MFPS X111 (1997) ?? pages

A categorical account of two-level languages

Eugenio Moggi

moggi@disi.unige.it
DISI, Univ. di Genova
Genova, Italy

Abstract

We give an account of two-level languages in terms of indexed categories and universal properties well-known in the context of categorical logic. This account provides three important insights: establishes precise analogies between two-level languages and module languages, explains the two-level languages used in partial evaluation (see [7]) in terms of those used for code generation (see [16]), suggests extensions that should be valuable for type-specialization (see [9]) and shape-analysis (see [1]).

Introduction

Two-level languages are an important tool for analyzing programs, their distinctive feature is to make binding-time information explicit. Once this information is made explicit, it can be used in several ways: analysis, optimization, or code generation. For instance, in partial evaluation two-level languages are used to identify those parts of a program that can be reduced statically, and those that have to be evaluated dynamically.

In the two-level language of [16] the binding-time information is recorded by two *sorts*, r (for run-time) and c (for compile-time), and a set of formation rules allows to infer the binding-time b of types $\tau : b$ and expressions $e : \tau : b$. Another feature of the language is that type and term constructors are *duplicated*, we do not regard this as fundamental. Indeed the parameterized semantics of [16] leaves complete freedom to the interpretation of run-time types, and so the interpretation $\underline{\to}$ (unlike that of $\to$) may have nothing to do with function spaces. We take as fundamental that "all compile-time computations take place before any run-time computation" (page 43 of [16]). Formally this is captured by the impossibility for compile-time expressions to have run-time free variable (this is easy to prove but not stated in [16]), i.e. $\Gamma \vdash e : \tau : c$ implies $\Gamma(x) = (\_ : c)$ for any $x \in \mathrm{FV}(e)$.

In the context of partial evaluation a different kind of two-level languages are used. In this case the two levels are called d (for dynamic) and s (for static). There are strong similarities between level d and r, while level s is drastically different from c (e.g. a term $e : t : s$ may have free variables of a d-type). Researchers in the areas think that these are *necessary variations*

This is a preliminary version. The final version can be accessed at
URL: `http://www.elsevier.nl/locate/entcs/volume5.html`

over a theme, and at best one can provide a common descriptive framework to highlight similarities and differences among such variations (see [15]). While others don't seem fully aware of the differences (e.g. [2] says that he is using the two-level language of [16], but he doesn't).

It is well-known that simply typed λ-calculus enjoys both a logical and categorical status, as the language for proofs in NJ (natural deduction for intuitionistic logic) and as the internal language of CCC (cartesian closed categories). This triangular relation between typed calculi, logic and category theory is very useful for transferring new ideas, testing their robustness, and establishing fruitful analogies. It was only recently that [5,4] have proposed a very clean account of (both variations of) two-level languages in terms of modal and temporal logic.

In this paper we give a categorical account of two-level languages, and discuss the new insights it provides. We show that **the categorical account of phase distinction and (higher-order) module facilities by [13,14] applies almost unchanged to two-level languages**. Indeed the *independence* property of type expressions from program expressions (in languages like Pascal, core ML, ADA), is the same phenomenon pointed out above for the two-level language of [16,5]. Moreover, **the static/dynamic classification of [7,4] can be explained in terms of the compile-time/run-time classification**, contrary to the belief that these are *necessary variations*.

There are important dividends paid off by our analysis. For instance, the analogies with module languages suggest:

- run-time types depending on compile-time values; for partial evaluation this had already been advocated by Launchbury (see [11,12]);
- new run-time types obtained by existential or universal quantification over compile-time types;
- the translation of a source program into a pair of terms, the first *living* at compile-time and the second at run-time.

All these extensions seem necessary for type-specialization (see [9]) and shape-analysis (see [1]). The categorical semantics is naturally biased towards dependent types, but we want to relate it to simple (and well-known) two-level languages, so we stick to non-dependent types.

Section 1 is about two-level languages based on the compile-time/run-time classification, while Section 2 is about those based on the static/dynamic classification. Both sections follow a similar pattern: syntax of an archetypical two-level language described in the common framework proposed by [15], the proposed categorical semantics, a discussion of how this fits with denotational semantics proposed in the literature. Finally Section 2.3 proposes a way of relating the categorical semantics for the two kinds of two-level languages.

While writing this paper we realized that most denotational semantics of two-level languages based on the static/dynamic classification are flawed (see Remark 2.3), so in Example 2.4 we give a realizability model based on operational considerations. In an e-mail exchange Henglein has proposed an alternative denotational semantics, which is described in Example 2.5.

1 The two-level language $\lambda 2cr$

We present the syntax of two-level languages following [15], since it highlights more clearly similarities and differences. We rely also on the additional insights provided by [5,4]. In this section we consider the two-level language of [16] (see also [5]), which we call $\lambda 2cr$: we recall its syntax, propose a categorical semantics, and discuss its consistency with the parameterized semantics of [16].

Two-level languages do not come with an equational calculus, but at best they come with a semantics (either operational or denotational) giving the intended meaning.

1.1 Syntax of $\lambda 2cr$

The language $\lambda 2cr$ has two sorts c and r classifying compile-time and run-time types respectively. The language is described by the following set of formation rules, which are parameterized w.r.t. a set of base types C (of given sort b_C) and constants c (of given type τ_c of sort b_c).

- $\vdash \tau : b$, i.e. τ is a well-formed type of sort b

$$(\rightarrow^b)\ \frac{\vdash \tau_1 : b \quad \vdash \tau_2 : b}{\vdash \tau_1 \rightarrow^b \tau_2 : b} \qquad (C)\ \vdash C : b_C \qquad (\Box)\ \frac{\vdash \tau : r}{\vdash \tau : c}$$

- $\Gamma \vdash$, i.e. Γ is a well-formed context

$$(\emptyset)\ \emptyset \vdash \qquad (\text{ext})\ \frac{\Gamma \vdash \quad \vdash \tau : b}{\Gamma, x : \tau : b \vdash}\ x \notin \mathrm{D}V(\Gamma)$$

  we write Γ^b for the sub-context of Γ of sort b, i.e.

$$\emptyset^b = \emptyset \text{ and } (\Gamma, x : \tau : b')^b = \begin{cases} \Gamma^b, x : \tau : b \text{ if } b' = b \\ \Gamma^b \qquad\qquad \text{otherwise} \end{cases}$$

- $\Gamma \vdash e : \tau$, i.e. e is a well-formed term of type τ of sort b in context Γ

$$(x)\ \frac{\Gamma \vdash}{\Gamma \vdash x : \tau : b}\ \tau : b = \Gamma(x) \qquad (c)\ \frac{\Gamma \vdash}{\Gamma \vdash c : \tau_c : b_c}$$

$$(\lambda^b)\ \frac{\Gamma, x : \tau_1 : b \vdash e : \tau_2 : b}{\Gamma \vdash (\lambda^b x : \tau_1 . e) : \tau_1 \rightarrow^b \tau_2 : b} \qquad (@^b)\ \frac{\Gamma \vdash e : \tau_1 \rightarrow^b \tau_2 : b \quad \Gamma \vdash e_1 : \tau_1 : b}{\Gamma \vdash e @^b e_1 : \tau_2 : b}$$

$$(\Box\text{-I})\ \frac{\Gamma \vdash \quad \Gamma^c \vdash e : \tau : r}{\Gamma \vdash e : \tau : c} \qquad (\Box\text{-E})\ \frac{\Gamma \vdash e : \tau : c \quad \vdash \tau : r}{\Gamma \vdash e : \tau : r}$$

A key property of this two-level language is independence of compile-time expressions from run-time variables.

Proposition 1.1 *If $\Gamma \vdash e : \tau : c$ is derivable, then also $\Gamma^c \vdash e : \tau : c$ is.*

This property can be hard wired in the judgments for terms (see [5]), by distinguishing between compile-time and run-time contexts: $\Gamma^c; \Gamma^r \vdash e : \tau : r$ and $\Gamma^c \vdash e : \tau : c$. With this change the $\Box$-rules become

$$(\Box\text{-I})\ \frac{\Gamma^c; \emptyset \vdash e : \tau : r}{\Gamma^c \vdash e : \tau : c} \qquad (\Box\text{-E})\ \frac{\Gamma^c; \Gamma^r \vdash \quad \Gamma^c \vdash e : \tau : c \quad \vdash \tau : r}{\Gamma^c; \Gamma^r \vdash e : \tau : r}$$

Remark 1.2 We have ignored other features of two-level languages, e.g.: fix-point operators, or coercions $(\text{coer})\ \dfrac{\Gamma \vdash e : C^c : c}{\Gamma \vdash e : C^r : r}$ from a compile-time base type to the *corresponding* run-time base type. In fact, these features can be described by suitable choices of (base types and) constants.

1.2 Semantics of $\lambda 2cr$

We take as models of $\lambda 2cr$ indexed categories, i.e. functors $\mathcal{C} : \mathcal{B}^{op} \to \mathbf{Cat}$, s.t.: $\mathcal{B}$ is a CCC, $\mathcal{C}$ is an indexed CCC (i.e. fibers are CCC and substitution functors preserve the CCC structure) with *boxing* (see Definition 1.3). Indexed categories have been used already to model predicate logic and various form of polymorphic λ-calculus. For instance, In the case of predicate logic, a type is interpreted by an object in the base category $\mathcal{C}$, while a predicate (over a type τ) is interpreted by an object in a fiber $\mathcal{C}_I$ (where I is the interpretation of τ). Indexed categories are ideal to capture the *independence* property stated in Proposition 1.1. Following the above intuition, we give the general pattern for interpreting the judgments of $\lambda 2cr$ (in the reformulation of [5]):

- $\vdash \tau : c$ and $\Gamma^c \vdash$ are interpreted by objects b and I of the base category $\mathcal{B}$;
- $\Gamma^c \vdash e : \tau : c$ is interpreted by a morphism $f : I \to b$ in $\mathcal{B}$;
- $\vdash \tau : r$ and $\Gamma^r \vdash$ are interpreted by objects c and J of the fiber $\mathcal{C}_I$; since in $\lambda 2cr$ run-time types do not depend on compile-time variable, we make the simplifying assumption that **all fibers have the same set of objects, and substitution functors $f^* : \mathcal{C}_I \to \mathcal{C}_J$ (where $f : J \to I$ in $\mathcal{B}$) are the identity on objects**;
- $\Gamma^c; \Gamma^r \vdash e : \tau : r$ is interpreted by a morphism $g : J \to c$ in $\mathcal{C}_I$.

It is clear how to interpret contexts and functional types of $\lambda 2cr$ by exploiting the CCC structure. To interpret the $\Box$-rules we need another universal property of $\mathcal{C}$.

Definition 1.3 We say that $b \in \mathcal{B}$ is a **boxing** of $c \in \mathcal{C}_1$ iff it satisfies the following universal property: $\mathcal{B}(J, b) \cong \mathcal{C}_J(1, c)$ naturally in $J \in \mathcal{B}$

With boxing the interpretation of the $\Box$-rules goes as follows:

- $(\Box)\ \dfrac{[\![\vdash \tau : r]\!] = c \in \mathcal{C}_1}{[\![\vdash \tau : c]\!] = \Box c \in \mathcal{B}}$ where $\Box c$ is a given boxing of c

- $(\Box)\ \dfrac{[\![\Gamma^c;\emptyset \vdash e:\tau:r]\!] = g \in \mathcal{C}_I(1,c)}{[\![\Gamma^c \vdash e:\tau:c]\!] = \phi_I(g) \in \mathcal{B}(I,\Box c)}\ \vdash \tau : r$ where ϕ_I is the isomorphism $\mathcal{C}_I(1,c) \to \mathcal{B}(I,\Box c)$ in the definition of boxing. From this bi-rule (and weakening) one can derive ($\Box$-I) and ($\Box$-E) in the formulation of [5].

The definition of boxing is an instance of Lawvere's comprehension, which captures the universal property of the type $\{x : \tau|\phi(x)\}$ in relation to the predicate $\phi(x)$ over type τ, corresponding to the case $\tau = 1$.

Definition 1.4 We say that $(\pi : b \to I) \in \mathcal{B}/I$ is a **comprehension** of $c \in \mathcal{C}_I$ iff it satisfies the following universal property:

$$\mathcal{B}/I(f,\pi) \cong \mathcal{C}_J(1, f^*c) \text{ naturally in } (f : J \to I) \in \mathcal{B}/I$$

where $\mathcal{B}/I$ is the slice category over I.

The relation between boxing and comprehension is as follows.

Proposition 1.5 *If b is a boxing of $c \in \mathcal{C}_1$, then $\pi_1 : I \times b \to I$ is a comprehension of $!^*c \in \mathcal{C}_I$, where $! : I \to 1$.*

When substitution functors are the identity on objects, then any object of $\mathcal{C}_I$ is of the form $!^*c$, and so comprehension is *reduced* to boxing. However, in a two-level language where run-time types may depend on compile-time variables, Lawvere's comprehension is the correct way of modeling the $\Box$-rules.

Example 1.6 We give some ways of constructing indexed categories for interpreting $\lambda 2cr$.

- **(fam-construction)** Any CCC $\mathcal{A}$ can be turned into an indexed CCC $\mathcal{C} : \mathbf{Set}^{op} \to \mathbf{Cat}$ by taking $\mathcal{C}_I = \mathcal{A}^I$ (and $f^* = \mathcal{A}^f$). Given $\langle a_i | i \in I\rangle \in \mathcal{C}_I$ its comprehension is given by the projection $\pi : (\Sigma i : I.\mathcal{A}(1, a_i)) \to I$.
- **(constant fam-construction)** There is a variant of the fam-construction, where substitution functors are the identity on objects, namely take $\mathcal{C}_I$ with the same objects of $\mathcal{A}$ and $\mathcal{C}_I(a_1,a_2) = \mathcal{A}(a_1,a_2)^I$. Given $a \in \mathcal{C}_1$ its boxing is simply the set $\mathcal{A}(1,a)$ of global elements of a.
- **(enriched constant fam-construction)** In the constant fam-construction one can also replace **Set** with **Cpo** (the category of ω-cpos and ω-continuous maps) or some other CCC. Namely, given a **Cpo**-enriched CCC $\mathcal{A}$ (so in particular $\mathcal{A}(a_1,a_2)$ is a cpo), define the indexed CCC $\mathcal{C} : \mathbf{Cpo}^{op} \to \mathbf{Cat}$ by taking $\mathcal{C}_I$ with the same objects of $\mathcal{A}$ and $\mathcal{C}_I(a_1,a_2) = \mathbf{Cpo}(I, \mathcal{A}(a_1,a_2))$ (incidentally $\mathcal{C}_I$ has an obvious **Cpo**-enrichment). In this case, given $a \in \mathcal{C}_1$ its boxing is the cpo $\mathcal{A}(1,a)$ of global elements of a.

A test for the categorical semantics of $\lambda 2cr$ is to check whether it subsumes previously proposed semantics.

Example 1.7 (Parameterized semantics) The two-level λ-calculus of [16] uses a restricted form of $\Box$-rule $(\Box)\ \dfrac{\vdash \tau_1 \to^r \tau_2 : r}{\vdash \tau_1 \to^r \tau_2 : c}$ where only run-time functional types can be turned into compile-time types. The parameterized semantics is given for a mixed λ-calculus, which uses combinators $c : \tau_c : c$, with the restricted form of $\Box$-rule. The main point of this calculus is to avoid run-time variables. The parameterized semantics interprets compile-time types $\vdash \tau : c$ and terms $\Gamma^c \vdash e : \tau : c$ in the category **Cpo** of cpos, and is parameterized w.r.t. an interpretation $\mathcal{I}$ of combinators and *frontier* types, i.e. $\vdash \tau : c$ s.t. $\vdash \tau : r$:

- $[\![\tau_1 \to^c \tau_2]\!] = [\![\tau_1]\!] \to [\![\tau_2]\!]$ and
- $[\![\tau_1 \to^r \tau_2]\!] = \mathcal{I}(\tau_1 \to^r \tau_2)$.

From a parameterized semantics one can recover an *indexed category* by applying the enriched constant fam-construction to the **Cpo**-enriched *category* $\mathcal{A}$ s.t. the objects are run-time types $\vdash \tau : r$ and $\mathcal{A}(\tau_1, \tau_2) = \mathcal{I}(\tau_1 \to^r \tau_2)$. However, on $\mathcal{A}$ we must define the CCC structure and show that it satisfies the necessary properties. The CCC structure can be defined in terms of the combinators of the mixed λ-calculus (composition, identity,...), but in general these may fail to satisfy the necessary equational laws (associativity,...).

Remark 1.8 We can say that there is a close match between categorical and parameterized semantics on the *structure*, but not on the equational properties of the run-time *structure*: the parameterized imposes no equations, while the categorical semantics requires $\beta\eta$ for $\to^r$. This is the same kind of mismatch between (functional) programming languages on one side and $\lambda\beta\eta$-calculus (or CCC) on the other. There are simple way to bridge this gap, e.g. by translating programming languages into metalanguages with computational types, which have a *neat* categorical semantics.

2 The two-level language $\lambda 2sd$

In this section we consider the two-level language used in the context of partial evaluation (see [8] and [7,6,4]), which we call $\lambda 2sd$. More precisely, $\lambda 2sd$ is the simply typed version of the two-level language in [8] as formulated in [15]. we follow the same structure of the previous section: syntax, semantics, comparison and discussion.

Syntactically the major difference is that in $\lambda 2sd$ the independence property fails. Semantically also $\lambda 2sd$ can be modeled in terms of indexed categories, though this is not immediate from the syntax (as it is for $\lambda 2cr$). This means that the arguments to support the categorical account of $\lambda 2sd$ are more subtle (and less compelling), but on the other hand the insights gained are more *interesting*, e.g. Section 2.3 indicates a relation between $\lambda 2sd$ and $\lambda 2cr$ not observed before.

2.1 Syntax of $\lambda 2sd$

The language $\lambda 2sd$ has two sorts s and d classifying static and dynamic types respectively. The language is described by the following set of formation rules (parameterized like those of $\lambda 2cr$).

- $\vdash \tau : b$, i.e. τ is a well-formed type of sort b

$$(\to^b)\ \frac{\vdash \tau_1 : b \quad \vdash \tau_2 : b}{\vdash \tau_1 \to^b \tau_2 : b} \qquad (C)\ \vdash C : b_C \qquad (\bigcirc)\ \frac{\vdash \tau : d}{\vdash \tau : s}$$

- $\Gamma \vdash$, i.e. Γ is a well-formed context

$$(\emptyset)\ \emptyset \vdash \qquad (\mathrm{ext})\ \frac{\Gamma \vdash \quad \vdash \tau : b}{\Gamma, x : \tau : b \vdash}\ x \notin \mathrm{DV}(\Gamma)$$

- $\Gamma \vdash e : \tau$, i.e. e is a well-formed term of type τ of sort b in context Γ

$$(x)\ \frac{\Gamma \vdash}{\Gamma \vdash x : \tau : b}\ \tau : b = \Gamma(x) \qquad (c)\ \frac{\Gamma \vdash}{\Gamma \vdash c : \tau_c : b_c}$$

$$(\lambda^b)\ \frac{\Gamma, x : \tau_1 : b \vdash e : \tau_2 : b}{\Gamma \vdash (\lambda^b x : \tau_1 . e) : \tau_1 \to^b \tau_2 : b} \qquad (@^b)\ \frac{\Gamma \vdash e : \tau_1 \to^b \tau_2 : b \quad \Gamma \vdash e_1 : \tau_1 : b}{\Gamma \vdash e @^b e_1 : \tau_2 : b}$$

$$(\bigcirc\text{-I})\ \frac{\Gamma \vdash e : \tau : d}{\Gamma \vdash e : \tau : s} \qquad (\bigcirc\text{-E})\ \frac{\Gamma \vdash e : \tau : s \quad \vdash \tau : d}{\Gamma \vdash e : \tau : d}$$

Remark 2.1 If one makes the obvious identifications in the descriptions of $\lambda 2sd$ and $\lambda 2cr$ (i.e. $s = c$, $d = r$ and $\bigcirc = \square$), the only difference between the two languages is rule ($\square$-I). However, this has two major consequences for $\lambda 2sd$:

- the independence property fails, i.e. static expressions may depend on dynamic variables (and conversely);
- dynamic types are a full sub-category of static types, i.e. $x : \tau_1 : d \vdash e : \tau_2 : d$ iff $x : \tau_1 : s \vdash e : \tau_2 : s$ for any e, provided $\vdash \tau_1 : d$ and $\vdash \tau_2 : d$.

If one assumes $\beta\eta$ for $\to^s$ and $\to^d$, then dynamic types form a full sub-CCC. This is very natural, and we will make this assumption to simplify the categorical semantics. However, in the context of partial evaluation it is **not reasonable** to assume $\beta\eta$ for $\to^d$, because at *specialization-time* dynamic term-constructors (like $@^d$ and λ^d) are symbols indicating the structure of *residual programs.*

2.2 Naive semantics of $\lambda 2sd$

We take as models of $\lambda 2sd$ a CCC $\mathcal{S}$ of static types with a full sub-CCC $\mathcal{D}$ of dynamic types. Following the above intuition, we give the general pattern for interpreting the judgments of $\lambda 2sd$:

- $\vdash \tau : s$ and $\Gamma \vdash$ are interpreted by objects c and I of the static category $\mathcal{S}$;
- $\vdash \tau : d$ is interpreted by an object c of the dynamic category $\mathcal{D}$;
- $\Gamma \vdash e : \tau : b$ is interpreted by a morphism $f : I \to c$ in $\mathcal{S}$.

The interpretation of $\to^b$ is clear, since $\mathcal{D}$ and $\mathcal{S}$ are CCC. While the interpretation of the $\bigcirc$-rules rests on $\mathcal{D}$ being a full sub-category of $\mathcal{S}$.

Remark 2.2 In Section 2.3 we give a more accurate semantics of $\lambda 2sd$ in terms of indexed categories. However, the naive semantics is already of some use. For instance, consider the base types $nat^s : s$ and $nat^d : d$, whose natural interpretation is the natural number object (NNO) in $\mathcal{S}$ and $\mathcal{D}$ respectively. Since $\mathcal{D}$ is a sub-CCC of $\mathcal{S}$, there is a unique NNO-homomorphism from nat^s to nat^d, which provides the natural interpretation of the coercion from nat^s to nat^d. On the other hand, the naive semantics does not capture that the only maps from nat^d to nat^s are the constant ones. This is supported by the syntax of $\lambda 2sd$, since x does not occur free in any term $x : nat^d : d \vdash e : nat^s : s$ in β-normal form (this is proved by case analysis on the normal forms).

Remark 2.3 All denotational semantics of $\lambda 2sd$ (and similar languages) proposed in the literature have problems with α-conversion, namely α-convertible terms may have different denotations (and thus basic properties such as the substitution lemma are in jeopardy): [7,6] use a *newname* construct with an unclear status; [8] does not use *newname*, but fails to be invariant under α-conversion.

In these semantics dynamic types are interpreted by sets of open terms (over a countable set $\{x_n | n \in N\}$ of variables), and

$$[\![\Gamma \vdash (\lambda^d x : \tau_1.e) : \tau_1 \to^d \tau_2 : d]\!]_\rho = \lambda z.[\![\Gamma, x : \tau_1 : d \vdash e : \tau_2 : d]\!]_{\rho[x:=z]}$$

where z is *new*, i.e. z is not *used* in ρ. First, the interpretation cannot be invariant under α-conversion, unless we work with open terms **modulo α-conversion**. Nevertheless there are other problems, exemplified by the interpretation of the term $f : o \to^s o \vdash (\lambda^d x : o.f @^s x) : o \to^d o : d$, which cannot be overcome easily. Let $\rho(f)$ be the function $f' : [\![o]\!] \to [\![o]\!]$ s.t. $f'(M)$ is the substitution instance $M[succ]$ of M, where $succ(x_n) = x_{n+1}$ for every $n \in N$. Let's assume that there are infinitely many variables not used in ρ (with only finitely many one gets in troubles by considering a more complex term). Then different choices of a *new* z (e.g. x_m and x_n), results in terms (i.e. $\lambda x_m.x_{m+1}$ and $\lambda x_n.x_{n+1}$) which are not α-convertible!

We consider two interpretations for the two-level language of [7], which differs from $\lambda 2sd$ because there is only one dynamic type $code : d$, and the typing rules for λ^d and $@^d$ become:

$$(\lambda^d)\ \frac{\Gamma, x : code : d \vdash e : code : d}{\Gamma \vdash (\lambda^d x.e) : code : d} \qquad (@^d)\ \frac{\Gamma \vdash e : code : d \quad \Gamma \vdash e_1 : code : d}{\Gamma \vdash e @^d e_1 : code : d}$$

The first interpretation is in a realizability model based on an operational semantics (see [4]), where the global elements of the object $[\![code]\!]$ are untyped

λ-terms modulo α-conversion. The second interpretation is in the category of cpos and $[\![code]\!]$ is the solution to the domain equation $X = N + X_\perp^X + X \times X$.

Example 2.4 We outline an operational semantics and a corresponding realizability model based on a pCA (partial combinatory algebra) on *open values*, where terms should be understood modulo α-conversion.

- expressions $e \in E ::= \ x \mid \lambda^d x.e \mid \lambda^s x.e \mid e_1 @^d e_2 \mid e_1 @^s e_2$
  static expressions $e^s \in E^s ::= \ x \mid \lambda^s x.e^s \mid e_1^s @^s e_2^s$
- values $v \in V ::= \ x \mid \lambda^d x.v \mid \lambda^s x.e \mid v_1 @^d v_2$
  dynamic values $v^d \in V^d ::= \ x \mid \lambda^d x.v^d \mid v_1^d @^d v_2^d$
  static values $v^s \in V^s ::= \ x \mid \lambda^s x.e^s$

We define a CBN SOS $\Rightarrow \subset E \times V$ for reducing *static redexes*:

$$(x)\ \ x \Rightarrow x \qquad (\lambda^d)\ \frac{e \Rightarrow v}{(\lambda^d x.e) \Rightarrow (\lambda^d x.v)} \qquad (\lambda^s)\ \ (\lambda^s x.e) \Rightarrow (\lambda^s x.e)$$

$$(@^d)\ \frac{e_1 \Rightarrow v_1 \quad e_2 \Rightarrow v_2}{e_1 @^d e_2 \Rightarrow v_1 @^d v_2} \qquad (@^s)\ \frac{e_1 \Rightarrow (\lambda^s x.e) \quad [e_2/x]e \Rightarrow v}{e_1 @^s e_2 \Rightarrow v}$$

This SOS induces a pCA $(V, \cdot)$, where $v_1 \cdot v_2 = v_3 \overset{\Delta}{\iff} (v_1 @^s v_2) \Rightarrow v_3$. The static category $\mathcal{S}$ is defined as follows

- an object $\underset{\sim}{X}$ is a pair $(X, \Vdash_X)$ where X is a set and $\Vdash_X \subseteq V \times X$ is (the graph of) a partial surjection from V to X
- a morphism $f : \underset{\sim}{X} \to \underset{\sim}{Y}$ is a function $f : X \to Y$ *realized* by some $v \in V$, i.e. $v \cdot v_x \Vdash_Y f(x)$ whenever $v_x \Vdash_X x$.

$\mathcal{S}$ is a CCC (and much more) equivalent to the category of PERs over $(V, \cdot)$. We interpret the dynamic type *code* with the object $\underset{\sim}{D} = (V^d, \Vdash_D)$ where $(v \Vdash_D v^d) \overset{\Delta}{\iff} v = v^d$. The dynamic category $\mathcal{D}$ could be identified with the full sub-category whose sole object is $\underset{\sim}{D}$. $\mathcal{D}$ is not a CCC, e.g. $\underset{\sim}{D}$ is not isomorphic to $\underset{\sim}{D}^{\underset{\sim}{D}}$, but there are morphisms $\underset{\sim}{\lambda} : \underset{\sim}{D}^{\underset{\sim}{D}} \to \underset{\sim}{D}$ and $\underset{\sim}{@} : \underset{\sim}{D} \times \underset{\sim}{D} \to \underset{\sim}{D}$ corresponding to *build_*λ and *build_@* of [7]. Again there is a mismatch with the categorical semantics, not in the structure but in the equational properties (see Remark 1.8 and 2.1).

Example 2.5 The following example was suggested by Henglein in an e-mail exchange, and is a way to use **Cpo** as static category, but to avoid the problems in [7] (see Remark 2.3). In this case we interpret the dynamic type *code* with the initial solution $\underset{\sim}{D}$ to the domain equation $X = N + X_\perp^X + X \times X$, this is not a flat cpo and has uncountably many elements (unlike the interpretation of *code* in [7]). As in the previous example, $\underset{\sim}{D}$ is not isomorphic to $\underset{\sim}{D}^{\underset{\sim}{D}}$, but there are morphisms $\underset{\sim}{\lambda} : \underset{\sim}{D}^{\underset{\sim}{D}} \to \underset{\sim}{D}$ and $\underset{\sim}{@} : \underset{\sim}{D} \times \underset{\sim}{D} \to \underset{\sim}{D}$ (the obvious inclusions) corresponding to *build_*λ and *build_@* of [7].

2.3 Semantics of $\lambda 2sd$ revised

In this section we show that a model $\mathcal{D} \hookrightarrow \mathcal{S}$ for $\lambda 2sd$ can be constructed starting from an indexed category $\mathcal{C} : \mathcal{B}^{op} \to \mathbf{Cat}$ (with some additional structure). The category theory underlying the construction was already used in [13,14] to account for *phase distinction* in program modules. This result identifies an interesting link between $\lambda 2cr$ and $\lambda 2sd$ (not observed before), which can be read as saying that the classification compile-time/run-time is more *fundamental* than the classification static/dynamic. More precisely, $\lambda 2cr$ play the role of a *core language*, while the static level of $\lambda 2sd$ is the analog of a *module language*.

The key idea is that a static type should be understood as a pair consisting of a compile-time and run-time type (while dynamic types are identified with run-time types). More precisely, $\mathcal{S}$ should be identified with the *total category* $\mathcal{GC}$ (this is part of the Grothendieck construction, which turns an indexed category $\mathcal{C} : \mathcal{B}^{op} \to \mathbf{Cat}$ into a fibration $\pi_{\mathcal{C}} : \mathcal{GC} \to \mathcal{B}$).

Definition 2.6 Given an indexed category $\mathcal{C} : \mathcal{B}^{op} \to \mathbf{Cat}$, the **total category** $\mathcal{GC}$ associated to it is given by

- an object is a pair $\langle I, c\rangle$ s.t. $I \in \mathcal{B}$ and $c \in \mathcal{C}_I$
- a morphism from $\langle I, c\rangle$ to $\langle I', c'\rangle$ is a pair $\langle f, g\rangle$ s.t. $f : I \to I'$ in $\mathcal{B}$ and $g : c \to f^*(c')$ in $\mathcal{C}_I$
- composition $\langle f_2, g_2\rangle \circ \langle f_1, g_1\rangle$ is given by $\langle f_2 \circ f_1, f_1^*(g_2) \circ g_1\rangle$.

[13] gives sufficient conditions on $\mathcal{C}$ to ensure that $\mathcal{GC}$ is a CCC (more refined results are given in [14], but they are not needed here).

Theorem 2.7 *If $\mathcal{B}$ is a CCC, $\mathcal{C} : \mathcal{B}^{op} \to \mathbf{Cat}$ is an indexed CCC and has universal quantification along projections, then $\mathcal{GC}$ is a CCC, namely*

- *$\langle 1, 1\rangle$ is the terminal object*
- *$\langle I_1, c_1\rangle \times \langle I_2, c_2\rangle$ is $\langle I_1 \times I_2, (\pi_1^* c_1) \times (\pi_2^* c_2)\rangle$*
- *$\langle J, b\rangle^{\langle I, a\rangle}$ is $\langle J^I, \forall_{\pi_1}(ev^* b)^{\pi_2^* a}\rangle$, where $ev : J^I \times I \to J$ is the evaluation map and π_i are the projections out of $J^I \times I$.*

For instance, when $\mathcal{C} : \mathbf{Set}^{op} \to \mathbf{Cat}$ is given by the fam-construction applied to $\mathbf{Set}$, then $\mathcal{GC}$ is s.t.

- an object is a family $\langle X_i | i \in I\rangle$ of sets
- a morphism from $X = \langle X_i | i \in I\rangle$ to $Y = \langle X_j | j \in J\rangle$ is a pair $\langle f, g\rangle$ s.t. $f : I \to J$ and $g \in \Pi i : I.X_i \to Y_{fi}$
- the exponential Y^X is the family $\langle \Pi i : I.Y_{fi}^{X_i} | f \in J^I\rangle$.

Example 2.8 We reconsider the examples given in Section 1.2, and discuss what additional properties must be imposed on $\mathcal{A}$ to ensure universal quantification along projections, so that the indexed category $\mathcal{C} : \mathcal{B}^{op} \to \mathbf{Cat}$ can be turned into a model $\lambda 2sd$ as described in Theorem 2.7:

- in the fam-construction $\mathcal{A}$ must have small products, then $\forall_I : \mathcal{A}^I \to \mathcal{A}$ maps $\langle a_i | i \in I\rangle$ to $\Pi_{i\in I} a_i$;
- in the constant fam-construction $\mathcal{A}$ must have small powers, i.e. for any $a \in \mathcal{A}$ and set I exists $(I \Rightarrow a)$ s.t. $\mathcal{A}(x, I \Rightarrow a) \cong \mathcal{A}(x, a)^I$ naturally in x;
- in the enriched constant fam-construction $\mathcal{A}$ must have small powers in the enriched sense, i.e. $\mathcal{A}(x, I \Rightarrow a) \cong \mathcal{A}(x, a)^I$ as cpos.

In the total category one can identify two *exponential ideals* corresponding to compile-time and run-time types respectively (an exponential ideal $\mathcal{A}$ of a CCC $\mathcal{B}$ is a full sub-CCC s.t. $a^b \in \mathcal{A}$ whenever $a \in \mathcal{A}$ and $b \in \mathcal{B}$).

Proposition 2.9 *Under the assumptions of Theorem 2.7 $\mathcal{B}$ and $\mathcal{C}_1$ are (equivalent to) exponential ideals of $\mathcal{GC}$*

$$\begin{tikzcd} \mathcal{B} \arrow[r, hook, bend right] \arrow[r, phantom, "\perp"] & \mathcal{GC} \arrow[l, dashed, bend right, "\pi_{\mathcal{C}}"'] & \mathcal{GC} & \mathcal{C}_1 \arrow[l, hook'] \\ I \arrow[r, mapsto] & \langle I, 1\rangle & \langle 1, c\rangle & c \arrow[l, mapsto] \end{tikzcd}$$

and both inclusions preserve and reflect limits.

Remark 2.10 This proposition and Theorem 2.7 say that a model of $\lambda 2cr$ (we need also universal quantification along projections but no boxing) can be turned into a model of $\lambda 2sd$ by taking

$$\mathcal{D} = \mathcal{C}_1 \hookrightarrow \mathcal{GC} = \mathcal{S}.$$

Intuitively this says: dynamic types correspond to run-time types (i.e. the objects of the fiber $\mathcal{C}_1$), while static types are pairs (τ^c, τ^r) consisting of a compile-time and a run-time type (more generally a compile-time type τ^c and a family of run-time types indexed over τ^c).

If we make the simplifying assumption (see Section 1.2) that substitution functors are the identity on objects (and the assumptions of Theorem 2.7 hold), then

- universal quantification along projects amounts to have for every $I \in \mathcal{B}$ and $c \in \mathcal{C}_1$ a $(I \Rightarrow c) \in \mathcal{C}_1$ satisfying the universal property:

$$\mathcal{C}_{J\times I}(b, c) \cong \mathcal{C}_J(b, I \Rightarrow c) \text{ naturally in } J \in \mathcal{B} \text{ and } b \in \mathcal{C}_J$$

- any object $\langle I, c\rangle$ of $\mathcal{GC}$ is (isomorphic to) the product $\langle I, 1\rangle \times \langle 1, c\rangle$ of an object coming from $\mathcal{B}$ and one coming from $\mathcal{C}_1$.

Remark 2.11 In $\mathcal{GC}$ the compile-time objects $\langle I, 1\rangle$ are *orthogonal* to the run-time objects $\langle 1, c\rangle$, i.e. the canonical map $\langle I, 1\rangle \to \langle I, 1\rangle^{\langle 1,c\rangle}$ is an iso. The following result exploit this observation to give a way for turning a model $\mathcal{D} \hookrightarrow \mathcal{S}$ of $\lambda 2sd$ into a model $\mathcal{C} : \mathcal{B}^{op} \to \mathbf{Cat}$ of $\lambda 2cr$ (without boxing), and to show that this is a partial inverse to the construction going from $\lambda 2cr$ to $\lambda 2sd$ (and described above).

Theorem 2.12 *Given a model $\mathcal{D} \hookrightarrow \mathcal{S}$ of $\lambda 2sd$, we can define an indexed category $\mathcal{C} : \mathcal{B}^{op} \to \mathbf{Cat}$ satisfying the assumptions of Theorem 2.7 s.t. the substitution functors are the identity on objects and there is a chain of full CCC-embeddings $\mathcal{D} \hookrightarrow \mathcal{C}_1 \hookrightarrow \mathcal{GC} \hookrightarrow \mathcal{S}$.*

Proof. Let $\mathcal{R}$ be the exponential ideal of $\mathcal{S}$ generated by $\mathcal{D}$, its objects are (isomorphic to) finite products of powers d^s with $d \in \mathcal{D}$. Let $\mathcal{B}$ be the category of objects in $\mathcal{S}$ orthogonal to $\mathcal{R}$, i.e. $I \in \mathcal{B}$ iff for all $r \in \mathcal{R}$ the canonical map $I \to I^r$ is an iso. One can prove that also $\mathcal{B}$ is an exponential ideal of $\mathcal{S}$.

Define $\mathcal{C} : \mathcal{B}^{op} \to \mathbf{Cat}$ as the indexed category s.t. the objects of $\mathcal{C}_I$ are those of $\mathcal{R}$ and $\mathcal{C}_I(r_1, r_2) = \mathcal{S}(I \times r_1, r_2)$. Clearly the substitution functors are the identity on objects. One can prove that the assumptions of Theorem 2.7 are satisfied, e.g. $(I \Rightarrow r) = r^I$.

Finally, we have to show the chain of full CCC-embeddings. the only subtle point is to show that $\langle I, r\rangle \mapsto I \times r$ gives a full CCC-embedding $\mathcal{GC} \hookrightarrow \mathcal{S}$, in particular that it preserves exponentials, i.e. $J^I \times (b^a)^I \cong (J \times b)^{I \times a}$. This follows from the isomorphisms $(b^a)^I \cong b^{I \times a}$ (true in any CCC) and $J^I \cong J^{I \times a}$, which is true because J^I is orthogonal to a. □

Proposition 2.13 *If $\mathcal{C} : \mathcal{B}^{op} \to \mathbf{Cat}$ satisfies the assumptions of Theorem 2.7 and its substitution functors are the identity on objects, then the construction of Theorem 2.12 applied to $\mathcal{C}_1 \hookrightarrow \mathcal{GC}$ gives an indexed category equivalent to $\mathcal{C} : \mathcal{B}^{op} \to \mathbf{Cat}$.*

Proof. Since $\mathcal{C}_1$ is already an exponential ideal of $\mathcal{GC}$, then $\mathcal{R}$ and $\mathcal{C}_1$ are equivalent. We know already that all $\langle I, 1\rangle$ are orthogonal to $\mathcal{R}$, thus we have to prove that (up to isomorphism) these are the only possibilities, i.e. if $\langle I, c\rangle$ is orthogonal to $\mathcal{R}$ then $\langle I, c\rangle \cong \langle I, 1\rangle$. Since $\langle I, c\rangle$ is orthogonal to $\langle 1, c\rangle$, we have that in $\mathcal{C}_I$ the canonical map $c \to c^c$ is an iso. Therefore c is isomorphic to 1 in $\mathcal{C}_I$ (but not necessarily in $\mathcal{C}_1$!). Thus we can conclude that $\mathcal{B}$ is equivalent to the category of objects orthogonal to $\mathcal{R}$. □

Example 2.14 When the construction of Theorem 2.12 is applied to the naive operational model $\mathcal{D} \hookrightarrow \mathcal{S}$ of Example 2.4, we get a non-trivial category $\mathcal{B}$: any $\underline{X} \in \mathcal{S}$ s.t. $\Vdash_X \subseteq V_0^s \times X$ is in $\mathcal{B}$ (V_0^s is the set of closed static values). The proof of this is a bit tedious and we skip it. Thus $\mathcal{B}$ contains as a sub-CCC the category equivalent to that of PERs over the sub-pCA $(V_0^s, \cdot)$. When the construction is applied to the model of Example 2.5, $\mathcal{B}$ degenerates.

Conclusions

This paper proposes a more abstract way of looking at two-level languages via category theory. A more abstract view can be useful to researchers and practitioners working on two-level languages in several ways:

- It suggests new extensions as discussed in the introduction, category theory is particularly suitable for establishing analogies. Some of these are considered in [1]. Of course only experimentation and a more concrete study can tell which extensions are really useful.

- It suggests a wider class of models for existing two-level languages (as shown in the examples), where one could find *better* models. In particular, we have given a general way of constructing a model of $\lambda 2sd$ from a model of $\lambda 2cr$. Moreover, we expect functor categories to provide other interesting denotational models for two-level languages used in partial evaluation.

On the negative side, we have pointed out that there are some mismatches between a straightforward categorical interpretation (e.g. interpreting $\rightarrow^r$ and $\rightarrow^d$ as exponentiation) and what one usually does in two-level languages.

References

[1] G. Belle' and E. Moggi. Typed intermediate languages for shape analysis. In *TLCA '97*, volume ?? of *LNCS*. Springer Verlag, 1997.

[2] O. Danvy. Type-directed partial evaluation. In Steele [17].

[3] O. Danvy, R. Gluck, and P. Thielmann, editors. *Workshop on partial evaluation*, volume 1110 of *LNCS*. Spinger-Verlag, 1996.

[4] R. Davies. A temporal-logic approach to binding-time analysis. In *LICS*, 1996.

[5] R. Davies and F. Pfenning. A modal analysis of staged computation. In Steele [17].

[6] K. Gomard. A self-applicable partial evaluator for the lambda calculus. *ACM Trans. on Progr. Lang. and Systems*, 14(2), 1992.

[7] K. Gomard and N. Jones. A partial evaluator for the untyped lambda calculus. *J. of Func. Program.*, 1(1), 1991.

[8] F. Henglein and C. Mossin. Polymorphic binding-time analysis. In D. Sanella, editor, *ESOP'94*, volume 788 of *LNCS*. Springer Verlag, 1994.

[9] J. Hughes. Type specialization for the lambda calculus. In Danvy et al. [3].

[10] J.M.E. Hyland, E.P. Robinson, and G. Rosolini. The discrete objects in the effective topos. *Proc. London Math. Soc.*, 60, 1990.

[11] J. Launchbury. Dependent sums express separation of binding times. In K. Davis and J. Hughes, editors, *Functional Programming, Glasgow, Scotland, 1989*, pages 238–253. Springer Verlag, 1990.

[12] J. Launchbury. *Projection Factorisations in Partial Evaluation*. CUP, 1991.

[13] E. Moggi. A category-theoretic account of program modules. In *CTCS 1989*, volume 530 of *LNCS*. Springer Verlag, 1989.

[14] E. Moggi. A category-theoretic account of program modules. *Mathematical Structures in Computer Science*, 1, 1991.

[15] F. Nielson. Multi-level lambda-calculi. In Danvy et al. [3].

[16] F. Nielson and H.R. Nielson. *Two-Level Functional Languages*. Number 34 in Cambridge Tracts in Theoretical Computer Science. CUP, 1992.

[17] G.L. Steele, editor. *POPL'96*. ACM Press, 1996.