximated" by another one. This can be useful when dealing with non-determinism or probabilistic choice, where one could replace a program with a non-equivalent one that has a strictly lower probability to fail. Contextual equivalence can then be obtained by considering the intersection of the contextual preorder with its opposite relation.

Because we are going to study the contextual *preorder*, the ground relation on trees of type Nat is going to be a *preorder*: a reflexive and transitive relation. By building computation trees and using

this basic preorder written $\sqsubseteq_b$, we are going to define an abstract contextual preorder as the largest relation satisfying some axioms [13] [8]. The definition is borrowed from the work of Andrew Pitts [21], but can be found in several other papers. The definition of *compatibility rules* in Figure 11 is simply obtained by looking at all the language constructions, and the corresponding type inference rule. The notion of *adequacy* on the other hand is uniquely determined by the preorder $\sqsubseteq_b$.

Definition 5 (Formalisation of relations respecting type)**.** Let $(\mathcal{E}_V, \mathcal{E}_C)$ be a pair of relations, the first one on values and the second one on computations. If $\bullet$ is V or C then the relation $\mathcal{E}_\bullet$ is a set of tuples of the form $(\Gamma_\bullet, M, M', \tau)$ and for every such tuple inside $\mathcal{E}$ we have $\Gamma \vdash_\bullet M : \tau$ and $\Gamma \vdash_\bullet M' : \tau$.

(i) We say that $\mathcal{E}$ is *compatible* when $\mathcal{E}$ is closed under rules in Figure 11[3]

(ii) We say that $\mathcal{E}$ is $\sqsubseteq_b$-adequate when for every pair of closed computation terms M and M' of ground type Nat we have $M\,\mathcal{E}_C\,M'$ implies $|M| \sqsubseteq_b |M'|$

We write $\Gamma \vdash M\,\mathcal{E}\,M' : \tau$ instead of $(\Gamma, M, M', \tau) \in \mathcal{E}$ to simplify reading.

$$\frac{}{\Gamma, x : \tau \vdash x\,\mathcal{E}_V\,x : \tau} \qquad \frac{\Gamma \vdash V\,\mathcal{E}_V\,V' : \tau}{\Gamma \vdash \mathrm{return}\,V\,\mathcal{E}_C\,\mathrm{return}\,V' : \tau}$$

$$\frac{\Gamma, x : \tau \vdash M\,\mathcal{E}_C\,M' : \tau'}{\Gamma \vdash (\lambda x : \tau.M)\,\mathcal{E}_V\,(\lambda x : \tau.M') : \tau \to \tau'} \qquad \frac{}{\Gamma \vdash \mathrm{Z}\,\mathcal{E}_V\,\mathrm{Z} : \mathrm{Nat}} \qquad \frac{\Gamma \vdash V\,\mathcal{E}_V\,V' : \mathrm{Nat}}{\Gamma \vdash \mathrm{S}\,V\,\mathcal{E}_V\,\mathrm{S}\,V' : \mathrm{Nat}}$$

$$\frac{\Gamma \vdash V\,\mathcal{E}_V\,V' : (\tau \to \tau') \to \tau \to \tau'}{\Gamma \vdash \mathrm{fix}\,V\,\mathcal{E}_C\,\mathrm{fix}\,V' : \tau \to \tau'} \qquad \frac{\Gamma \vdash V\,\mathcal{E}_V\,V' : \tau \to \tau' \qquad \Gamma \vdash W\,\mathcal{E}_V\,W' : \tau}{\Gamma \vdash VW\,\mathcal{E}_C\,V'W' : \tau'}$$

$$\frac{\Gamma \vdash V\,\mathcal{E}_V\,V' : \mathrm{Nat} \qquad \Gamma \vdash M_1\,\mathcal{E}_C\,M_1' : \tau \qquad \Gamma, x : \mathrm{Nat} \vdash M_2\,\mathcal{E}_C\,M_2' : \tau}{\Gamma \vdash (\mathrm{case}\,V\,\mathrm{of}\;\mathrm{Z} \Rightarrow M_1; \mathrm{S}(x) \Rightarrow M_2)\,\mathcal{E}_C\,(\mathrm{case}\,V'\,\mathrm{of}\;\mathrm{Z} \Rightarrow M_1'; \mathrm{S}(x) \Rightarrow M_2') : \tau}$$

$$\frac{\Gamma \vdash M\,\mathcal{E}_C\,M' : \tau \qquad \Gamma, x : \tau \vdash N\,\mathcal{E}_C\,N' : \tau'}{\Gamma \vdash (\mathrm{let}\,x : \tau \Leftarrow M\,\mathrm{in}\,N)\,\mathcal{E}_C(\mathrm{let}\,x : \tau \Leftarrow M'\,\mathrm{in}\,N') : \tau'}$$

$$\frac{(\sigma, n) \in \Sigma \qquad \forall 1 \leq i \leq n, \quad \Gamma \vdash M_i\,\mathcal{E}_C\,M_i' : \tau}{\Gamma \vdash \sigma(M_1, \ldots, M_n)\,\mathcal{E}_C\,\sigma(M_1', \ldots, M_n') : \tau}$$

Figure 11: Rules of compatibility with the language constructions

Definition 6 (Contextual Preorder)**.** There exists a largest compatible and $\sqsubseteq_b$-adequate relation called $\sqsubseteq_{ctx}$

In order to derive some of the theorems, we need to have more information about the $\sqsubseteq_b$ preorder. One of the first properties is that it should behave nicely with approximation of trees (admissibility) and that it should behave nicely with composition of trees (compositionality).

Definition 7 (Admissibility)**.** A relation R on $\mathrm{Tree}_{\mathrm{Nat}}$ is admissible for $\sqsubseteq$ when for every ascending chain $(t_i)_{i\geq 0}$ and $(t_i')_{i\geq 0}$ such that $t_i\,R\,t_i'$:

$$\left(\bigsqcup_{i\geq 0} t_i\right)\;R\;\left(\bigsqcup_{i\geq 0} t_i'\right)$$

[3] Any such relation is a congruence, and therefore stable under any context

Definition 8 (Compositionality)**.** A relation R on Tree_{Nat} is compositional when $t\,R\,t'$ and $\forall n, t_n\,R\,t'_n$ implies that:

$$t[\bar{n} := t_n]\,R\,t'[\bar{n} := t'_n]$$

Where $t[\bar{n} := t_n]$ is defined as the lifted $h_\sigma(t)$ where $\sigma(n) = t_n$.

The compositionality and admissibility requirements are natural ones, and they automatically have good properties on natural numbers (lemma 4), are nicely related to $\sqsubseteq$ (lemma 5) and can be constructed as the smallest preorder satisfying some inequational theory (lemma 6).

Lemma 4 (Behaviour on natural numbers $\star$)**.** *If the preorder $\sqsubseteq_b$ is compositional, then on natural numbers we have either that two distinct natural numbers are not comparable, or that every pair of tree is equated by $\sqsubseteq_b$.*

Proof. Assume that there exists two distinct numbers $\underline{m}$ and $\underline{n}$ such that $\underline{n} \sqsubseteq_b \underline{m}$. Let t and t' be two arbitrary trees and define:

$$\sigma(k) = \begin{cases} t & \text{if } k = n \\ t' & \text{if } k = m \\ \bot & \text{otherwise} \end{cases}$$

We have $\sigma \sqsubseteq_b \sigma$ because $\sqsubseteq_b$ is reflexive, and therefore using compositionality of $\sqsubseteq_b$ we have:

$$\underline{n}\sigma \sqsubseteq_b \underline{m}\sigma$$

Hence for every trees t and t' we have $t \sqsubseteq_b t'$. □

Lemma 5 (The preorder is coarser than $\sqsubseteq$ $\star\star$)**.** *Assume that $\sqsubseteq_b$ is admissible and compositional, then $(\sqsubseteq) \subseteq (\sqsubseteq_b)$ if and only if $\bot$ is a least element for $\sqsubseteq_b$.*

Example 5 (Simple counter example)**.** There exists an admissible and compositional preorder that does not extend $\sqsubseteq$.

Proof. Define $t \sqsubseteq_b t' \iff \bot \in t \implies \bot \in t'$ where the $\bot \in t$ means that there exists a leaf of t which is $\bot$ *or* that there exists an infinite branch in t. Compositionality and admissibility are simple, and we note that $\bot \not\sqsubseteq_b \underline{n}$, which proves the claim. □

Because the semantics of effects in Σ is given through the preorder $\sqsubseteq_b$, it can be useful to be able to automatically build such preorders. In fact it is possible to build free preorders respecting $\sqsubseteq$ given an inequational theory $\mathcal{T}$ as shown in the following lemma 6. This construction is not a trivial one: in the case of angelic non-determinism, the free preorder constructed is in fact the one that is expected (see Lemma 6).

Definition 9 (Horn Clause Inequational Theory)**.** A theory $\mathcal{T}$ is a horn-clause inequational theory over Tree_{Nat} if and only if it consists in a list of formulas obtained with the following grammar:

$$\begin{aligned} t &:= x \mid \sigma(\overbrace{t, \ldots, t}^{n}) \quad (\sigma, n) \in \Sigma \\ \phi &:= t \leq t \\ \psi &:= \phi \vee \cdots \vee \phi \vee \neg\phi \end{aligned}$$

Example 6 (Angelic non-determinism)**.** It can be shown that the inequational theory characterising the powerdomain for angelic non-determinism is the following one [1]:

$$x \leq \mathrm{or}(x,y) \qquad \mathrm{or}(x,x) \leq x \qquad \mathrm{or}(x,\mathrm{or}(y,z)) \leq \mathrm{or}(\mathrm{or}(x,y),z)$$
$$\mathrm{or}(x,y) \leq \mathrm{or}(y,x) \qquad \mathrm{or}(\mathrm{or}(x,y),z) \leq \mathrm{or}(x,\mathrm{or}(y,z))$$

Moreover the free preorder constructed using this theory corresponds to the following one: a tree t is under a tree t' if and only if all numbers that are obtained by leaves of t are also obtained in leaves of t'.

Lemma 6 (Free preorder construction $\star\star$)**.** *Given an inequational theory with horn-clauses $\mathcal{T}$ there exists a smallest admissible and compositional preorder $\sqsubseteq_{\mathcal{T}}$ on* $\mathrm{Tree}_{\mathrm{Nat}}$ *satisfying the inequational theory such that for any tree t, $\bot \sqsubseteq_{\mathcal{T}} t$.*

Proof. It is clear that satisfying some inequational theory is stable by arbitrary intersection. Because admissibility, compositionality and being a preorder are also stable properties by such intersection, one can take the intersection of all such preorders. □

We now have defined a notion of *contextual preorder* uniformly across the class of languages considered. The only specific point is the definition of the preorder $\sqsubseteq_b$ which is a preorder on trees with leaves of type Nat and nodes labelled by Σ. Therefore the required informations to use the results is going to be the pair $(\Sigma, \sqsubseteq_b)$ where Σ is simply carrying the *syntactic constructions* and $\sqsubseteq_b$ is carrying the actual *semantics* for the effects. We then saw that it was possible to build abstractly the preorder $\sqsubseteq_b$ given a set of Horn-Clauses that the preorder should satisfy.

5 Logical Relation

This section consists in defining a *relational interpretation* of types by induction on their structure, and proving that this relational interpretation characterises the contextual preorder $\sqsubseteq_{ctx}$. The overall proof is parametrized in both Σ and $\sqsubseteq_b$, under the assumption that the preorder is admissible and compositional.

The use of relational interpretation goes back to Reynolds [24] and Wadler [28] where instead of interpreting terms as elements of a set, and types as sets, terms are left uninterpreted, and types are interpreted as *relations*. To benefit from theses results, one need to parametrize over a class of *well-behaved* relations [21] and it turns out that *biorthogonality* is a generic construction that allows to construct them [16].

In the following section, we will fix a preorder $\sqsubseteq_b$ and assume it admissible and compositional. This preorder can be used to define an antitone Galois Connection written $\top$ between relations on stacks and relations on closed computation terms, meaning that if r is a relation on closed computation terms of type τ and s is a relation on stacks of type $\tau \multimap \mathrm{Nat}$:

$$r^{\top} \subseteq s \iff s^{\top} \subseteq r$$

Definition 10 (The $\top$ operation)**.** Let s be a relation on stacks of type $\tau \to \mathrm{Nat}$ and r be a relation on closed computation terms of type τ:

$$(S,S') \in r^{\top} \iff \forall (M,M') \in r, |S,M| \sqsubseteq_b |S',M'|$$
$$(M,M') \in s^{\top} \iff \forall (S,S') \in s, |S,M| \sqsubseteq_b |S',M'|$$

The main interpretation is that it is possible to relate stacks when we apply them to closed computation terms we can relate to each other and we can relate closed computation terms when we apply them to stacks we can relate to each other. The *biorthogonal* of a relation r on closed computation terms of type τ is then simply $r^{\top\top}$. It can be intuited that a relation r satisfying $r = r^{\top\top}$ going to be a relation preserving observational equivalence in some way (see Lemma 7), which is exactly the kind of relation we are looking for. In fact, the function $r \mapsto r^{\top\top}$ is a closure operator, that transforms any relation into a *biorthogonal* one.

We are now going to prove a saturation lemma that is going to justify the use of biorthogonality, and show how most of the proofs using biorthogonality are done: to prove that (M, N) is in r, it suffices to prove that (M, N) is in $r^{\top\top}$, which is equivalent to proving that for all pairs of stacks $(S, S') \in r^{\top}$, $|S, M| \sqsubseteq_b |S', N|$. One can see that this is clearly where the preorder $\sqsubseteq_b$ is taken into account, but also where the small-step semantics and computation trees can play a role.

Lemma 7 (Saturation for $\top\top$-closed relations and $\sqsubseteq_{ctx} \star$)**.** *Let r be a $\top\top$-closed relation on computations, we always have:*

$$(\sqsubseteq_{ctx} r \sqsubseteq_{ctx}) \subseteq r$$

Proof. Assume M, M', M'', M''' are closed computation terms of type τ such that $M \sqsubseteq_{ctx} M'$, $(M', M'') \in r$ and $M'' \sqsubseteq_{ctx} M'''$. Let $(S, S') \in r^{\top}$ we can have the following reasoning:

$$\begin{array}{rl}
|S, M| = |S\{M\}| & \text{definition of } |-,-| \\
\sqsubseteq_b |S\{M'\}| & S \text{ is a context} \\
= |S, M'| & \text{definition of } |-,-| \\
\sqsubseteq_b |S', M''| & \text{definition of } r^{\top} \\
= |S'\{M''\}| & \text{definition of } |-,-| \\
\sqsubseteq_b |S'\{M'''\}| & S' \text{ is a context} \\
= |S', M'''| & \text{definition of } |-,-|
\end{array}$$

Therefore we proved that $(M, M''') \in (r^{\top})^{\top}$, but the assumption was that r was $\top\top$-closed, and therefore $(M, M''') \in r$. □

5.1 Definition of the relation

We combine ideas from [13] for the treatment of effects and [20] for the adaptation to the call-by-value setting. The evaluation strategy can also be seen in [7] with an equivalent definition and interesting description of the actual way things are going to compute.

As usual, an operation on relations is defined for each type constructor. Because there is only one type constructor it suffices to define what is the *arrow* relation between two relations.

Definition 11 (Arrow relation)**.** Let r_1 be a relation on values and r_2 be a relation on computations of type respectively τ_1 and τ_2, we define $r_1 \to r_2$ a relation on values of type $\tau_1 \to \tau_2$ as:

$$r_1 \to r_2 = \left\{(V, V') \mid \forall (W, W') \in r_1, (VW, V'W') \in r_2\right\}$$

Note that values of type $\tau_1 \to \tau_2$ are all of the form $\lambda x : \tau_1.M$ where $x : \tau_1 \vdash_C M : \tau_2$.

The arrow relation is simply stating that two values of type $\tau_1 \to \tau_2$ are related when for all related arguments (values) they give related results (computations). It is now possible to write in a very simple way the logical relation on closed terms, which is going to be automatically $\top\top$-closed on computation terms at any type.

Definition 12 (Logical relation on closed terms)**.** The logical relation on closed terms is defined in Figure 12. For every type τ it defines a relation $\|\tau\|_V$ on values of type τ and $\|\tau\|_C$ on computations of type τ.

$$\begin{aligned} \|\text{Nat}\|_V &= \sqsubseteq_b \\ \|\tau \to \tau'\|_V &= \|\tau\|_V \to \|\tau'\|_C \\ \|\tau\|_C &= \left\{(\text{return}\, V, \text{return}\, V') \mid (V, V') \in \|\tau\|_V\right\}^{\top\top} \end{aligned}$$

Figure 12: Logical relation

If one of the key results in a logical relation argument is the reflexivity of the relation, it is very common to have a similar reflexivity result on stacks [21]. We are going to use several times the fact that $(Id, Id) \in \|\text{Nat}\|_C^{\top}$ which is a consequence of the stack reflexivity at ground type Nat.

Lemma 8 (Stack reflexivity at ground type Nat $\star\star$)**.** *For any stack* $\vdash S : \text{Nat} \multimap \text{Nat}$ *the pair* (S, S) *is inside* $\|\text{Nat}\|_C^{\top}$

Proof. Let S be a stack such that $\vdash S : \text{Nat} \multimap \text{Nat}$. Let V and V' be two values of type Nat such that $V \sqsubseteq_b V'$. Using the stack commutation lemma 3, $|S, \text{return}\, V| = |\,\text{return}\, V|\sigma_S$. But we know that $|\,\text{return}\, V| = V$, and now using compositionality and reflexivity of $\sqsubseteq_b$ it can be shown that $|S, \text{return}\, V| \sqsubseteq_b |S, \text{return}\, V'|$. This proves that $(S, S) \in \|\text{Nat}\|_C^{\top}$ because:

$$\|\text{Nat}\|_C^{\top} = \left\{(\text{return}\, V, \text{return}\, V') \mid (V, V') \in \|\text{Nat}\|_V\right\}^{\top}$$

□

Now in order to compare the logical relation to the contextual preorder, it is needed to be able to relate *open* terms (terms with free variables). The usual open extension of a relation is used, while being careful to only substitute *value* terms for the free variables.

Definition 13 (Generalisation to open terms)**.** If M and M' are two open terms with variables typed by Γ then $\Gamma \vdash M \,\|\tau\|_C\, M'$ if and only if for any pair of substitutions $\vec{V}$ and $\vec{V'}$ for the free variables such that $\vdash V_x \|\tau_x\|_V V'_x$ for any $x : \tau_x \in \Gamma$ we have:

$$\vdash M[x := V_x] \,\|\tau\|_C M'[x := V'_x]$$

The definition can be adapted to value terms in the obvious way.

One direct consequence of this defininition as an open extension is that two related computation terms with one free variable can be used to extend a stack.

Lemma 9 (Stack extension $\star\star$)**.** *Let* $(S, S') \in \|\tau\|_C^{\top}$ *be two related stacks, and* $x : \sigma \vdash M \|\tau\|_C M'$*. One can construct new stacks* $S \circ (\text{let}\, x : \sigma \Leftarrow \square \,\text{in}\, M)$ *and* $S' \circ (\text{let}\, x : \sigma \Leftarrow \square \,\text{in}\, M')$ *that have type* $\sigma \multimap \text{Nat}$*. Theses two new stacks are related for* $\|\sigma\|_C^{\top}$*.*

Proof. Let $(V, V') \in \|\sigma\|_V$, we can use the definition of the computation tree to reduce the stack:

$$|S \circ (\text{let}\, x : \sigma \Leftarrow \square \,\text{in}\, M, \text{return}\, V| = |S, M[x := V]|$$

This equation is also valid for the second computation tree built using S', M' and V'.

But we know that $\vdash M[x := V] \|\tau\|_C M'[x := V']$ by definition of the open extension of the relation. This proves that $|S, M[x := V]| \sqsubseteq_b |S', M'[x := V']|$ and therefore that the two stacks are indeed related for $\|\sigma\|_C^{\top}$. □

5.2 Inclusion in contextual preorder

The first step is to prove a soundness result, mainly that if two terms are logically related then they are contextually related. This soundness is proven in two steps: first prove adequacy of the logical relation, then prove compatibility with the constructions of the language.

Lemma 10 (Adequacy $\star$)**.** *The logical relation is adequate.*

Proof. Let M and M' be two computation terms of type Nat such that $\emptyset \vdash M \| \text{Nat} \|_C M'$. We know that for every pair $(S, S') \in \| \text{Nat} \|_C^\top$ we have:

$$|S, M| \sqsubseteq_b |S', M'|$$

Therefore it suffices to show that $(Id, Id) \in \| \text{Nat} \|_C^\top$ to conclude, but this is a direct consequence of lemma 8. □

We are now going to prove compatibility of the logical relation with the different constructions of the language. To simplify the proofs, compatibility is proven in an empty typing context Γ, this is allowed because the logical relation is defined on open terms using closing substitutions.

The compatibility rules are numerous and proofs of half of them are one-liners (variables, return, function-application, S, Z...). In order to save space, only the proof for the computation binding is done, but it is they all follow a similar pattern. There is however the exception of the fixed-point rule, requiring the use of the *admissibility* property of $\sqsubseteq_b$ and the *unrolling* theorem on fixed-points as one could expect.

Lemma 11 (Compatibility for computation binding $\star\star$)**.** *If Γ is a typing context, $\Gamma \vdash M \| \tau \|_C M'$ and $\Gamma, x : \tau \vdash N \| \tau' \|_C M'$ then:*

$$\Gamma \vdash (\mathrm{let}\, x : \tau \Leftarrow M \,\mathrm{in}\, N) \| \tau' \|_C (\mathrm{let}\, x : \tau \Leftarrow M' \,\mathrm{in}\, N')$$

Proof. Let $M = \mathrm{let}\, x : \sigma \Leftarrow M_1 \,\mathrm{in}\, M_2$, $M' = \mathrm{let}\, x : \sigma \Leftarrow M_1' \,\mathrm{in}\, M_2'$ and assume that $\emptyset \vdash M_1 \| \sigma \|_C M_1'$ and $x : \sigma \vdash M_2 \| \tau \|_C M_2'$.

Let $(S, S') \in \| \tau \|_C^\top$, we know that

$$\begin{cases} |S, M| & = |S \circ \mathrm{let}\, x : \sigma \Leftarrow \Box \,\mathrm{in}\, M_2, M_1| \\ |S', M'| & = |S' \circ \mathrm{let}\, x : \sigma \Leftarrow \Box \,\mathrm{in}\, M_2', M_1'| \end{cases}$$

Therefore to prove the inequality for $\sqsubseteq_b$ it suffices to show that

$$(S \circ \mathrm{let}\, x : \sigma \Leftarrow \Box \,\mathrm{in}\, M_2, S' \circ \mathrm{let}\, x : \sigma \Leftarrow \Box \,\mathrm{in}\, M_2') \in \| \sigma \|_C^\top$$

This is the exact conclusion of Lemma 9 and therefore we can conclude.

□

We therefore have the compatibility with respect to the language constructions, allowing us to state the inclusion of the logical relation inside the contextual preorder.

Theorem 2 (Inclusion of the preorders $\star$)**.** *The logical relation is included in the contextual preorder.*

Proof. The logical relation is adequate and compatible, and therefore is included in the largest relation that is adequate and compatible. □

5.3 Equality with contextual preorder

Because the language is call-by-value, the logical relation satisfies a very strong property that can be found in [20] in a slightly different form: the relation can be recovered from its restriction to values. Separating values from computation makes this result easier to prove and to grasp as seen in lemma 12. This result is necessary to prove completeness of the logical relation with respect to contextual equivalence, and can easily be extended when adding new types to the language like sum types or product types.

The proof of equality with the contextual preorder is done as follows:
(a) Prove that the logical relation is the largest adequate, compatible and *substitutive* relation
(b) Prove that the contextual preorder is substitutive

Lemma 12 (Value relation $\star\star$)**.** *For any type τ we have*

$$(\mathrm{return}\, V, \mathrm{return}\, V') \in \|\tau\|_C \iff (V, V') \in \|\tau\|_V$$

Proof. By doing a analysis on τ. □

Lemma 13 (Largest adequate compatible and substitutive relation $\star\star\star$)**.** *The logical relation is the largest adequate, compatible and substitutive relation. Where being substitutive is being compatible with the rules in Figure 13.*

$$\frac{\Gamma, x : \tau \vdash W \,\mathcal{E}_V\, W' : \tau' \qquad \Gamma \vdash V \,\mathcal{E}_V\, V' : \tau}{\Gamma \vdash W[x := V] \,\mathcal{E}_C\, W'[x := V'] : \tau'}$$

Figure 13: Substitutivity

Lemma 14 (Contextual preorder is substitutive $\star\star$)**.**

Proof. Because contextual preorder is transitive, it suffices to show that we have the following equality when M is a computation term of type τ and V a value term of type σ to have the substitutive property on computation terms:

$$\Gamma \vdash M[x := V] \equiv_{ctx} (\lambda x : \sigma.M)V : \tau$$

But we have already seen that this equivalence is true for the logical relation, and we know that the logical relation is included in the contextual preorder, allowing us to conclude. □

Theorem 3 (Contextual preorder equals the logical relation $\star$)**.**

Proof. We already have one inclusion with the Theorem 2 and the second one is given because contextual preorder is an adequate, compatible and substitutive relation, therefore included into the largest one (the logical relation). □

Using the abstract equality between the contextual preorder and the logical relation we defined, we can prove generic theorems about contextual equivalence independently of the properties of effects.

The first one is that a relation between effects is true at ground type if and only if they are true at all type. This justifies the fact that we only ask for a preorder on $\mathrm{Tree}_{\mathrm{Nat}}$.

Theorem 4 (Inequalities between effects are seen at ground type $\star\star$)**.** *Let t and t' be two finite trees in* Tree_{Nat}*, they can be identified as computation terms of type τ when substituting the leaves with computation terms of a type τ.*

For any pair of such substitutions (σ, σ') such that the images satisfies $\vdash \sigma(n)\|\tau\|_C\sigma'(n)$, the terms $t\sigma$ and $t'\sigma'$ are related for $\|\tau\|_C$ if and only if *it is the case when the property is restricted to substitutions of computation terms of type* Nat*.*

This theorem is stating that any *generic* inequalities on effects is already encoded in $\sqsubseteq_b$, and that any inequality in $\sqsubseteq_b$ is in fact a generic one at any type. For instance, if $\mathrm{pr}(a, b)$ is always approximating $\mathrm{or}(a, b)$ for $\sqsubseteq_b$ on trees of type Nat, then it will also be the case when looking at functions, or arbitrary complicated computation terms with complex type.

Another example of generic theorem is the reduction of contextual preorder on *open* terms to the contextual preorder on *closed* terms.

Theorem 5 (Contextual preorder can be reduced to closed terms $\star$)**.** *Two open computation terms M and M' are contextually related if and only if for any closing substitution of contextually related values, the two closed terms obtained are related. This is stating that $\sqsubseteq_{ctx}$ is the open extension of $\sqsubseteq_{ctx}$ restricted to closed terms.*

Several other global results can be declined, but it is not the goal to enumerate them here. Moreover, it is obvious that without specifying how $\sqsubseteq_b$ is behaving, proving specific results is not going to be possible.

We can now ask ourselves to what extend we could change the hypothesis on $\sqsubseteq_b$. The only result that was discovered is that removing compositionality allows to "break" the behaviour of effects. The following result is *not* using the meta-theorems that are themselves using compositionality and admissibility of $\sqsubseteq_b$.

Lemma 15 (Admissibility and good behaviour on Tree_{Nat} implies compositionality $\star\star\star$)**.** *Namely, we are going to show that if $\sqsubseteq_b$ is admissible then*

$$\left[(M, M') \in \|\text{Nat}\|_C \iff |M| \sqsubseteq_b |M'|\right] \iff \sqsubseteq_b \text{ is compositional}$$

This means that if the preorder is not compositional, the behaviour of computation terms of type Nat is not fully captured by $\sqsubseteq_b$.

6 Domain theoretic preorders

The goal of this section is to link the domain theoretic semantics with our setting. The first remark is that if $[\![\cdot]\!]$ is a semantic map from Tree_{Nat} to an ωCPPO $(D, \leq)$, one can define:

$$t \sqsubseteq_b t' \iff [\![t]\!] \leq [\![t']\!]$$

This preorder is not necessarily admissible or compositional, therefore we can ask ourselves what conditions on the semantic map $[\![\cdot]\!]$ are sufficient to obtain the desired properties.

One natural requirement for a semantic map is to be scott-continuous, and it turns out that it automatically gives an admissible preorder.

Lemma 16 (Admissibility $\star$)**.** *If the function is scott-continuous, then the relation defined is admissible.*

Proof. Let $(t_i)_i$ and $(t'_i)_i$ be two ascending chains for $\sqsubseteq$ with least upper bounds t and t' such that $[\![t_i]\!] \leq [\![t'_i]\!]$ for every i.

$$\begin{aligned}
\left[\!\!\left[\bigsqcup_i t_i\right]\!\!\right] &= \bigsqcup_i [\![t_i]\!] && \text{scott-continuity} \\
&\leq \bigsqcup_i [\![t'_i]\!] && \text{hypothesis} \\
&= \left[\!\!\left[\bigsqcup_i t'_i\right]\!\!\right] && \text{scott-continuity}
\end{aligned}$$

□

A natural assumption for compositionality is asking for the interpretation to be a homomorphism and for D to be a Σ-continuous algebra. Indeed, we are trying to interpret effects and it seems obvious that it is a good way to do so: in Plotkin and Power's work algebraic effects are the one that are preserved under some homomorphisms.

Lemma 17 (First half of compositionality $\star\star$)**.** *Assume D is a Σ-continuous algebra and $[\![\cdot]\!]$ is a homomorphism, then for every tree t and pair of substitutions (σ_1, σ_2) such that $\sigma_1(n) \sqsubseteq_b \sigma_2(n)$ for every n, we have:*

$$t\sigma_1 \sqsubseteq_b t\sigma_2$$

But it is not enough to guarantee compositionality of the preorder, as it can be seen in the example 7.

Example 7 (Simple counterexample)**.** Let $\Sigma = \{(+, 2), (\times, 2)\}$ be a signature. Let D be $\mathbb{N} \cup \{+\infty\}$ with the usual ordering and the usual Σ-algebra structure. It is a continuous Σ-algebra and the homomorphism that arises from identity on natural numbers is just the regular interpretation of tree of operations.

But if $t_1 = 1 + 2$ and $t_2 = 1 \times 3$, we can take σ defined as follows to break compositionality:

$$\sigma(k) = \begin{cases} 0 & \text{if } k = 1 \\ k & \text{otherwise} \end{cases}$$

We have $t_1 \sqsubseteq_b t_2$ but not $t_1\sigma \sqsubseteq_b t_2$ because $2 \not\leq 0$.

It is therefore necessary to ask for the interpretation to respect effects in a deeper way, related to the definition of *algebraic* effects.

Definition 14 (Factorisation of homomorphism)**.** The interpretation $[\![\cdot]\!]$ factors homomorphisms when for every function $\sigma : \text{Nat} \to D$ there exists a homomorphism $h_\sigma : D \to D$ such that $\sigma = h_\sigma \circ [\![\cdot]\!]$.

$$\text{Nat} \xrightarrow{[\![\cdot]\!]} D \overset{h_\sigma}{\dashrightarrow} D \qquad \text{Nat} \xrightarrow{\sigma} D$$

Lemma 18 (Factorisation of tree homomorphisms $\star\star$)**.** *Assume that $[\![\cdot]\!]$ factors homomorphisms. If $\tau : \text{Tree}_{\text{Nat}} \to D$ is a homomorphism then there exists h_τ a homomorphism from D to D such that $\tau = h_\tau \circ [\![\cdot]\!]$.*

Intuitively, it states that the domain D contains enough information about the tree to be able to use just the information in D when doing computation: this is why a substitution on trees can be converted into a computation inside D.

Example 8 (Probabilities)**.** Restricting ourselves to the case were values on trees can only be in $\{1,2\}\cup\{\bot\}$. If the domain D is the domain of sub-probability measures on $\{1,2\}$, then a tree is mapped into a vector of probabilities for each outcome and $\bot$ is ignored. The probability vector obtained after a substitution is exactly the product of the original probability vector with the probability matrix the substitution represents.

Let $t = \mathrm{pr}(1,2)$, $\sigma(1) = \mathrm{pr}(1,2)$ and $\sigma(2) = 1$. The semantics of t is the vector $(0.5, 0.5)$, and the semantics of $t\sigma$ is $(0.5 \times 0.5 + 0.5, 0.5 \times 0.5)$. The calculation is exactly equivalent to the following matrix product:

$$\begin{pmatrix} 0.5 & 1 \\ 0.5 & 0 \end{pmatrix} \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

Where the matrix has been obtained by putting in column the probability vectors of $\sigma(1)$ and $\sigma(2)$. The function h_σ is in our case the product with the transition matrix given by σ.

This example can be turned into a counter example. Keeping the same domain, but allowing trees to have leaves in $\{1,2,3\}\cup\{\bot\}$, we are forgetting information that is crucial when doing substitution. Indeed, because we don't remember what is the probability of 3, a substitution that turns 3 into 1 cannot be turned into a homomorphism from D to D.

Lemma 19 (Second half of compositionality $\star\star$)**.** *If $[\![\cdot]\!]$ is a homomorphism that factors functions from* Nat *to D then the preorder $\sqsubseteq_b$ is compositional.*

6.1 Examples

We now list examples where the previous results can be used to reuse work done on denotational semantics. The first example is when D is a free algebra. That can happen in cases such as Powerdomains [1], Powercones [27], or Powerkegelspitze [14].

Example 9 (Free algebras)**.** Assume that D is a free algebra over Nat and it holds that it is a Σ-continuous algebra. Then the preorder defined using the embedding of Nat into the free structure gives an admissible and compositional preorder.

The second possibility is to already have a denotational semantics for the language defined in a monadic way on the category of ωCPPOs.

Example 10 (Monadic interpretation)**.** If the language is interpreted in a ωCPPO with interpretations for effect that are *natural* with respect to the EM-morphisms then the preorder obtained is admissible and compositional.

The proof is requiring numerous definitions and is therefore omitted even if it is not a complex one. This result can be used to show that given an interpretation, the contextual preorder we can define is the same as the contextual preorder defined using first the tree calculation and then the monadic interpretation. This result comes from [22]. Indeed, in the setting of a monadic interpretation $[\![\cdot]\!]$, Plotkin and Power showed that $[\![|M|]\!] = [\![M]\!]$: the monadic semantics of the tree is the same as the monadic semantics of the original term. This shows that one can define the contextual preorder using the interpretation on terms or the interpretation on trees *indifferently*, showing that using computation trees does not restrict generality in this case.

Using free structures in the category of ωCPPO [1] it is possible to define a free preorder in an abstract way.

Lemma 20 (Free interpretational preorder $\star$)**.** *Given an inequational theory $\mathcal{T}$ there exists a preorder $\sqsubseteq_{[\![\mathcal{T}]\!]}$ arising from an interpretation into a Σ-continuous algebra satisfying the inequational theory $\mathcal{T}$, and containing any preorder constructed using the same pattern.*

This gives us yet another way to define a preorder $\sqsubseteq_b$ from a theory $\mathcal{T}$, and comparing on different examples the two is going to be the subject of the next section.

7 Free preorders

Given an inequational theory $\mathcal{T}$ one can always build some preorder $\sqsubseteq_b$ corresponding to it as in Lemma 20. This uses the previous part about denotational semantics and preorders built using them. However we saw that one could naturally build the smallest admissible, compositional preorder extending $\sqsubseteq$ without requiring such machinery by using Lemma 6. It is not known (yet) if the two constructions coincide, however one inclusion is clear: the free preorder obtained using a denotational interpretation always contains the other one, defined as the intersection of all preorders satisfying admissibilty, compositionality and pointedness properties. The difficulty is going to prove the equality between the free preorder (obtained via Lemma 6) and the other ones either obtained operationally or using the denotational free preorder.

Lemma 21 (Demonic preorder $\star\star$)**.** *Let $\mathcal{T}_D$ be the following theory:*

$$\begin{aligned} a \operatorname{or} a &= a \\ a \operatorname{or} b &= b \operatorname{or} a \\ a \operatorname{or} (b \operatorname{or} c) &= (a \operatorname{or} b) \operatorname{or} c \\ a \operatorname{or} b &\leq a \end{aligned}$$

The free preorder for the theory $\mathcal{T}_D$ coincides with the preorder $\sqsubseteq_D$ given by:

$$t \sqsubseteq_D t' \iff \begin{cases} \bot \in t' \implies \bot \in t \\ n \in t' \implies n \in t \\ \bot \in t \end{cases}$$

And $\sqsubseteq_D$ itself coincides with the denotational iterpretation in the Smyth powerdomain for Nat *(the free meet-semilattice over* Nat*).*

The proof of Lemma 21 is omitted because it consists in applying several rewriting steps on finite trees and because infinite trees are all equivalent to $\bot$ when considering demonic choice[4]. Moreover, explicit characterisations of the Smyth powerdomain can be found in the literature [1] and allow to link the operational view to the denotational one.

Lemma 22 (Probabilistic preorder $\star\star$)**.** *We use the notation $\oplus$ for the infix notation of* pr *to ease lecture. Let $\mathcal{T}_P$ be the following probabilistic theory developed by Heckmann and Reinhold [11]:*

$$\begin{aligned} a \oplus a &= a \\ a \oplus b &= b \oplus a \\ (a \oplus b) \oplus (c \oplus d) &= (a \oplus c) \oplus (b \oplus d) \\ a \oplus b \leq b &\implies a \leq b \end{aligned}$$

The free preorder for the theory $\mathcal{T}_P$ coincides with the preorder defined by:

$$t \sqsubseteq_P t' \iff \forall n \in \mathrm{Nat}, \nu(t) \leq \nu(t')$$

Where $\nu(t)$ corresponds to the probability distribution over Nat *encoded by the tree t.*

And $\sqsubseteq_P$ itself coincides with the denotational iterpretation in the probabilistic powerdomain for Nat *(the free full kegelspitze over* Nat *as defined by Keimel [14]).*

Proof. It is easy to check that the operational preorder satisfies the theory $\mathcal{T}_P$.

For the other inclusion, on finite trees, we can compare two trees t and t' by putting them in a common form (complete binary trees of the same depth), and if their distributions were ordered

[4]It is indeed always possible for the demon to make the program fail to terminate by always selecting the branch with infinite depth, existing because the tree is infinite

$\nu(t) \leq \nu(t')$, then each leaf in the smaller tree t can be put under the same leaf in the bigger one t', and a proof of the inequality can be built using compositionality and the fact that $\bot$ is under any other element.

Let t and t' be two possibly infinite trees of natural numbers such that the leaves are bounded by a constant C and assume that $\nu(t) \leq \nu(t')$. We can make a case distinction:

1. There exists a finite tree t'_i such that $\nu(t'_i) = \nu(t')$ and $t'_i \sqsubseteq t'$.
2. For any finite tree t'_i such that $t'_i \sqsubseteq t'$, there exists an n such that $\nu(t'_i)(n) < \nu(t')(n)$.

Now let's take two approximating chains of finite trees for t and t', and build the chain $t_i \oplus t'_i$.

We know that $\nu(t_i \oplus t'_i) \leq \nu(t')$ by some simple calculation. Now, assume we are in the first case, then we have a finite approximating tree t'_j with $j > i$ such that $\nu(t_i \oplus t'_i) \leq \nu(t'_j)$. In the second case, we know that there is an n such that the inequality is strict: but because ν is scott-continuous, there is a $j > i$ such that $\nu(t_i \oplus t'_i)(n) < \nu(t'_j)$. Because the support is finite, we know that we can take the maximum of such j's and have a finite tree t'_j such that $\nu(t_i \oplus t'_i)(n) \leq \nu(t'_j)$.

But all the trees in this last equation are finite, and therefore they are true for $\sqsubseteq_{\mathcal{T}_P}$.

$$\forall i \in \mathbb{N}, \exists j > i, t_i \oplus t'_i \sqsubseteq_{\mathcal{T}_P} t'_j$$

Using admissibility, we can now conclude:

$$t \oplus t' \sqsubseteq_{\mathcal{T}_P} t'$$

But then we can deduce that $t \sqsubseteq_{\mathcal{T}_P} t'$ using the last axiom of $\mathcal{T}$.

To extend this result to infinite support, it suffices to use the family of substitutions:

$$\sigma_k(i) = \begin{cases} \bot & \text{when } i > k \\ i & \text{otherwise} \end{cases}$$

If $\nu(t) \leq \nu(t')$ then we know that for all k, $\nu(t\sigma_k) \leq \nu(t'\sigma_k)$ and they have finite support, therefore $t\sigma_k \sqsubseteq_{\mathcal{T}_P} t'\sigma_k$. To conclude it suffices to see that $t = \sqcup_k t\sigma_k$ and use admissibility.

□

8 Preorder for combined non determinism and probabilities

In this section we are going to fix a specific signature Σ containing two binary operators `pr` and `or`. The two operators are used to model a language where both probabilistic choice and non-determinism coexist. Combining these specific effects has been the subject of numerous papers, and even when restricting ourselves to the denotational setting, the work of Regina Tix on powercones [27] continued afterwards by Plotkin and Keimel [14] on Kegelspitze shows the interest of such combination. A more functional version of theses domains can also be found in the work of Jean-Goubault Larrecq [10].

Given the recent developments of the denotational interpretation, it would be perfectly fine to use an interpretation to define our basic preorder $\sqsubseteq_b$. But as we are studying operational semantics, it is more consistent to use an operational definition of the said preorder.

Example 11 (Concurrent processes)**.** A concrete example where this combination arises is the study of concurrent processes with probabilistic choice: the scheduler for the different processes is a demon playing the worst move every time, and the different parts of the program can use a random number generator to make decisions.

A program M is an under approximation of a program M' if and only if any choice of the scheduler in M' leading to some distribution of probability on the values there exists a choice of the scheduler of M giving a distribution that is point wise lower.

The natural interpretation is that we are always considering the worst case scenario, and if something can go wrong (not producing a value because of non-termination) in M' then it is necessarily worse for the approximation M. This way, reasoning on M is *safe* because if the probability of failure is bounded for M, the bound is also valid for M'.

In the case of combined (demonic) non-determinism and probabilities we can define the preorder $\sqsubseteq_b$ on trees over natural numbers in a simple and effective way. We consider a tree as a Markov Decision Process and given an cost function from Nat $\to \overline{\mathbb{R}_+}$ we find a strategy for the `or` nodes that minimizes the average cost of the tree. A tree t is under a tree t' for this preorder when for any cost function, the minimal expected cost for t is under the minimal expected cost for t'.

In order to formalise this intuition, while not diving into the details of all the specifications one can define $\mathcal{S}$ to be the space representing the set of strategies. Given a strategy $s \in \mathcal{S}$ and a tree $t \in \text{Tree}_{\text{Nat}}$, one can build $t * s$ the application of the strategy to the tree, that builds a new *probability* tree, that is *without* `or` nodes. Given a probability tree t, and a cost function h from Nat to $\overline{\mathbb{R}_+}$, one can define the expected cost $\mathbb{E}(h(t))$. It is now possible to write the following definition for the operational preorder:

$$t \sqsubseteq_b t' \iff \forall h : \text{Nat} \to \overline{\mathbb{R}_+}, \inf_{s \in \mathcal{S}} \mathbb{E}(h(t * s)) \leq \inf_{s \in \mathcal{S}} \mathbb{E}(h(t' * s))$$

Admissibility is going to rely on the *Scott-continuity* of the following function:

$$t \mapsto \left(h \mapsto \inf_{s \in \mathcal{S}} \mathbb{E}(h(t * s)) \right)$$

Compositionality on the other hand is going to rely on an elementary decomposition result of the above function.

In order to save space and keep this report readable, definitions and lemmas are not written explicitly but listed and explained below, highlighting the shape of the construction more than the technicalities.

1. **Definition** of the strategy space $\mathcal{S}$ as the set of infinite trees labelled by `Left` and `Right`.
2. **Definition** of the application of a strategy to a tree in Tree_{Nat} as the tree obtained by "selecting" the left or right branch for all $\sqcap$-nodes, based on the label found in the corresponding place in the strategy tree.
3. **Definition** allowing to transform a tree with only $\oplus$-nodes into a random variable, allowing to consider a tree with probabilistic choice and natural number leaves as a random variable outputting natural numbers.
4. **Definition** of a function $F(t, s, h)$ that takes as input a tree t, a strategy s and a cost function h : Nat $\to \overline{\mathbb{R}_+}$ and outputs the expected cost of the random variable $h(t * s)$ where $t * s$ is considered as a random variable
5. **Definition** of the preorder $\sqsubseteq_b$ on trees as follows

$$t \sqsubseteq_b t' \iff \forall h : \text{Nat} \to \overline{\mathbb{R}_+}, \inf_{s \in \mathcal{S}} F(t, s, h) \leq \inf_{s \in \mathcal{S}} F(t, s, h)$$

After formalising the definition of the preorder $\sqsubseteq_b$, the first goal is to prove its admissibility, which is done by studying the function F. The proof is done using the following lemmas:

1. **Lemma** stating that the space $\mathcal{S}$ is *compact* by relating it to the Cantor Space
2. **Lemma** stating that the function $(t, s) \mapsto t * s$ from $\text{Tree}_{\text{Nat}} \times \mathcal{S}$ to Tree_{Nat} applying a strategy is *continuous*
3. **Lemma** stating that the function $(t, s) \mapsto F(t, s, h)$ is *continuous* for all fixed cost function h
4. **Lemma** stating that the function $t \mapsto \inf_{s \in \mathcal{S}} F(t, s, h)$ is *continuous* in t. This is the difficult result because it requires using the compactness of $\mathcal{S}$ and either a tedious proof, or a generic result about infimum in the case of domains and compact sets [25] (Theorem 7.31).
5. **Conclusion:** the preorder is admissible when considering the previous Lemma and by adapting the proof of Lemma 16

The second goal is to obtain compositionality result, this is done using the following steps:

1. **Lemma** stating a decomposition result about the function F. This result is difficult to prove, but is a very natural one. Given a function h, a tree t and a substitution σ, the following equality holds:
$$\inf_s F(t\sigma, s, h) = \inf_s F(t, s, h_\sigma)$$
Where
$$h_\sigma(n) = \inf_s F(\sigma(n), s, h)$$
2. **Conclusion:** the preorder $\sqsubseteq_b$ is compositional, and this follows directly from the previous Lemma, in conjunction with monotonicity properties of F.

Everything can be adapted to the angelic case by replacing $\inf$ with $\sup$ in the definition of the preorder. In fact, admissibility even becomes easier because suprema commute but the general proof can be almost copy-pasted.

8.1 Link with the denotation and the free preorder

One can consider the preorder $\sqsubseteq_{\mathcal{T}}$ freely generated (using lemma 6) by some horn-clause inequational theory $\mathcal{T}$. The inequational theory for demonic non-determinism is well known and we call it $\mathcal{D}$, a good choice of axiomatisation $\mathcal{P}$ for the probability can be found in the work of Heckmann and Reinhold [11]. This theory has the advantage of not explicitly referring to real numbers and is therefore perfectly suited to our setting.

$\mathcal{P}$
$$\begin{aligned} a \oplus a &= a \\ a \oplus b &= b \oplus a \\ (a \oplus b) \oplus (c \oplus d) &= (a \oplus c) \oplus (b \oplus d) \\ a \oplus b \leq b &\implies a \leq b \end{aligned}$$

$\mathcal{D}$
$$\begin{aligned} a \sqcap a &= a \\ a \sqcap b &= b \sqcap a \\ (a \sqcap b) \sqcap c &= a \sqcap (b \sqcap c) \\ a \sqcap b &\leq a \end{aligned}$$

Distributivity
$$(a \sqcap b) \oplus c = (a \oplus c) \sqcap (b \oplus c)$$

Figure 14: Inequational theory for mixed probability and demonic non determinism

Given the two theories, and following the laws from [14] we can build the combined theory of demonic non-determinism and probabilities by adding a distributivity axiom as seen in Figure 14.

It is already known that each part corresponds to the usual preorders for probability (resp. non determinism) using Lemma 22 (resp. Lemma 21), and we are going to show the following theorem.

Theorem 6 (Equality of preorders $\star\star\star\star$)**.** *The free preorder of the joint theories as described in Figure 14 is the one that was obtained operationally which is itself equal to the preorder obtained by the interpretation inside the free algebra for this theory in ωCPPO.*

The proof of this theorem is long and complex. It uses both domain theory to relate the operational preorder to the denotational one obtained using a Power Kegelspitze [14], but also when proving the equality between the freely generated preorder and the operationally defined one. The proof follows the same pattern as the one for probabilities in Lemma 22 but is very technical and quite long.

9 Conclusion and future work

The first part of this internship was about extending the results of Patricia Johann, Alex Simpson and Janis Voigtländer [13] to a call-by-value setting. By doing so, some properties of the basic preorder $\sqsubseteq_b$ were developed and a direct link to denotational semantics has been made. This is a first step in a better understanding of basic preorders and the generality of the method itself. Some generic theorems and sanity checks have been proven abstractly for the logical relation and the contextual preorder arising from $\sqsubseteq_b$, allowing to decline them with any effect signature Σ. The ability to automatically build free preorders for an equational theory $\mathcal{T}$ was studied, and compared to the operational and denotational method in the case of probability, non-determinism and the combination of both, showing how robust this general setting is.

It is clear that the study of basic preorders is not fully satisfactory, if only because the admissibility property does not seem to be a necessary one. For instance, countable non-determinism does *not* have this admissibility property, but is believed to still fit in our setting. On the other side of the requirements, compositionality could be better understood using sets of observations as done in [13] or looking at the continuity properties of the monadic multiplication on trees.

It would then be interesting to try and generalise the class of effects, be it by adding more effects known to be algebraic, or blatantly non algebraic effects such as exception handling that would require changing the language itself, but still be captured by the same general method.

This work can be extended to a richer type system in an obvious way, and even recursive types are not problematic thanks to step-indexing techniques [5] or syntactic projections [6].

It could be interesting to generalise the notion of $\top\top$-closure to metric relations and talk about the distance of terms, as many speakers in CALCO and MFPS have suggested.

Finally, the study of mixed non-determinism and probability is done very briefly in this paper, and there would be a lot more to talk about. For instance, how does the functional representation evolves when combining angelic, demonic and probability operators ?

References

[1] Abramsky, S., Jung, A.: Domain theory. In: Handbook of Logic in Computer Science. pp. 1–168. Clarendon Press (1994)

[2] Abramsky, S., McCusker, G.: Game semantics. In: Computational logic. pp. 1–55. Springer (1999)

[3] Amadio, R.M., Curien, P.L.: Domains and Lambda-Calculi (Cambridge Tracts in Theoretical Computer Science). Cambridge University Press, New York, NY, USA, 1 edn. (2008)

[4] Bauer, A., Pretnar, M.: Programming with algebraic effects and handlers. CoRR abs/1203.1539 (2012), http://arxiv.org/abs/1203.1539

[5] Benton, N., Hur, C.K.: Biorthogonality, step-indexing and compiler correctness. ACM Sigplan Notices 44(9), 97–108 (2009)

[6] Crary, K., Harper, R.: Syntactic logical relations for polymorphic and recursive types. Electronic notes in theoretical computer science 172, 259–299 (2007)

[7] Dagand, P.É., Scherer, G.: Normalization by realizability also evaluates. In: Vingt-sixièmes Journées Francophones des Langages Applicatifs (JFLA 2015) (2015)

[8] Dal Lago, U., Gavazzo, F., Blain Levy, P.: Effectful Applicative Bisimilarity: Monads, Relators, and Howe's Method (Long Version). ArXiv e-prints (Apr 2017)

[9] Felleisen, M., Hieb, R.: The revised report on the syntactic theories of sequential control and state. Theor. Comput. Sci. 103(2), 235–271 (Sep 1992), http://dx.doi.org/10.1016/0304-3975(92)90014-7

[10] Goubault-Larrecq, J.: Isomorphism theorems between models of mixed choice. Mathematical Structures in Computer Science (2016), http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/JGL-mscs16.pdf, to appear

[11] Heckmann, R.: Probabilistic domains. Trees in Algebra and Programming—CAAP'94 pp. 142–156 (1994)

[12] Honda, K., Yoshida, N., Berger, M.: An observationally complete program logic for imperative higher-order functions. In: Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on. pp. 270–279. IEEE (2005)

[13] Johann, P., Simpson, A., Voigtländer, J.: A generic operational metatheory for algebraic effects. In: 2010 25th Annual IEEE Symposium on Logic in Computer Science. pp. 209–218 (July 2010)

[14] Keimel, K., Plotkin, G.D.: Mixed powerdomains for probability and nondeterminism. Logical Methods in Computer Science 13(1) (2017), http://dx.doi.org/10.23638/LMCS-13(1:2)2017

[15] Koutavas, V., Levy, P.B., Sumii, E.: From applicative to environmental bisimulation. Electronic Notes in Theoretical Computer Science 276, 215–235 (2011)

[16] Mellies, P.A., Vouillon, J.: Recursive polymorphic types and parametricity in an operational framework. In: Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on. pp. 82–91. IEEE (2005)

[17] Mislove, M.: Models Supporting Nondeterminism and Probabilistic Choice, pp. 993–1000. Springer Berlin Heidelberg, Berlin, Heidelberg (2000), http://dx.doi.org/10.1007/3-540-45591-4_136

[18] Mislove, M., Ouaknine, J., Worrell, J.: Axioms for probability and nondeterminism. Electronic Notes in Theoretical Computer Science 96, 7–28 (2004)

[19] Pitts, A.M.: Operationally-based theories of program equivalence. Semantics and Logics of Computation 14, 241 (1997)

[20] Pitts, A.M.: Existential types: Logical relations and operational equivalence. In: International Colloquium on Automata, Languages, and Programming. pp. 309–326. Springer (1998)

[21] Pitts, A.M.: Parametric polymorphism and operational equivalence. Mathematical. Structures in Comp. Sci. 10(3), 321–359 (Jun 2000), http://dx.doi.org/10.1017/S0960129500003066

[22] Plotkin, G., Power, J.: Adequacy for algebraic effects. In: International Conference on Foundations of Software Science and Computation Structures. pp. 1–24. Springer (2001)

[23] Plotkin, G.D.: Lcf considered as a programming language. Theoretical computer science 5(3), 223–255 (1977)

[24] Reynolds, J.C.: Types, abstraction and parametric polymorphism. In: IFIP Congress. pp. 513–523 (1983), http://dblp.uni-trier.de/db/conf/ifip/ifip83.html#Reynolds83

[25] Schalk, A.: Algebras for generalized power constructions. Ph.D. thesis, TH Darmstadt (1993)

[26] Scott, D.S.: Domains for denotational semantics. In: International Colloquium on Automata, Languages, and Programming. pp. 577–610. Springer (1982)

[27] Tix, R., Keimel, K., Plotkin, G.: Semantic domains for combining probability and nondeterminism. Electronic Notes in Theoretical Computer Science 222, 3–99 (2009)

[28] Wadler, P.: Theorems for free! In: Proceedings of the fourth international conference on Functional programming languages and computer architecture. pp. 347–359. ACM (1989)