Lecture Notes for

Monad-Based Programming

Recorded by Hans-Peter Deifel (hpd@hpdeifel.de)
Edited by Sergey Goncharov (sergey.goncharov@fau.de)

by PD Dr. Sergey Goncharov

July 19, 2019

Contents

1 Semantic Origins

In mathematics we do not distinguish between *expressions* and their *meanings*. The meaning of $2 + 2$ is 4 and both objects are indistinguishable. In computer science we <u>do</u> distinguish expressions or terms from their meanings, for which we use *semantic brackets*

$$[\![-]\!]\colon \text{Terms} \to \text{Meanings}$$

The style of semantics involving such brackets is called *denotational semantics*: Denotational semantics has been developed in 70's by Christopher Strachey and Dana Scott.

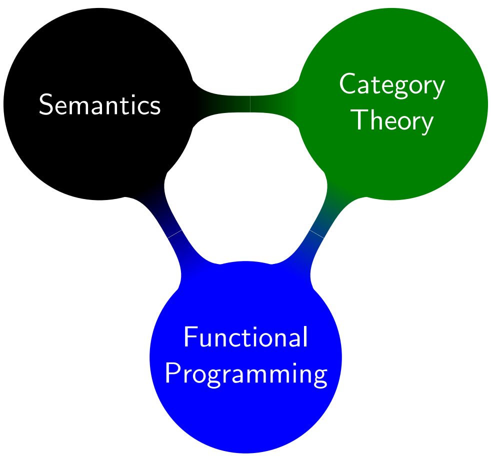
Probably the best way to illustrate the essence of the denotational (and other) semantics is by giving semantics of languages based on the λ-calculus.

Classical styles of semantics

- Denotational Semantics (what the program means?)
- Operational Semantics (how the program behaves?)
- Axiomatic Semantics (what properties the program satisfies?)

We stick to the first two styles of semantics, of which we first consider the second one (which is easier) to approach the first one (which is harder). Example of axiomatic semantics is *Hoare logic* (not covered here).

What we do in the course? The course revolves around the triad:



Starting from one node you will be able to connect to the other nodes, transferring the knowledge and understanding.

• Denotational semantics is motivated by computation and ultimately involves advanced mathematical structures, for which category theory is arguably the most natural language to use. We thus transfer computational intuition from semantics to category theory to approach the latter.

• Good understanding of semantics helps in functional programming, in particular Haskell, since it has been designed by computer scientists who took semantics very seriously. We thus learn Haskell in a semantic-oriented way.

• Category theory influenced semantics, since many abstract, purely mathematical concepts, such as *monads*, were utilized in semantics to organize constructions and reasoning. We thus use semantics to develop a computational intuition of formal categorical concepts.

• Similarly, a great amount of abstract categorical concepts was utilized in functional programming, again, most notably by Haskell. Specifically, monads were introduced to Haskell as a practical organization tool for writing programs – even writing the "Hello World" program in Haskell requires a monad!

• Therefore, in this course, conversely, we use Haskell as a showcase for advanced categorical concepts, such as monads, adjunctions, Cartesian closure.

• Semantically, Haskell is a statically typed, purely functional lazy programming language, which can be regarded as a far-reaching generalization of the typed λ-calculus, and as such it provides as excellent playground for illustrated various important semantics concepts.

1.1 The Untyped Lambda Calculus

Untyped λ-calculus is a proto-programming language introduced by a mathematician *Alonzo Church* in 30's prior to any actual programming languages and computers.

Variables	$x, y, z, \dots$
Terms	$t, s \coloneqq x, y, z \mid \lambda x.\, t \mid ts$

- α-conversion $\lambda x.\, t \longrightarrow_\alpha \lambda y.\, t[y/x]$, where y is not free in t (see definition below)
- β-reduction $(\lambda x.\, t)s \longrightarrow_\beta t[s/x]$
- η-reduction] $\lambda x. fx \longrightarrow_\eta f$
- Derived reductions
- – $\alpha\beta$-reduction is: $\longrightarrow^\star_{\alpha\beta} = (\to_\alpha \cup \to_\beta)^\star$
- – $\alpha\beta\eta$-reduction is: $\longrightarrow^\star_{\alpha\beta\eta} = (\to_\alpha \cup \to_\beta \cup \to_\eta)^\star$

Definition (Free Variables)**.**

- $\mathrm{Free}(x) = \{x\}$
- $\mathrm{Free}(st) = \mathrm{Free}(s) \cup \mathrm{Free}(t)$
- $\mathrm{Free}(\lambda x.s) = \mathrm{Free}(s) \setminus \{x\}$

A variable x is *free* in t, if $x \in \mathrm{Free}(t)$. A variable x is *bound* in t, if $x \notin \mathrm{Free}(t)$.

Definition (Substitution)**.**

- $x[t/x] = t$;
- $x[t/y] = x$ if $x \neq y$;
- $(pq)[t/x] = p[t/x]q[t/x]$;
- $(\lambda x.\, p)[t/x] = \lambda x.\, p$;
- $(\lambda y.\, p)[t/x] = \lambda z.\, p[z/y][t/x]$ if $z \notin \mathrm{Free}(\lambda y.\, p) \cup \mathrm{Free}(t)$.

Example. $(\lambda x.\, yx)[yx/y] = \lambda z.(yx)[z/x][yx/y] = \lambda z.(yz)[yx/y] = \lambda z.(yx)z$.

Proposition (Diamond Property, aka Confluence)**.** Independent reductions starting from the same term can always eventually be joined in the following sense:

$$\begin{tikzcd}
& t \arrow[dl, "\alpha\beta"', "*"] \arrow[dr, "\alpha\beta", "*"'] & \\
t_1 \arrow[dr, dashed, "*", "\alpha\beta"'] & & t_2 \arrow[dl, dashed, "*"', "\alpha\beta"] \\
& s &
\end{tikzcd}
\qquad
\begin{tikzcd}
& t \arrow[dl, "\alpha\beta\eta"', "*"] \arrow[dr, "\alpha\beta\eta", "*"'] & \\
t_1 \arrow[dr, dashed, "*", "\alpha\beta\eta"'] & & t_2 \arrow[dl, dashed, "*"', "\alpha\beta\eta"] \\
& s &
\end{tikzcd}$$

Proposition. $\longrightarrow^{\star}_{\alpha\beta}$ is not terminating:

Proof. Since $\Omega = (\lambda x.\, xx)(\lambda x.\, xx) \longrightarrow_{\beta} (\lambda x.\, xx)(\lambda x.\, xx) = \Omega$, we obtain and infinite reduction $\Omega \to_{\beta} \Omega \to_{\beta} \ldots$ $\square$

Definition (Fixpoint Combinator)**.**

$$\begin{aligned}
&Y = \lambda f.(\lambda y.f(yy))(\lambda x.f(xx)) \\
&Yf \longrightarrow_{\beta} (\lambda y.f(yy))(\lambda x.f(xx)) \longrightarrow_{\beta} f(Yf)
\end{aligned}$$

Definition (Church Numerals)**.**

$$\begin{aligned}
\underline{0} &= \lambda f.\, \lambda z.z \\
\underline{1} &= \lambda f.\, \lambda z.fz \\
\underline{2} &= \lambda f.\, \lambda z.ffz \\
&\vdots
\end{aligned}$$

In a similar way one can define $+$, $-$, *True*, *False*, *if-then-else*, etc.

1.2 Evaluation Strategies

We specify evaluation strategies with rules of structural operational semantics (SOS).

1.2.1 Standard Evaluation Strategy

The order imposed here is called *left-most-outermost-ordering*.

$$\frac{}{(\lambda x.\, p)q \longrightarrow_{\text{so}} p[q/x]} \qquad \frac{p \longrightarrow_{\text{so}} p' \qquad p \neq \lambda y.\, t}{pq \longrightarrow_{\text{so}} p'q} \qquad \frac{p \longrightarrow_{\text{so}} p'}{\lambda x.\, p \longrightarrow_{\text{so}} \lambda x.\, p'}$$

$$\frac{q \longrightarrow_{\text{so}} q' \qquad p \downarrow_{\text{so}} \qquad p \neq \lambda x.\, t}{pq \longrightarrow_{\text{so}} pq'}$$

where $p \downarrow_{\text{so}}$ means that p is irreducible with respect to $\longrightarrow_{\text{so}}$, i.e. p is so-normal.

This style of reductions is also called *small-step semantics* because in order to find an so-normal form p' of some p we generally need a chain of reductions $p \longrightarrow_{\text{so}} \ldots \longrightarrow_{\text{so}} p'$.

Definition. Using these rules, we define $p \downarrow_{\text{so}} v$, if there is a derivation of $p \longrightarrow_{\text{so}} v$ and v is so-normal.

Example.

$$\frac{\dfrac{}{(\lambda x.\, xy)(\lambda x.\, x) \longrightarrow_{\text{so}} (\lambda x.\, x)y} \qquad y \downarrow_{\text{so}} \qquad y \neq \lambda x.\, t}{y((\lambda x.\, xy)(\lambda x.\, x)) \longrightarrow_{\text{so}} y((\lambda x.\, x)y)}$$

Proposition (Standardization Theorem[1]). If $s \longrightarrow^{\star}_{\alpha\beta} t$ and t is $\alpha\beta$−normal, then $s \longrightarrow^{\star}_{\text{so}} t$ and t is so-normal.

Note the following.

- The definition of $\longrightarrow_{\text{so}}$ is *structural*, i.e. a sucessor of a term t w.r.t. $\longrightarrow_{\text{so}}$ is calculated by structural induction over t.
- The relation $\longrightarrow_{\text{so}}$ is *deterministic* in the sense that there is only one way to build a (possibly nonterminating) reduction starting from a given t; this contrasts $\alpha\beta$-reduction: we both have $(\lambda x.\lambda y.\, y)\Omega \longrightarrow_{\beta} \lambda y.\, y$ and
$$(\lambda x.\lambda y.\, y)\Omega \longrightarrow_{\text{so}} (\lambda x.\lambda y.\, y)\Omega \longrightarrow_{\text{so}} \cdots$$
- The standartization theorem indicates that all existing $\alpha\beta$-normal forms can be calculated by the standard evaluation, e.g. $(\lambda x.\lambda y.\, y)\,\Omega \longrightarrow_{\text{so}} \lambda y.\, y$ and $\lambda y.\, y \downarrow_{\text{so}}$.
- As a consequence of the previous clause $\longrightarrow_{\text{so}}$ diverges on a term t iff t does not have an $\alpha\beta$-normal form.

[1] Hendrik Pieter Barendregt. *The Lambda calculus: Its syntax and semantics.* Amsterdam: North-Holland, 1984.

1.2.2 Call-by-Name (Lazy) Evaluation Strategy

Lazy or *call-by-name (CBN)* evaluation strategy refines and simplifies the standard evaluation strategy as follows:

$$\frac{}{(\lambda x.\, p)q \longrightarrow_{\text{cbn}} p[q/x]} \qquad\qquad \frac{p \longrightarrow_{\text{cbn}} p'}{pq \longrightarrow_{\text{cbn}} p'q}$$

That is, we assume

- no rewriting under λ (therefore $\lambda x.\, \Omega \downarrow_{cbn}$);
- all terms are closed.

We thus reject η-reduction, in order to capture the fundamental distinction between *computations* and *values*. Roughly, a λ-term p represents a program, and $\lambda x.\, px$ represents its program code. While p can diverge, $\lambda x.\, p$ cannot diverge, because it is just a text of the program. However $\lambda x.\, p$ can be *executed* with β-reduction, which then can again result in divergence.

Proposition. Like SO, CBN does not diverge on terms which have $\alpha\beta$-normal forms, but CBN-normal forms need not be $\alpha\beta$-normal forms, e.g. $\lambda x.\, (\lambda y.\, y)x \downarrow_{cbn}$ but $\lambda x.\, (\lambda y.\, y)x \rightarrow_{\alpha\beta} \lambda x.\, x$.

Definition (Redex)**.** A *redex (=reducible expression)* is a subterm of the form $(\lambda x.\, t)s$ of a given term, which can be reduced with an evaluation strategy at hand.

Example.

$$\begin{aligned}
& (\lambda x.\, xx)((\lambda x.\, x)(\lambda x.\, x)) \\
\longrightarrow_{\text{cbn}}\ & ((\lambda x.\, x)(\lambda x.\, x))((\lambda x.\, x)(\lambda x.\, x)) \\
\longrightarrow_{\text{cbn}}\ & (\lambda x.\, x)((\lambda x.\, x)(\lambda x.\, x)) \\
\longrightarrow_{\text{cbn}}\ & (\lambda x.\, x)(\lambda x.\, x) \\
\longrightarrow_{\text{cbn}}\ & (\lambda x.\, x)
\end{aligned}$$

1.2.3 Call-by-Value (Eager) Evaluation Strategy

Definition (Value)**.** A value is a term of the form $\lambda x.\, t$.

Under the same assumption as with CBN we define the *call-by-value (CBV)* evaluation strategy:

$$\frac{p \longrightarrow_{\text{cbv}} p'}{pq \longrightarrow_{\text{cbv}} p'q} \qquad \frac{q \longrightarrow_{\text{cbv}} q' \qquad p \ \text{ is a value}}{pq \longrightarrow_{\text{cbv}} pq'} \qquad \frac{q \ \text{ is a value}}{(\lambda x.\, p)q \longrightarrow_{\text{cbv}} p[q/x]}$$

instead of "p is a value", one could write $p \downarrow_{\text{cbv}}$.

Proposition. CBV calculates *properly fewer* normal forms than CBN, e.g. $(\lambda x.\lambda y.\, y)\Omega \downarrow_{cbn} \lambda y.\, y$, but

$$(\lambda x.\lambda y.\, y)\,\Omega \longrightarrow_{\mathrm{cbv}} (\lambda x.\lambda y.\, y)\,\Omega \longrightarrow_{\mathrm{cbv}} \cdots$$

However, CBV is generally more efficient than CBN.

Example.

$$\begin{aligned}
& (\lambda x.\, xx)((\lambda x.\, x)(\lambda x.\, x)) \\
\longrightarrow_{\mathrm{cbv}}\ & (\lambda x.\, xx)(\lambda x.\, x) \\
\longrightarrow_{\mathrm{cbv}}\ & (\lambda x.\, x)(\lambda x.\, x) \\
\longrightarrow_{\mathrm{cbv}}\ & (\lambda x.\, x)
\end{aligned}$$

1.2.4 Big-Step Call-by-Name

In big-step styles of semantics we relate a term not to its one-step successor, but directly to its normal form.

$$\frac{}{\lambda x.\, p \Downarrow_{\mathrm{cbn}} \lambda x.\, p} \qquad \frac{p \Downarrow_{\mathrm{cbn}} \lambda x.\, p' \qquad p'[q/x] \Downarrow_{\mathrm{cbn}} c}{pq \Downarrow_{\mathrm{cbn}} c}$$

Proposition. $p \longrightarrow^{\star}_{\mathrm{cbn}} q$ and $q \downarrow_{\mathrm{cbn}}$ iff $p \Downarrow_{\mathrm{cbn}} q$.

Proving this requires the following

Lemma. $p \longrightarrow_{\mathrm{cbn}} q$ with $q \Downarrow_{\mathrm{cbn}} r$ imply $p \Downarrow_{\mathrm{cbn}} r$.

Proof. Induction over the proof of $p \longrightarrow_{\mathrm{cbn}} q$:

Induction base: $p = \lambda x.\, t$, $p = \lambda x.\, t$. Then $r = \lambda x.\, t$ and we are trivially done.

Induction step: $p = st$, $q = s't$ and $s \longrightarrow_{\mathrm{cbn}} s'$. By assumption, $s't \Downarrow_{\mathrm{cbn}} r$, which implies $s' \Downarrow_{\mathrm{cbn}} \lambda x.\, u$, $u[t/x] \Downarrow_{\mathrm{cbn}} r$. By induction, $s \Downarrow_{\mathrm{cbn}} \lambda x.\, u$. Hence $st \Downarrow_{\mathrm{cbn}} r$, as required. □

1.2.5 Big-Step Call-by-Value

Call-by-value requires evaluation of arguments of function application:

$$\frac{}{\lambda x.\, p \Downarrow_{\mathrm{cbv}} \lambda x.\, p} \qquad \frac{p \Downarrow_{\mathrm{cbv}} \lambda x.p' \qquad q \Downarrow_{\mathrm{cbv}} q' \qquad p'[q'/x] \Downarrow_{\mathrm{cbv}} c}{pq \Downarrow_{\mathrm{cbv}} c}$$

Proposition. $p \longrightarrow^{\star}_{\mathrm{cbv}} q$ and $q \downarrow_{\mathrm{cbv}}$ iff $p \Downarrow_{\mathrm{cbv}} q$.

Example.

$$\dfrac{\dfrac{}{\lambda x.\, xx \Downarrow_{\mathrm{cbv}} \lambda x.\, xx} \qquad \dfrac{\dfrac{}{\lambda x.\, x \Downarrow_{\mathrm{cbv}} \lambda x.\, x} \quad \dfrac{}{\lambda x.\, x \Downarrow_{\mathrm{cbv}} \lambda x.\, x} \quad \dfrac{}{\lambda x.\, x \Downarrow_{\mathrm{cbv}} \lambda x.\, x}}{(\lambda x.\, x)(\lambda x.\, x) \Downarrow_{\mathrm{cbv}} \lambda x.\, x} \qquad \dfrac{}{(\lambda x.\, x)(\lambda x.\, x) \Downarrow_{\mathrm{cbv}} \lambda x.\, x}}{(\lambda x.\, xx)((\lambda x.\, x)(\lambda x.\, x)) \Downarrow_{\mathrm{cbv}} \lambda x.\, x}$$

1.3 PCF (Programming Computable Functions)

1.3.1 Simply-Typed λ-calculus

$$\text{Type} \coloneqq \underbrace{A, B, C, \ldots}_{\text{base types}} \mid \underbrace{1}_{\substack{\text{unit}\\\text{type}}} \mid A \times B \mid A \to B$$

Proposition. $\Omega = (\lambda x.\, xx)(\lambda x.\, xx)$ is not typable, and hence not a valid term.

Proof. By contradiction: if $x \colon A$ then $xx \colon A$ and $x \colon A \to A$, hence $A = A \to A$, contradiction. $\square$

Proposition. $\longrightarrow_{\alpha\beta}$ is strong normalising for simply typed λ-calculus.

PCF is obtained from the simply typed λ-calculus by

- adding the fixpoint combinator $Y_A \colon (A \to A) \to A$ for every type α;
- fixing *Nat* and *Bool* as the base types;
- postulating the corresponding signature of arithmetic and logical operations.

Definition (Terms-In-Context)**.** A *term in context* has the form

$$\Gamma \vdash t \colon A,$$

where A is a type and Γ is a context, which is a list of pairs $x_i \colon A_i$ such that x_i occur non-repetitively.

We work only with those $\Gamma \vdash t \colon A$ which are derivable using the following rules:

$$\textbf{(Var)}\ \frac{x \colon A \quad \text{is in } \Gamma}{\Gamma \vdash x \colon A} \qquad \textbf{(1I)}\ \frac{}{\Gamma \vdash \star \colon 1} \qquad (\times\ \mathbf{I})\ \frac{\Gamma \vdash t \colon A \qquad \Gamma \vdash s \colon B}{\Gamma \vdash \langle t, s\rangle \colon A \times B}$$

$$(\times\ \mathbf{E}_1)\ \frac{\Gamma \vdash t \colon A \times B}{\Gamma \vdash \mathsf{fst}\, t \colon A} \qquad (\times\ \mathbf{E}_2)\ \frac{\Gamma \vdash t \colon A \times B}{\Gamma \vdash \mathsf{snd}\, t \colon B}$$

$$(\to \mathbf{I})\ \frac{\Gamma, x \colon A \vdash t \colon B}{\Gamma \vdash \lambda x.\, t \colon A \to B} \qquad (\to \mathbf{E})\ \frac{\Gamma \vdash s \colon A \to B \qquad \Gamma \vdash t \colon A}{\Gamma \vdash st \colon B}$$

$$\textbf{(Const)}\ \frac{}{\Gamma \vdash c \colon A} \qquad \textbf{(Fun)}\ \frac{\Gamma \vdash t_1 \colon A_1 \quad \cdots \quad \Gamma \vdash t_n \colon A_n}{\Gamma \vdash f(t_1, \ldots, t_n) \colon B}$$

where $c \in \{\mathit{True}, \mathit{False}\} \cup \{0, 1, \ldots\}$ where $f \in \{\wedge, \vee, \neg, +, -, \ldots\}$

$$\textbf{(Eq)}\quad \frac{\Gamma \vdash s\colon A \qquad \Gamma \vdash t\colon A \qquad A \in \{\mathit{Bool}, \mathit{Nat}, 1\}}{\Gamma \vdash s = t\colon \mathit{Bool}}$$

$$\textbf{(If)}\quad \frac{\Gamma \vdash b\colon \mathit{Bool} \qquad \Gamma \vdash s\colon A \quad \Gamma \vdash t\colon A}{\Gamma \vdash \mathsf{if}\ b\ \mathsf{then}\ s\ \mathsf{else}\ t\colon A} \qquad\qquad \textbf{(Fix)}\quad \frac{\Gamma \vdash f\colon A \to A}{\Gamma \vdash Y_A f\colon A}$$

Definition (Term)**.** A PCF *term* t is obtained from $\Gamma \vdash t\colon A$ by removing the return type A and the context Γ.

The PCF syntax corresponds to the Haskell syntax rather accurately, e.g.:

```
-- | single element () of the unit type ()
() :: ()

-- | first component of a pair
fst                                 :: (a,b) -> a
fst (x,_)                           =  x

-- | second component of a pair
snd                                 :: (a,b) -> b
snd (_,y)                           =  y

-- | logical constants
True                                :: Bool
False                               :: Bool

-- | Numeric constants
0                                   :: Num a => a
42                                  :: Num a => a

-- | lambda-abstraction, assuming f :: a -> b
\x -> f x                           :: a -> b

-- | application, assuming f :: a -> b, x :: a
f x                                 :: b

-- | equality
(==)                                :: Eq a => a -> a -> Bool

-- | if-then-else, assuming b :: Bool, x :: a, y :: a
if b then a else b                  :: a

-- | fixpoint operator is definable:
```

```
fix                             :: (a -> a) -> a
fix f                           = f(fix f)
```

1.3.2 Call-by-Name Operational Semantics for PCF

We modify the concept of value as follows.

Definition (Value)**.** A value is a Boolean, a natural number, $\star$, a pair of closed terms or a closed term $\lambda x.\, t$.

The call-by-name semantics for PCF is obtained by modifying the call-by-name semantics of λ-calculus. We discuss the most important/nontrivial rules.

$$\frac{t \Downarrow \langle p, q\rangle \qquad p \Downarrow c}{\mathsf{fst}\, t \Downarrow c} \qquad\qquad \frac{t \Downarrow \langle p, q\rangle \qquad q \Downarrow c}{\mathsf{snd}\, t \Downarrow c}$$

which means that pairing is lazy. Note that there is no rule for reducing which $\langle t, s\rangle$, is, by definition, already a value. Hence, in particular, $\mathsf{fst}\langle 1, \Omega\rangle \Downarrow 1$, but $\mathsf{snd}\langle 1, \Omega\rangle$ diverges.

$$\frac{b \Downarrow \mathit{True} \qquad p \Downarrow c}{\mathsf{if}\ b\ \mathsf{then}\ p\ \mathsf{else}\ q \Downarrow c} \qquad\qquad \frac{b \Downarrow \mathit{False} \qquad q \Downarrow c}{\mathsf{if}\ b\ \mathsf{then}\ p\ \mathsf{else}\ q \Downarrow c}$$

The rules for application and abstraction are as in the λ-calculus.

$$\frac{p \Downarrow c_1 \qquad q \Downarrow c_2}{p + q \Downarrow c_1 + c_2}$$

Variant 1 for $\vee$: $\quad \dfrac{b \Downarrow \mathit{True}}{b \vee c \Downarrow \mathit{True}} \qquad \dfrac{c \Downarrow \mathit{True}}{b \vee c \Downarrow \mathit{True}} \qquad \dfrac{b \Downarrow \mathit{False} \qquad c \Downarrow \mathit{False}}{b \vee c \Downarrow \mathit{False}}$

This is known as *"parallel or"* and it does make certain sense, but in our case it would make the semantics unintentionally non-deterministic. So, we use the following one.

Variant 2 for $\vee$: $\quad \dfrac{b \Downarrow \mathit{True} \qquad c \Downarrow d}{b \vee c \Downarrow \mathit{True}} \qquad \dfrac{b \Downarrow d \qquad c \Downarrow \mathit{True}}{b \vee c \Downarrow \mathit{True}}$

This semantics can be readily tested in Haskell, since it is lazy:

```
fix f = f (fix f)                              -- fixpoint combinator
omega = fix id                                 -- divergence
success = ()                                   -- successful termination

test1 = fst $ (success, omega)                 -- terminates
test2 = fst $ (success, omega)                 -- diverges

test3 = True  || omega                         -- terminates
test4 = omega || True                          -- diverges
test4 = False || omega                         -- diverges
```

1.3.3 Call-by-Value Operation Semantics for PCF

Definition (Value)**.** A value is a Boolean, or a natural number, or $\star$, or a pair of values or a closed term $\lambda x.\, t$.

$$\frac{p \Downarrow_{\text{cbv}} c_1 \qquad q \Downarrow_{\text{cbv}} c_2}{\langle p, q\rangle \Downarrow_{\text{cbv}} \langle c_1, c_2\rangle} \qquad \frac{p \Downarrow_{\text{cbv}} \langle c_1, c_2\rangle}{\mathsf{fst}\, p \Downarrow_{\text{cbv}} c_1} \qquad \frac{p \Downarrow_{\text{cbv}} \langle c_1, c_2\rangle}{\mathsf{snd}\, p \Downarrow_{\text{cbv}} c_2}$$

If we used the same rule for the Y-combinator, as for call-by-name, we would diverge:

$$Y f \longrightarrow f(Y f) \longrightarrow f(f(Y f)) \longrightarrow \cdots$$

(Evaluating the argument would use the same rule on and on). In order to prevent this, for the CBV semantics:

- we require C in Y_C to be of the form $A \to B$,
- the small-step rule for Y: $Y f \to f(\lambda x.(Y f)x)$, or, alternatively, as a big-step rule:

$$\frac{f \Downarrow_{\text{cbv}} \lambda x.\, g \qquad g[\lambda y.\,(Y f)y/x] \Downarrow_{\text{cbv}} c}{Y f \Downarrow_{\text{cbv}} c}$$

Example (Factorial)**.** Consider the program:

$$p := x\colon Nat \vdash (Y_{Nat \to Nat}(\underbrace{\lambda f.\lambda x.\text{if } x \leqslant 1 \text{ then } 1 \text{ else } x \cdot f(x-1)}_{g}))(x)$$

We show that: $(\lambda x.\, p)(n) \Downarrow n!$ (in CBV)

Proof. Induction over n.
Induction base (w $= 0$):

$$
\dfrac{\lambda x.\, p \Downarrow \lambda x.\, p \qquad 0 \Downarrow 0 \qquad \dfrac{\dfrac{g \Downarrow g \qquad \dfrac{g \Downarrow g \qquad \lambda x.(Yg)x \Downarrow \lambda x.(Yg)x \qquad \begin{array}{l} g = (\lambda x.\text{if } x \leqslant 1 \text{ then } 1 \\ \text{else } x \cdot (\lambda x.\, yg)x)(x-1) \Downarrow g \end{array}}{g(\lambda x.(Yg)x) \Downarrow g}}{Yg \Downarrow g} \qquad 0 \Downarrow 0 \qquad \dfrac{g \Downarrow g \qquad 0 \Downarrow 0 \qquad \dfrac{\dots}{g[0/x] \Downarrow 1}}{g0 \Downarrow 1}}{(Yg)0 \Downarrow 1}}{(\lambda x.\, p)0 \Downarrow 1}
$$

Induction step:
Insert lengthy proof of $(\lambda x.\, p)(n+1) \Downarrow (n+1)!$
Insert proof that this is indeed a well-typed term $\square$

1.4 Denotational Semantics of PCF

Operational semantics is non-compositional, is the sense that it does not yield a function $[\![-]\!]$ from terms to meanings, so that for every n-ary term construct op, $[\![op(t_1, \dots, t_n)]\!]$ could be calculated as a function of $[\![t_1]\!], \dots, [\![t_n]\!]$. In particular, operational semantics does not directly define meanings of functions, hence we cannot express $[\![ft]\!]$ via $[\![f]\!]$ and $[\![t]\!]$.

Definition (Exponential)**.** Recall that in set theory the exponential B^A (latter also written as $A \to B$) is the set of relations $P \subseteq A \times B$, which are

- functional: $\forall x.\, \forall y.\, \forall z.\, P(x,y) \wedge P(x,z) \implies y = z$, and
- total: $\forall x.\, \exists y.\, P(x,y)$.

Given A, B, then $A^B \subseteq A \times B$ can be formed and used as a domain for functions.

We could use A^B to give the denotational semantic of the PCF function type $A \to B$: $[\![A \to B]\!] = [\![B]\!]^{[\![A]\!]}$, but doesn't work because of the possibility of divergence.

Another candidate would be $[\![A \to B]\!] = ([\![B]\!] \uplus \bot)^{[\![A]\!]}$, but then B can again be a function space and we would have an unwanted distinction between divergence $\bot$ and everywhere diverging function $\lambda x.\, \bot$.

The right idea is to use *complete partial orders (cpos)*!

Definition (Partial Orders)**.** A partial order $(A, \sqsubseteq)$ is a relation satisfying the following axioms:

- $a \sqsubseteq a$;
- $a \sqsubseteq b \wedge b \sqsubseteq c \Rightarrow a \sqsubseteq c$;
- $a \sqsubseteq b \wedge b \sqsubseteq a \Rightarrow a = b$.

Definition (Complete Partial Orders)**.** A(n ω-)cpo is a partial order $(A, \sqsubseteq)$, such that for any infinite chain

$$a_1 \sqsubseteq a_2 \sqsubseteq \dots,$$

there is an a, such that

1. $\forall i.\, a_i \sqsubseteq a$;
2. $\forall i.\, a_i \sqsubseteq b \Rightarrow a \sqsubseteq b$.

We denote such a by $\bigsqcup_i a_i$. More, generally we write $\bigsqcup_{i\in I} a_i$ for any least upper bound (not necessarily of a chain) if $\forall i.\, a_i \sqsubseteq \bigsqcup_{i\in I} a_i$ and $\bigsqcup_{i\in I} a_i \sqsubseteq b$ once $\forall i.\, a_i \sqsubseteq b$.

Definition (Pointed Cpos)**.** A cpo $(A, \sqsubseteq)$ is pointed if it contains such an element $\bot$, that $\forall a \in A.\, \bot \sqsubseteq a$

Every set A is trivially a cpo $(A, \sqsubseteq)$ with $a \sqsubseteq b$ iff $a = b$.

Definition (Monotonicity, Continuity, Strictness)**.** A function $f\colon A \to B$ between partial orders is *monotone* if $a \sqsubseteq b \Rightarrow f(a) \sqsubseteq f(b)$; a monotone function $f\colon A \to B$ between cpos $(A, \sqsubseteq)$ and $(B, \sqsubseteq)$ is *(Scott-)continuous* if for any chain $a_1 \sqsubseteq a_2 \sqsubseteq \ldots$:

$$f\Big(\bigsqcup\nolimits_i a_i\Big) = \bigsqcup\nolimits_i f(a_i)$$

A function $f\colon A \to B$ is *strict* if $f(\bot) = \bot$. This extends to the multi-ary functions in the obvious way, e.g. if-then-else is strict in the first argument, but not in the second and the third.

Definition ((Pre-)Domain)**.** We agree to refer to cpos as *pre-domains*, and to pointed cpos as *domains.*

1.4.1 Constructions on Predomains

Product of Predomains $A \times B = \{(a, b) \mid a \in A, b \in B\}$

$$(a_1, b_1) \sqsubseteq (a_2, b_2) \quad \text{if} \quad a_1 \sqsubseteq a_2 \quad \text{and} \quad b_1 \sqsubseteq b_2$$

Properties:

- Continuity of pairing: $\bigsqcup_i (a_i, b_i) = \big(\bigsqcup_i a_i, \bigsqcup_j b_j\big)$;
- Continuity of projections: $\mathsf{fst}\colon A \times B \to A$ and $\mathsf{snd}\colon A \times B \to B$ are continuous, i.e.: $\mathsf{fst}\big(\bigsqcup_j a_j\big) = \bigsqcup_j \mathsf{fst}\, a_j$, $\mathsf{snd}\big(\bigsqcup_j a_j\big) = \bigsqcup_j \mathsf{snd}\, a_j$;
- Products of domains are again domains with $(\bot, \bot)$ as the least element.

Lifting Predomains and Functions The correspondence $A \mapsto A_\bot$ defines a *lifing* of A where $A_\bot = A \uplus \{\bot\} = \{(\star, a) \mid a \in A\} \cup \{(\bot, \star)\}$.

$$a \sqsubseteq b \quad \text{if} \quad a = \bot \quad \text{or} \quad a \in A, b \in A \text{ and } a \sqsubseteq b$$

Let for any $a \in A$: $\lfloor a \rfloor = (\star, a) \in A_\bot$.

Let B be a domain and let $f\colon A \to B$ be continuous. Then we define $f^\star\colon A_\bot \to B$ as follows:

$$f^\star(x) = \begin{cases} f(y) & \text{if } x = \lfloor y \rfloor \\ \bot & \text{if } x = \bot \end{cases}$$

The result $f^\star$ is the *lifting* of f.

Example (Flat Domains). : Given a set A, $A_\bot$ is called the *flat domain* over A, regarded as a trivially ordered set (i.e. $\sqsubseteq$ is $=$).

$Bool_\bot$:

$$
\begin{tikzcd}[column sep=small]
\mathit{True} & & \mathit{False} \\
& \bot \arrow[ul, dash] \arrow[ur, dash] &
\end{tikzcd}
$$

$Nat_\bot$:

$$
\begin{tikzcd}[column sep=small]
0 & 1 & 2 & \cdots \\
& & \bot \arrow[ull, dash] \arrow[ul, dash] \arrow[u, dash] &
\end{tikzcd}
$$

Non-example $1_\bot \times 1_\bot$:

$$
\begin{tikzcd}[column sep=small]
& (\star, \star) & \\
(\star, \bot) \arrow[ur, dash] & & (\bot, \star) \arrow[ul, dash] \\
& (\bot, \bot) \arrow[ul, dash] \arrow[ur, dash] &
\end{tikzcd}
$$

Notation. We use the point-full notation $(\mathsf{let}\, x = p \,\mathsf{in}\, q)$ alongside with the point-free one $(\lambda x.\, q)^\star(p)$ where $\lambda x.\, q \colon A \to B$ and $p \colon A_\bot$.

Properties:

- $\lfloor - \rfloor$ is continuous: $\lfloor \bigsqcup_i a_i \rfloor = \bigsqcup_i \lfloor a_i \rfloor$.
- Lifting is continuous: $\left(\bigsqcup_i f_i\right)^\star = \bigsqcup_i f_i^\star$ where continuous functions are compared *pointwise*, that is $f \sqsubseteq g$ if $f(x) \sqsubseteq g(x)$ for any x (see the definition of function spaces bellow).

For every $op \colon X \times Y \to Z$ with X, Y, Z being sets, we define the *strict extension*:

$$
\begin{aligned}
op_\bot &: X_\bot \times Y_\bot \to Z_\bot \\
op_\bot(p, q) &= \mathsf{let}\, x = p \,\mathsf{in}\, \mathsf{let}\, y = q \,\mathsf{in}\, \lfloor op(x, y) \rfloor
\end{aligned}
$$

Function Spaces Let $(A, \sqsubseteq)$ and $(B, \sqsubseteq)$ be two predomains. Then $(A \to B, \sqsubseteq)$ is the function space predomain, where

$$A \to B = \{f \colon A \to B \mid f \text{ is continuous}\}$$

and

$$f \sqsubseteq g \Leftrightarrow \forall x.\, f(x) \sqsubseteq g(x) \ \text{(pointwise)}$$

We define two operations:

$$\begin{aligned}
\mathsf{curry}\colon (A \times B \to C) &\to (A \to (B \to C)) \\
(\mathsf{curry}\, f)(x)(y) &= f(x,y)
\end{aligned}$$

$$\begin{aligned}
\mathsf{uncurry}\colon (A \to (B \to C)) &\to (A \times B \to C) \\
(\mathsf{uncurry}\, f)(x,y) &= f(x)(y)
\end{aligned}$$

from which we can derive

$$\mathsf{ev} = \mathsf{uncurry}((A \to B) \to (A \to B))\colon (A \to B) \times A \to B$$

Properties:

- curry and $\mathsf{uncurry}$ are continuous.
- If B is a domain then so is $A \to B$ with the bottom element being the completely undefined function $\lambda x.\, \bot$.

Theorem 1 (Kleene's Fixpoint Theorem)**.** Let f be a continuous function $f\colon D \to D$ over a domain D. Then

1. There is $\mu f \in D$—the *least fixpoint of* f, i.e.
   a) $f(\mu f) = \mu f$
   b) $\forall x \in D.\, f(x) = x \Rightarrow \mu f \sqsubseteq x$
2. $\mu f = \bigsqcup_i f^i(\bot)$, where $f^0(x) = \bot$, $f^{i+1}(x) = f(f^i(x))$
3. $\mu f \in D$ is moreover the *least pre-fixpoint of* f, i.e.
   a) $f(\mu f) \sqsubseteq \mu f$
   b) $\forall x \in D.\, f(x) \sqsubseteq x \Rightarrow \mu f \sqsubseteq x$

Proof. Let us first show that μf as defined in clause 2 is a fixpoint of f. Indeed, $f(\mu f) = f\Big(\bigsqcup_i f^i(\bot)\Big) = \Big(\bigsqcup_i f^{i+1}(\bot)\Big) = \mu f$. Hence is it also a prefixpoint. Let us show that it is the least one. Suppose that c is another prefixpoint, i.e. $f(c) \sqsubseteq c$. From $\bot \sqsubseteq c$, inductively, $f^i(\bot) \sqsubseteq f^i(c) = c$, hence $\mu f = \bigsqcup_i f^i(\bot) \sqsubseteq c$. Since μf is the least prefixpoint and a fixpoint, it is in particular the least prefixpoint. $\square$

Example.

$$\begin{aligned}
&f_0(x) = \bot(\forall x) \\
&f_1(0) = 1, f_1(x) = \bot(x > 0) \\
&f_2(0) = 1, f_2(1) = 1, f_2(x) = \bot(x > 1) \\
&f_3(0) = 1, f_3(1) = 1, f_3(2) = 2, f_3(x) = \bot(x > 2) \\
&f_4(0) = 1, f_4(1) = 1, f_4(2) = 2, f_4(3) = 6, f_4(x) = \bot(x > 2) \\
&\vdots
\end{aligned}$$

It's easy to see that every f_i is continuous.

It's also easy to prove that $f_i \sqsubseteq f_{i+1}$ for any i. Let

$$f = \bigsqcup_i f_i$$

By Kleene's fixpoint theorem we can argue that f captures the semantics of the factorial function $n \mapsto n!$. Note that

$$f_{i+1} = F(f_i) \qquad \forall (i \in \mathbb{N})$$

where

$$F(g)(x) = \begin{cases} 1 & \text{if } x = 1 \\ x \cdot g(x-1) & x > 1 \end{cases}$$

which is the definition of the factorial. By Kleene's theorem this definition is indeed correct:

$$f = \mu F = \bigsqcup_i F^i(\bot) = \bigsqcup_i f_i$$

Proposition. $\mu\colon (D \to D) \to D$ is continuous.

Definition (Cond)**.** Let $\mathsf{cond}\colon \mathit{Bool}_\bot \times X \times X \to X$:

$$\mathsf{cond}(b, x, y) = \begin{cases} x & \text{if } b = \lfloor \mathit{True} \rfloor \\ y & \text{if } b = \lfloor \mathit{False} \rfloor \\ \bot & \text{otherwise} \end{cases}$$

Proposition. cond is continuous.

1.4.2 CBN Denotational Semantics

We assign to every type A a domain $[\![A]\!]$ as follows:

- $[\![1]\!] = 1_\bot$;
- $[\![\mathit{Nat}]\!] = \mathit{Nat}_\bot$;
- $[\![\mathit{Bool}]\!] = \mathit{Bool}_\bot$;
- $[\![A \times B]\!] = [\![A]\!] \times [\![B]\!]$;
- $[\![A \to B]\!] = [\![A]\!] \to [\![B]\!]$.

Now, given a term in context $\Gamma \vdash t\colon A$ where $\Gamma = x_1\colon A_1, \ldots, x_n\colon A_n$ the semantics $[\![\Gamma \vdash t\colon A]\!]$ is a continuous function $[\![A_1]\!] \times \ldots \times [\![A_n]\!] \to [\![A]\!]$ recursively computed according to the following clauses where $[\![\cdots]\!]_\rho$ reads as $[\![\cdots]\!](\rho)$:

- $[\![\Gamma \vdash x_i\colon A_i]\!]_\rho = \mathsf{pr}_i(\rho)$;

- $[\![\Gamma \vdash n \colon Nat]\!]_\rho = \lfloor n \rfloor$;
- $[\![\Gamma \vdash b \colon Bool]\!]_\rho = \lfloor b \rfloor$;
- $[\![\Gamma \vdash f(t,s) \colon A]\!]_\rho = f_\perp([\![\Gamma \vdash t \colon B]\!]_\rho, [\![\Gamma \vdash s \colon C]\!]_\rho) \qquad (f \in \{\wedge, \to, +, -, \times, =\})$;
- $[\![\Gamma \vdash \textsf{if}\ b\ \textsf{then}\ s\ \textsf{else}\ t \colon A]\!]_\rho = cond([\![\Gamma \vdash b \colon Bool]\!]_\rho, [\![\Gamma \vdash s \colon A]\!]_\rho, [\![\Gamma \vdash t \colon A]\!]_\rho)$;
- $[\![\Gamma \vdash \langle t, s\rangle \colon A \times B]\!]_\rho = \langle [\![\Gamma \vdash t \colon A]\!]_\rho, [\![\Gamma \vdash s \colon B]\!]_\rho \rangle$;
- $[\![\Gamma \vdash \textsf{fst}\, t \colon A]\!]_\rho = \textsf{fst}[\![\Gamma \vdash t \colon A_1 \times A_2]\!]_\rho$;
- $[\![\Gamma \vdash \textsf{snd}\, t \colon B]\!]_\rho = \textsf{snd}[\![\Gamma \vdash t \colon A_1 \times A_2]\!]_\rho$;
- $[\![\Gamma \vdash \lambda x.\, t \colon A \to B]\!]_\rho = (\textsf{curry}\, [\![\Gamma, x \colon A \vdash t \colon B]\!])(\rho)$;
- $[\![\Gamma \vdash s\, t \colon B]\!]_\rho = \textsf{ev}([\![\Gamma \vdash s \colon A \to B]\!]_\rho, [\![\Gamma \vdash t \colon A]\!]_\rho)$;
- $[\![\Gamma \vdash Y_A\, f \colon A]\!]_\rho = \mu[\![\Gamma \vdash f \colon A \to A]\!]_\rho$.

Lemma (Substitution Lemma)**.** Given $\Gamma \vdash q \colon A$, $\Gamma, x \colon A \vdash p \colon B$ and $\rho \in [\![\Gamma]\!]$

$$[\![\Gamma \vdash p[q/x] \colon B]\!]_\rho = [\![\Gamma, x \colon A \vdash p \colon B]\!](\rho, [\![\Gamma \vdash q \colon A]\!]_\rho)$$

Proof. Induction over the structure of p. Let us consider the there last clauses in the semantics for p, which are the only non-trivial ones.

- $p = \lambda y.\, t$ with some $\Gamma, y \colon C \vdash t \colon D$ and then $B = C \to D$. It follows by assumption that $x \neq y$. Then, by induction,

$$\begin{aligned}
[\![\Gamma \vdash p[q/x] \colon B]\!]_\rho &= [\![\Gamma \vdash \lambda y.\, t[q/x] \colon B]\!]_\rho \\
&= (\textsf{curry}[\![\Gamma, y \colon C \vdash t[q/x] \colon D]\!])(\rho) \\
&= (\textsf{curry}([\![\Gamma, y \colon C, x \colon A \vdash t \colon D]\!] \circ (\textsf{id}, [\![\Gamma, y \colon C \vdash q \colon A]\!])))(\rho) \\
&= (\textsf{curry}[\![\Gamma, x \colon A, y \colon C \vdash t \colon D]\!])(\rho, [\![\Gamma \vdash q \colon A]\!]_\rho) \\
&= [\![\Gamma, x \colon A \vdash \lambda y.\, t \colon B]\!](\rho, [\![\Gamma \vdash q \colon A]\!]_\rho) \\
&= [\![\Gamma, x \colon A \vdash p \colon B]\!](\rho, [\![\Gamma \vdash q \colon A]\!]_\rho).
\end{aligned}$$

- $p = s\, t$ with some $\Gamma, x \colon A \vdash t \colon C$ and $\Gamma, x \colon A \vdash s \colon C \to B$. Then, by induction,

$$\begin{aligned}
[\![\Gamma \vdash p[q/x] \colon B]\!]_\rho &= [\![\Gamma \vdash (s[q/x])\, (t[q/x]) \colon B]\!]_\rho \\
&= [\![\Gamma \vdash s[q/x] \colon C \to B]\!]_\rho([\![\Gamma \vdash t[q/x] \colon C]\!]_\rho) \\
&= ([\![\Gamma, x \colon A \vdash s \colon C \to B]\!](\rho, [\![\Gamma \vdash q \colon A]\!]_\rho)) \\
&\quad ([\![\Gamma, x \colon A \vdash t \colon C]\!](\rho, [\![\Gamma \vdash q \colon A]\!]_\rho)) \\
&= [\![\Gamma, x \colon A \vdash s\, t \colon B]\!](\rho, [\![\Gamma \vdash q \colon A]\!]_\rho) \\
&= [\![\Gamma, x \colon A \vdash p \colon B]\!](\rho, [\![\Gamma \vdash q \colon A]\!]_\rho).
\end{aligned}$$

- $p = Y_B\, f$ with some $\Gamma, x\colon A \vdash f\colon B \to B$. Analogously to the previous clauses:

$$\begin{aligned}
[\![\Gamma \vdash p[q/x]\colon B]\!]_\rho &= [\![\Gamma \vdash (Y_B\, f)[q/x]\colon B]\!]_\rho \\
&= [\![\Gamma \vdash Y_B\, f[q/x]\colon B]\!]_\rho \\
&= \mu[\![\Gamma \vdash f[q/x]\colon B \to B]\!]_\rho \\
&= \mu([\![\Gamma, x\colon A \vdash f\colon B \to B]\!](\rho, [\![\Gamma \vdash q\colon A]\!]_\rho)) \\
&= [\![\Gamma, x\colon A \vdash \mu f\colon B]\!](\rho, [\![\Gamma \vdash q\colon A]\!]_\rho) \\
&= [\![\Gamma, x\colon A \vdash p\colon B]\!](\rho, [\![\Gamma \vdash q\colon A]\!]_\rho).
\end{aligned}$$

□

Definition (Soundness)**.** A denotational semantics is *sound* if

$$p \Downarrow c \Rightarrow [\![p]\!] = [\![c]\!]$$

Definition (Adequacy)**.** A denotational semantics is *adequate*, if

$$[\![p]\!] = [\![c]\!] \Rightarrow p \Downarrow c \quad \text{if the type of } p \text{ is either 1 or } \mathit{Bool} \text{ or } \mathit{Nat}$$

Proposition. The presented call-by-name denotational semantics is sound and adequate with respect to $\Downarrow_{\text{cbn}}$.

1.4.3 CBV Denotational Semantics

We we assign to every type A a predomain $[\![A]\!]$ as follows:

- $[\![1]\!] = 1$;
- $[\![\mathit{Nat}]\!] = \mathit{Nat}$;
- $[\![\mathit{Bool}]\!] = \mathit{Bool}$;
- $[\![A \times B]\!] = [\![A]\!] \times [\![B]\!]$;
- $[\![A \to B]\!] = [\![A]\!] \to [\![B]\!]_\bot$.

Now, the semantics of a term in context $\Gamma \vdash t\colon A$ with $\Gamma = (x_1\colon A_1, \ldots, x_n\colon A_n)$ is a continuous function $[\![A_1]\!] \times \ldots \times [\![A_n]\!] \to [\![A]\!]_\bot$ defined by structural induction as follows.

- $[\![\Gamma \vdash x_i\colon A_i]\!]_\rho = \lfloor \mathsf{pr}_i(\rho) \rfloor$;
- $[\![\Gamma \vdash n\colon \mathit{Nat}]\!]_\rho = \lfloor n \rfloor$;
- $[\![\Gamma \vdash b\colon \mathit{Bool}]\!]_\rho = \lfloor b \rfloor$;
- $[\![\Gamma \vdash f(t,s)\colon A]\!]_\rho = f_\bot([\![\Gamma \vdash t\colon B]\!]_\rho, [\![\Gamma \vdash s\colon C]\!]_\rho) \qquad (f \in \{\wedge, \to, +, -, \times, =\})$;
- $[\![\Gamma \vdash \mathsf{if}\ b\ \mathsf{then}\ s\ \mathsf{else}\ t\colon A]\!]_\rho = \mathit{cond}([\![\Gamma \vdash b\colon \mathit{Bool}]\!]_\rho, [\![\Gamma \vdash s\colon A]\!]_\rho, [\![\Gamma \vdash t\colon A]\!]_\rho)$;
- $[\![\Gamma \vdash \langle t, s\rangle\colon A \times B]\!]_\rho = \mathsf{let}\ x = [\![\Gamma \vdash t\colon A]\!]_\rho\ \mathsf{in}\ \mathsf{let}\ y = [\![\Gamma \vdash s\colon B]\!]_\rho\ \mathsf{in} \lfloor \langle x, y\rangle \rfloor$;
- $[\![\Gamma \vdash \mathsf{fst}\, t\colon A]\!]_\rho = \mathsf{let}\ v = [\![\Gamma \vdash t\colon A \times B]\!]_\rho\ \mathsf{in} \lfloor \mathsf{fst}\, v \rfloor$;

- $[\![\Gamma \vdash \mathsf{snd}\, t \colon B]\!]_\rho = \mathsf{let}\, v = [\![\Gamma \vdash t \colon A \times B]\!]_\rho\, \mathsf{in} \lfloor \mathsf{snd}\, v \rfloor$;
- $[\![\Gamma \vdash \lambda x.\, t \colon A \to B]\!]_\rho = \lfloor (\mathsf{curry}\, [\![\Gamma, x \colon A \vdash t \colon B]\!])(\rho) \rfloor$;
- $[\![\Gamma \vdash s\, t \colon B]\!]_\rho = \mathsf{let}\, v = [\![\Gamma \vdash t \colon A]\!]_\rho\, \mathsf{in}\, \mathsf{let}\, f = [\![\Gamma \vdash s \colon A \to B]\!]_\rho\, \mathsf{in}\, \mathsf{ev}(f, v)$;
- $[\![\Gamma \vdash Y_{A \to B}\, f \colon A \to B]\!]_\rho = \mu g$ where
  - $g(p \colon ([\![A]\!] \to [\![B]\!]_\perp)_\perp) = \mathsf{let}\, h = [\![\Gamma \vdash f \colon (A \to B) \to (A \to B)]\!]_\rho\, \mathsf{in}\, h(u(p))$,
  - $u(p \colon ([\![A]\!] \to [\![B]\!]_\perp)_\perp)(x \colon [\![A]\!]) = \mathsf{let}\, h = p\, \mathsf{in}\, h(x)$.

The analogue of the substitution lemma is as follows.

Lemma (Substitution Lemma)**.** Given $\Gamma \vdash q \colon A$, $\Gamma, x \colon A \vdash p \colon B$ and $\rho \in [\![\Gamma]\!]$,

$$[\![\Gamma \vdash p[q/x] \colon B]\!]_\rho = \mathsf{let}\, v = [\![\Gamma \vdash q \colon A]\!]_\rho\, \mathsf{in} [\![\Gamma, x \colon A \vdash p \colon B]\!](\rho, v)$$

provided that q is of the form $\lambda z.\, r$.

In contrast to the call-by-name case, the assumption that $q = \lambda z.\, r$ is essential. For example, if q diverges, but p does not depend on x, we would have $[\![\Gamma \vdash p \colon B]\!]$ on the left-hand side and $\perp$ on the right-hand side.

Proof. The proof is by structural induction over p. Again, only the last three clauses in the definition of semantics of p are sophisticated. Still the other ones require some properties of the let-construct (commutativity and copyability).

Assume that $\Gamma, z \colon E \vdash r \colon F$, i.e. $A = E \to F$.

- $p = \lambda y.\, t$ with some $\Gamma, y \colon C, x \colon A \vdash t \colon D$ and then $B = C \to D$. It follows by assumption that $x \neq y$. Let us fix $c \in [\![C]\!]$, $\rho \in [\![\Gamma]\!]$ and let $s = \mathsf{let}\, v = [\![\Gamma \vdash q \colon A]\!]_\rho\, \mathsf{in} [\![\Gamma, x \colon A \vdash p \colon B]\!](\rho, v)$. It is easy to check that $s = \lfloor g \rfloor$ for some g. Then

$$\begin{aligned}
&[\![\Gamma, y \colon C \vdash t[q/x] \colon D]\!](\rho, c)\\
&\qquad = \mathsf{let}\, v = [\![\Gamma, y \colon C \vdash q \colon A]\!](\rho, c)\, \mathsf{in} [\![\Gamma, y \colon C, x \colon A \vdash t \colon D]\!](\rho, c, v)\\
&\qquad = \mathsf{let}\, v = [\![\Gamma \vdash q \colon A]\!]_\rho\, \mathsf{in} [\![\Gamma, x \colon A, y \colon C \vdash t \colon D]\!](\rho, v, c)\\
&\qquad = \mathsf{let}\, v = [\![\Gamma \vdash q \colon A]\!]_\rho\, \mathsf{in}\, \mathsf{let}\, f = [\![\Gamma, x \colon A \vdash p \colon B]\!](\rho, v)\, \mathsf{in}\, f(c)\\
&\qquad = \mathsf{let}\, f = (\mathsf{let}\, v = [\![\Gamma \vdash q \colon A]\!]_\rho\, \mathsf{in} [\![\Gamma, x \colon A \vdash p \colon B]\!](\rho, v))\, \mathsf{in}\, f(c)\\
&\qquad = \mathsf{let}\, f = \lfloor g \rfloor\, \mathsf{in}\, f(c)\\
&\qquad = g(c)
\end{aligned}$$

using the fact that q does not depend on y. Now

$$\begin{aligned}
&[\![\Gamma \vdash p[q/x] \colon B]\!]_\rho\\
&\qquad = [\![\Gamma \vdash \lambda y.\, t[q/x] \colon B]\!]_\rho\\
&\qquad = \lfloor (\mathsf{curry} [\![\Gamma, y \colon C \vdash t[q/x] \colon D]\!])(\rho) \rfloor\\
&\qquad = \lfloor g \rfloor\\
&\qquad = s.
\end{aligned}$$

- $p = s\,t$ with some $\Gamma, x\colon A \vdash t\colon C$ and $\Gamma, x\colon A \vdash s\colon C \to B$. Then, by induction,

$$\begin{aligned}
[\![\Gamma \vdash p[q/x]\colon B]\!]_\rho &= [\![\Gamma \vdash (s[q/x])\,(t[q/x])\colon B]\!]_\rho \\
&= \mathsf{let}\, v = [\![\Gamma \vdash t[q/x]\colon C]\!]_\rho \\
&\qquad \mathsf{in}\,\mathsf{let}\, f = [\![\Gamma \vdash s[q/x]\colon C \to B]\!]_\rho\, \mathsf{in}\, f(v) \\
&= \mathsf{let}\, w = [\![\Gamma \vdash q\colon A]\!]_\rho\, \mathsf{in}\,\mathsf{let}\, v = [\![\Gamma, x\colon A \vdash t\colon C]\!](\rho, w) \\
&\qquad \mathsf{in}\,\mathsf{let}\, f = [\![\Gamma, x\colon A \vdash s\colon C \to B]\!](\rho, w)\, \mathsf{in}\, f(v) \\
&= \mathsf{let}\, w = [\![\Gamma \vdash q\colon A]\!]_\rho\, \mathsf{in} [\![\Gamma, x\colon A \vdash s\,t\colon B]\!](\rho, w).
\end{aligned}$$

- $p = Y_B\, f$ with some $\Gamma, x\colon A \vdash f\colon (C \to D) \to (C \to D)$, hence $B = (C \to D)$.

Note that for a suitable w, $[\![\Gamma \vdash q\colon A]\!]_\rho = \lfloor w \rfloor$. Then

$$\begin{aligned}
[\![\Gamma \vdash p[q/x]\colon B]\!]_\rho &= [\![\Gamma \vdash (Y_B\, f)[q/x]\colon B]\!]_\rho \\
&= [\![\Gamma \vdash Y_B\, f[q/x]\colon B]\!]_\rho \\
&= \mu(g) \\
&= [\![\Gamma, x\colon A \vdash Y_B f\colon B]\!](\rho, w) \\
&= \mathsf{let}\, v = [\![\Gamma \vdash q\colon A]\!]_\rho\, \mathsf{in} [\![\Gamma, x\colon A \vdash p\colon B]\!](\rho, v).
\end{aligned}$$

where $g(p) = \mathsf{let}\, h = [\![\Gamma, x\colon A \vdash f\colon B \to B]\!](\rho, w)\, \mathsf{in}\, h(u(p))$ and $u(p)(x) = \mathsf{let}\, h = p\, \mathsf{in}\, h(x)$ □

Proposition. The CBV semantics of PCF is sound and adequate.

Proposition (let-unit-1)**.** $\mathsf{let}\, x = \lfloor t \rfloor\, \mathsf{in}\, p = p[t/x]$.

Proof.

$$\mathsf{let}\, x = \lfloor t \rfloor\, \mathsf{in}\, p = (\lambda x.\, p)^\star \lfloor t \rfloor \quad = \quad \begin{cases} (\lambda x.\, p)(s) & \text{if} \quad \lfloor t \rfloor = \lfloor s \rfloor \\ \bot & \text{otherwise} \end{cases} \quad = \quad \begin{cases} (\lambda x.\, p)t \\ p[t/x] \end{cases}$$

□

Proposition (let-unit-2)**.** $\mathsf{let}\, x = p\, \mathsf{in} \lfloor x \rfloor = p$.

Proof. $\mathsf{let}\, x = p\, \mathsf{in} \lfloor x \rfloor = (\lambda x.\, \lfloor x \rfloor)^\star(p) = (\lambda x.\, x)(p) = p$. □

Proposition (let-assoc)**.**

$$\mathsf{let}\, x = p\, \mathsf{in} (\mathsf{let}\, y = q\, \mathsf{in}\, r) = \mathsf{let}\, y = (\mathsf{let}\, x = p\, \mathsf{in}\, q)\, \mathsf{in}\, r.$$

where $x \notin \mathrm{Free}(r)$.

Alternatively, the three laws for the let-operator can be presented in the pointfree form as follows:

$$f^\star \eta = \eta \qquad\qquad \eta^\star = \mathsf{id} \qquad\qquad f^\star g^\star = (f^\star g)^\star$$

where $\eta\colon A \to A_\perp$ sends x to $\lfloor x \rfloor$. These are known as *monad laws*, and they identify the map $A \mapsto A_\perp$ as a monad whose *unit* is $\lfloor - \rfloor$ and whose *Kleisli lifting* is the operation $(-)^\star$.

Thus, a monad can be understood as a certain type constructor that transforms *values* to *computations* and induces a notion of generalized function, carrying a certain *(side-)effect* in contrast to "normal functions". The side-effect of the lifting monad is *divergence*. Further side-effects that can be abstracted in monads include

- abortion,
- non-determinism,
- store,
- input/output,

and in fact many others. In order to make these considerations rigorous, we proceed with the basic concepts of category theory. As we will see, monads is a genuinely categorical concept.

2 Categories and Monads

Let us consider the *do-notation*, as a generalization of our previous *let-notation*. The idea is to capture the most abstract properties of computation, e.g. the let-notation also satisfies the following *commutativity* property:

$$\mathsf{let}\, x = p \,\mathsf{in}\, \mathsf{let}\, y = q \,\mathsf{in}\, \lfloor \langle x, y\rangle \rfloor = \mathsf{let}\, y = q \,\mathsf{in}\, \mathsf{let}\, x = p \,\mathsf{in}\, \lfloor \langle x, y\rangle \rfloor,$$

but this is not abstract enough: if p writes to a store and q reads from that store the order in which p and q are executed obviously matters.

Essentially we introduce two term constructs:

$$\mathsf{do}\, x \leftarrow \underbrace{p}_{TA};\ \underbrace{f}_{A\to TB}(x) \qquad\qquad \mathsf{ret}\colon A \to TA$$

In conjunction with other (obvious) term constructs this forms what is known as (first-order) *computational metalanguage* whose syntax is *Haskell's do-notation.*

2.1 Introducing Monads

Definition (Category)**.** A Category $\mathcal{C}$ consists of a collection of objects $\mathsf{Ob}(\mathcal{C})$ and a collection of morphisms $\mathsf{Hom}_{\mathcal{C}}(A, B)$ for any $A, B \in \mathsf{Ob}(\mathcal{C})$, such that the following properties hold:

- for every $A \in \mathsf{Ob}(\mathcal{C})$ there is an *identity morphism* $\mathsf{id}_A \in \mathsf{Hom}_{\mathcal{C}}(A, A)$;
- for any $f \in \mathsf{Hom}_{\mathcal{C}}(B, C)$ and $g \in \mathsf{Hom}_{\mathcal{C}}(A, B)$ we can form a *composition* $f \circ g \in \mathsf{Hom}_{\mathcal{C}}(A, C)$;
- $\mathsf{id} \circ f = f$, $f \circ \mathsf{id} = f$, $(f \circ g) \circ h = f \circ (g \circ h)$.

We also write $f\colon A \to B$ instead of $f \in \mathsf{Hom}_{\mathcal{C}}(A, B) = \mathsf{Hom}(A, B)$.

A "collection" in the definition of a category is in fact a *"class"*, i.e. something generally larger than a set, e.g. the "set of all sets" does not make sense, but "all sets" form a class. Categories in which any $\mathsf{Hom}(A, B)$ is a set are called *locally small* and the categories in which $\mathsf{Ob}(\mathcal{C})$ is a set are called *small.* Most of our examples of categories are locally small but not small.

Example. Examples of categories:

- Sets: $\mathsf{Ob}(\mathrm{Sets})$ = "all sets" and $\mathsf{Hom}(A, B)$ = "functions from A to B".
- Cpo: $\mathsf{Ob}(Cpo)$ = "all cpos" and $\mathsf{Hom}(A, B)$ = "continuous functions from A to B".

- Rel: $\mathsf{Ob}(Rel)$ = "all sets" and $\mathsf{Hom}(A, B)$ = "relations $R \subseteq A \times B$" with

$$\begin{aligned} \mathsf{id}_A &= \{(x, x) \mid x \in A\} \\ R \circ S &= \{(x, z) \in A \times C \mid \exists y \in B.\, (x, y) \in R, (y, z) \in S\} \end{aligned}$$

- PFun: $\mathsf{Ob}(PFun)$ = "all sets" and $\mathsf{Hom}(A, B)$ = "partial functions from A to B".

Definition (Commutative Diagrams)**.** We consider diagrams whose nodes are labeled with objects and whose edges are oriented and labelled with morphisms. A diagram *commutes* if all paths with the same start and endpoint produce equal morphisms (the morphism are formed by composing the labels along paths).

For example, the axioms for identity can be stated as follows:

$$\begin{tikzcd} A \arrow[r, "\mathsf{id}"] \arrow[dr, bend right, "f"'] & A \arrow[d, "f"] & A \arrow[l, "\mathsf{id}"'] \arrow[dl, bend left, "f"] \\ & B & \end{tikzcd}$$

Curiously, we cannot express associativity of composition in this way, because it is already baked in to the diagrammatic language.

In category theory, it is customary to prove equations between morphisms $f = g$ *"by diagram chasing"*, that is, by producing a commutative diagram, from which a chain of equations $f = f' = f'' = \ldots = g' = g$ can be read out. Importantly, not every commutative diagram produces a proof like this. For example, the diagram

$$\begin{tikzcd} \bullet \arrow[rr, "a"] \arrow[dd, "c"'] \arrow[dr] & & \bullet \arrow[dd, "b"] \\ & \bullet \arrow[ur] \arrow[dl] & \\ \bullet \arrow[rr, "d"] & & \bullet \arrow[ul] \end{tikzcd}$$

does not prove the equation $ba = dc$ even though all the triangles commute.

2.1.1 Products and Coproducts

Definition (Products)**.** A product of objects A, B in a category $\mathcal{C}$ is a triple $(A \times B \in \mathsf{Ob}(\mathcal{C}), \mathsf{fst}\colon A \times B \to A, \mathsf{snd}\colon A \times B \to B)$, such that for any $C \in \mathsf{Ob}(\mathcal{C})$ with $f\colon C \to A, g\colon C \to B$, there is a unique (!) morphism $\langle f, g\rangle\colon C \to A \times B$, such that the following diagram commutes:

$$\begin{tikzcd} & C \arrow[dl, bend right, "f"'] \arrow[d, "\langle f,s\rangle"'] \arrow[dr, bend left, "g"] & \\ A & A \times B \arrow[l, "\mathsf{fst}"] \arrow[r, "\mathsf{snd}"'] & B \end{tikzcd}$$

As a text: $\mathsf{fst} \circ \langle f, g\rangle = f$, $\mathsf{snd} \circ \langle f, g\rangle = g$. The morphisms fst and snd are called *(left and right) projections* and the operation $f, g \mapsto \langle f, g\rangle$ is called *pairing.*

Example.

- In Sets, products are Cartesian products.
- In Cpo, products are products of Cpos.

Definition (Terminal Object)**.** A terminal object is an object $1 \in \mathsf{Ob}(\mathcal{C})$, such that for any $A \in \mathsf{Ob}(\mathcal{C})$, there is a unique morphism: $!_A : A \to 1$

Definition (Cartesian Category)**.** A *Cartesian category* is a category with a terminal object and products.

Equivalently, a Cartesian category is the one which has all finite products: products of a nonempty finite number of components are obviously induced by binary products, the product of the empty family of components is the terminal object.

Examples: Sets and functions, Cpos and continuous functions, ...

Definition (Isomorphism)**.** An *isomorphism* between objects A and B in a category $\mathcal{C}$ is given by a pair of morphisms: $f : A \to B$, $g : B \to A$, such that the following diagram commutes:

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[dr, "\mathsf{id}_A"'] & B \arrow[d, "g"] \arrow[dr, "\mathsf{id}_B"] & \\ & A \arrow[r, "f"'] & B \end{tikzcd}$$

Example. In Sets, an isomorphism is a bijection.

Here is a translation table, between the different languages of set theory, category theory and Haskell.

Set	Categories	Haskell
function	morphism	program
set	object	type
singleton set	terminal object	unit type
Cartesian product	(Cartesian) product	product type
element	morphism $1 \to X$	—
predicate	—	—
bijection	isomorphism	—

Theorem 2. Let $A, B, C \in \mathsf{Ob}(\mathcal{C})$. A triple $(C, \mathsf{fst} : C \to A, \mathsf{snd} : C \to B)$, is a product of A and B if there is an operation

$$\frac{f : D \to A \qquad g : D \to B}{\langle f, g \rangle : D \to C}$$

such that

$$\mathsf{fst} \circ \langle f, g \rangle = f, \quad \mathsf{snd} \circ \langle f, g \rangle = g, \quad \langle \mathsf{fst}, \mathsf{snd} \rangle = \mathsf{id}, \quad \langle f, g \rangle \circ h = \langle f \circ h, g \circ h \rangle.$$

Proof. The proof consist of the soundness ($\Rightarrow$) and completeness ($\Leftarrow$) directions.

($\Rightarrow$) We need to show the claimed identities. The first two are obvious by definition. The other two are by diagram chasing:

$$\begin{tikzcd} & A\times B \arrow[ld, bend right, "\mathsf{fst}"'] \arrow[d, dashed, "\langle\mathsf{fst},\mathsf{snd}\rangle"] \arrow[rd, bend left, "\mathsf{snd}"] & \\ A & A\times B \arrow[l, "\mathsf{fst}"] \arrow[r, "\mathsf{snd}"'] & B \end{tikzcd} \qquad \begin{tikzcd} & D \arrow[ddl, bend right, "f\circ h"'] \arrow[d, "h"] \arrow[ddr, bend left, "g\circ h"] & \\ & C \arrow[dl, bend right, "f"'] \arrow[d, "\langle f,g\rangle"] \arrow[dr, bend left, "g"] & \\ A & A\times B \arrow[l, "\mathsf{fst}"] \arrow[r, "\mathsf{snd}"'] & B \end{tikzcd}$$

The first identity holds because in the left diagram replacing $\langle\mathsf{fst},\mathsf{snd}\rangle$ with id would produce a diagram, which still commutes, but $\langle\mathsf{fst},\mathsf{snd}\rangle$ is unique, hence $\langle\mathsf{fst},\mathsf{snd}\rangle = \mathsf{id}$.

The second identity holds analogously because by the second diagram $\langle f, g\rangle \circ h$ satisfies the characteristic property of $\langle f \circ h, g \circ h\rangle$.

($\Leftarrow$) Suppose, conversely, the identities hold and for some $h \colon D \to C$ the diagram:

$$\begin{tikzcd} & D \arrow[ld, bend right, "f"'] \arrow[d, "h"] \arrow[rd, bend left, "g"] & \\ A & C \arrow[l, "\mathsf{fst}"] \arrow[r, "\mathsf{snd}"'] & B \end{tikzcd}$$

commutes. Then

$$h = \mathsf{id} \circ h = \langle\mathsf{fst},\mathsf{snd}\rangle \circ h = \langle\mathsf{fst} \circ h, \mathsf{snd} \circ h\rangle = \langle f, g\rangle. \qquad \square$$

Products are defined *not uniquely*, but only *uniquely up to (a unique) isomorphism.* Let e.g. $(A \times A, \mathsf{fst}, \mathsf{snd})$ be a product of A, A. Then $(A\times A, \mathsf{snd}, \mathsf{fst})$ is also a product of A, A:

$$\mathsf{swap}_A \colon A \times A \xrightarrow{\langle\mathsf{snd},\mathsf{fst}\rangle} A \times A \qquad \begin{tikzcd} A\times A \arrow[r, bend left, "\mathsf{swap}"] & A\times A \arrow[l, bend left, "\mathsf{swap}"] \end{tikzcd}$$

The pair $(\mathsf{swap}_A, \mathsf{swap}_{A,A})$ is an isomorphism of $A \times A$ and $A \times A$:

$$\begin{aligned} \mathsf{swap} \circ \mathsf{swap} &= \langle\mathsf{snd},\mathsf{fst}\rangle \circ \langle\mathsf{snd},\mathsf{fst}\rangle \\ &= \langle\mathsf{snd} \circ \langle\mathsf{snd},\mathsf{fst}\rangle, \mathsf{fst} \circ \langle\mathsf{snd},\mathsf{fst}\rangle\rangle \\ &= \langle\mathsf{fst},\mathsf{snd}\rangle = \mathsf{id}. \end{aligned}$$

Theorem 3. Products (if they exists) are unique up to isomorphism.

Proof. Let $(A \times B, \mathsf{fst}, \mathsf{snd})$ be a product of A, B and let $(A\Box B, \mathsf{fst}', \mathsf{snd}')$ be another product. Then the following diagram commutes:

$$\begin{tikzcd} & A\times B \arrow[ddl, bend right, "\mathsf{fst}"'] \arrow[d, "\langle\mathsf{fst},\mathsf{snd}\rangle"'] \arrow[ddr, bend left, "\mathsf{snd}"] & \\ & A\Box B \arrow[dl, bend right, "\mathsf{fst}'"'] \arrow[d, "\langle\mathsf{fst}',\mathsf{snd}'\rangle"] \arrow[dr, bend left, "\mathsf{snd}'"] & \\ A & A\times B \arrow[l, "\mathsf{fst}"] \arrow[r, "\mathsf{snd}"'] & B \end{tikzcd}$$

Hence, $\overbrace{\langle \mathsf{fst}, \mathsf{snd}\rangle}^{f} \circ \overbrace{\langle \mathsf{fst}' \mathsf{snd}'\rangle}^{g} = \mathsf{id}$ (because both morphisms satisfy the same characteristic property). Because of symmetry, also $g \circ f = \mathsf{id}$. Hence (f, g) is an isomorphism between $A \times B$ and $A \Box B$. $\square$

Definition (Coproducts)**.** An object $A + B$ together with morphisms $\mathsf{inl}\colon A \to A + B$ and $\mathsf{inr}\colon B \to A + B$ called left and right *injections* is a *coproduct* of A and B if for any $f\colon A \to C, g\colon B \to C$, there is a unique morphism $[f, g]\colon A + B \to C$, such that the following diagram commutes:

$$\begin{tikzcd} & C & \\ A \arrow[ur, bend left, "f"] \arrow[r, "\mathsf{inl}"] & A + B \arrow[u, "{[f,g]}"] & B \arrow[l, "\mathsf{inr}"'] \arrow[ul, bend right, "g"'] \end{tikzcd}$$

Intuitively, $[f, g]$ is defined by case distinction: if we are on the left of $A + B$ then we apply f; if we are on the right of $A + B$ then we apply g.

Example. In Sets $A + B$ is the disjoint union of A and B.

Dually to products we have a complete axiomatization for coproducts:

1. $[f, g] \circ \mathsf{inl} = f$;
2. $[f, g] \circ \mathsf{inr} = g$;
3. $[\mathsf{inl}, \mathsf{inr}] = \mathsf{id}$;
4. $h \circ [f, g] = [h \circ f, h \circ g]$.

2.1.2 Functors and Monads

Definition (Functor)**.** A *(covariant) functor* between categories $\mathcal{C}$ and $\mathcal{D}$ is a correspondence sending any $A \in \mathsf{Ob}(\mathcal{C})$ to $FA \in \mathsf{Ob}(\mathcal{D})$ and any $f \in \mathsf{Hom}_{\mathcal{C}}(A, B)$ to $Ff \in \mathsf{Hom}_{\mathcal{D}}(FA, FB)$ in such a way that:

$$F(\mathsf{id}_A) = \mathsf{id}_{FA}, \qquad\qquad F(f \circ g) = (Ff) \circ (Fg).$$

Example (Forgetful Functor)**.** Forgetful functor is an informal concept: this is a functor that "forgets" some information about the category. One example is

$$\begin{aligned} &G\colon \mathrm{Cpo} \to \mathrm{Set} \\ &G(A, \sqsubseteq) = A \\ &G(f) = f \end{aligned}$$

G is a typical name for forgetful functors (to remember: for<u>G</u>etful).

Example (Endofunctor)**.** An *endofunctor* is a functor from a category into itself. E.g.,

$$\begin{aligned}
&F\colon \mathrm{Set} \to \mathrm{Set}\\
&FX = X + E\\
&(Ff)(\mathsf{inl}\,x) = \mathsf{inl}(fx)\\
&(Ff)(\mathsf{inr}\,e) = \mathsf{inr}(e)
\end{aligned}$$

Example (Finite Lists)**.** Another endofunctor over *Set*:

$$\begin{aligned}
&F\colon \mathrm{Set} \to \mathrm{Set}\\
&FX = [X] \quad \text{(finite lists over } X)\\
&(Ff)[x_1, \ldots, x_n] = [fx_1, \ldots, fx_n]
\end{aligned}$$

Definition (Monad/Kleisli Triple)**.** A *Monad* in a category $\mathcal{C}$ is given by a triple $(T, \eta, \_^\star)$ (*Kleisli triple*) where

- $T\colon\ \mathsf{Ob}(\mathcal{C}) \to \mathsf{Ob}(\mathcal{C})$,
- η is a family $(\eta_X\colon X \to TX)_{X \in \mathsf{Ob}(\mathcal{C})}$ (*unit*),
- for any $f\colon A \to TB$, $f^\star\colon TA \to TB$ (*(Kleisli) lifting*)

and the following laws are satisfied:

$$\eta^\star = \mathsf{id}, \qquad\qquad f^\star\eta = f, \qquad\qquad (f^\star g)^\star = f^\star g^\star.$$

Example (Exception monad)**.** $TX = X + E$ is a monad with:

$$\eta_X(a) = \mathsf{inl}\,a \qquad\qquad f^\star(\mathsf{inl}\,a) = fa \qquad\qquad f^\star(\mathsf{inr}\,e) = \mathsf{inr}\,e$$

This works in any category $\mathcal{C}$ with coproducts, $TX = X + E$ extends to a monad under the following definitions:

$$\begin{aligned}
&\eta_X = \mathsf{inl}\colon X \to X + E\\
&f^\star = [f, \mathsf{inr}]\colon X + E \to Y + E \text{ where } f\colon X \to Y + E
\end{aligned}$$

Intuitively, f is a function, which may *raise* an exception, and $f^\star$ completes the definition of f by the clause: "if an exception has already been raised before, pass it as the result".

It is easy to check that T from a monad $(T, \eta, -^\star)$ is a functor. We call it the *functorial part* of the monad.

Definition (Kleisli Category)**.** Given a monad T over a category $\mathcal{C}$, the *Kleisli category* $\mathcal{C}_T$ of T is defined as follows:

- $\mathsf{Ob}(\mathcal{C}_T) = \mathsf{Ob}(\mathcal{C})$;
- $\mathsf{Hom}_{\mathcal{C}_T}(A, B) = \mathsf{Hom}_{\mathcal{C}}(A, TB)$;

- identity morphisms in $\mathcal{C}_T$ are $\eta_X \in \mathsf{Hom}_{\mathcal{C}_T}(X, X) = \mathsf{Hom}_{\mathcal{C}}(X, TX)$;
- composition of $f\colon A \to TB$ and $g\colon B \to TC$ is *Kleisli composition*: $g^\star f\colon A \to TC$.

Theorem 4. $\mathcal{C}_T$ is a category:

1. $\eta^\star f = \mathsf{id} \circ f = f$
2. $f^\star \eta = f$
3. $f^\star(g^\star h) = (f^\star g^\star)h = (f^\star g)^\star h$

Let $f \times g$ denote $\langle f \circ \mathsf{fst}, g \circ \mathsf{snd}\rangle\colon A \times B \to A' \times B'$ where $f\colon A \to A'$ and $g\colon B \to B'$. It is easy to check some obvious properties of this notation like $(f \times g) \circ (f' \times g') = (f \circ f') \times (g \circ g')$ and $(f \times g) \circ \langle f', g'\rangle = \langle f \circ f', g \circ g'\rangle$.

Let

$$\alpha_{A,B,C} = \langle \mathsf{id} \times \mathsf{fst}, \mathsf{snd} \circ \mathsf{snd}\rangle\colon\ A \times (B \times C) \to (A \times B) \times C;$$
$$\alpha^{-1}_{A,B,C} = \langle \mathsf{fst} \circ \mathsf{fst}, \mathsf{snd} \times \mathsf{id}\rangle\colon\ (A \times B) \times C \to A \times (B \times C).$$

Obviously, α and α^{-1} are mutualy inverse. Analogously, we define *unitors*:

$$\lambda_A = \big(A \times 1 \xrightarrow{\mathsf{fst}} A\big), \qquad\qquad \rho_A = \big(1 \times A \xrightarrow{\mathsf{snd}} A\big)$$

for which $\lambda^{-1}_A = \langle \mathsf{id}_A, !\rangle$, $\rho^{-1}_A = \langle !, \mathsf{id}_A\rangle$.

Theorem 5 (Mac Lane's Coherence Theorem[1])**.** Any diagram with labels composed from $\mathsf{id}, \times, \alpha, \alpha^{-1}, \lambda, \lambda^{-1}, \rho, \rho^{-1}$ commutes.

2.1.3 Natural Transformations: Relating Functors

Associativity morphisms $\alpha_{A,B,C}$ are examples of natural transformations, which are a categorical formalization of parametric dependency.

Definition (Natural Transformation)**.** Let $\mathcal{C}, \mathcal{D}$ be categories and $F, G\colon \mathcal{C} \to \mathcal{D}$ be functors. A *natural transformation* $\vartheta\colon F \to G$ is a *family* of morphisms in $\mathcal{D}$:

$$(\vartheta_C\colon FC \to GC)_{C \in \mathsf{Ob}(\mathcal{C})},$$

such that, for any $f\colon C \to C'$ in $\mathcal{C}$, the following (naturality) diagram commutes:

$$\begin{tikzcd} FC \arrow[r, "\vartheta_C"] \arrow[d, "Ff"'] & GC \arrow[d, "Gf"] \\ FC' \arrow[r, "\vartheta_{C'}"] & GC' \end{tikzcd}$$

[1]simplified version

The morphisms $\vartheta_C : FC \to GC$ are called *components* of $\vartheta\colon F \to G$.

Intuitively, natural transformations are such morphisms $\vartheta_C : FC \to GC$ that do not use any information about C. Instead of saying "$\vartheta\colon F \to G$ is a natural transformation" one often uses equivalent formulation "$\vartheta_C\colon FC \to GC$ is a morphism natural in C".

Semantically, naturality corresponds to a specific form of *parametric polymorphism.* Haskell functions are automatically polymorphic in the corresponding *type variables*, but not necessarily natural. E.g. Haskell's function

```
reverse :: [a] -> [a]
```

for list reversal is polymorphic in `a` as well as natural it in the categorical sense, but

```
sort :: Ord a => [a] -> [a]
```

for sorting lists is not natural, which is indicated by the *type constraint* "`Ord a =>`" telling that sorting is not independent of the type `a` – the result depends on the fact that `a` is an ordered type and on that how it is ordered.

Another example of a natural transformation:

```
maybeToList :: Maybe a -> [a]
maybeToList (Just a) = [a]
maybeToList Nothing  = []
```

Definition. For any functor F and natural transformation $\vartheta\colon G \to H$ we define natural transformations $\vartheta_F : GF \to HF$ and $F\vartheta\colon FG \to FH$ as follows:

$$(\vartheta_F)_X = \vartheta_{FX}$$
$$(F\vartheta)_X = F(\vartheta_X).$$

(Easy) exercise: show that ϑ_F and $F\vartheta$ are indeed natural transformations.

Remark A natural transformation $F \xrightarrow{\xi} G$ is often drawn as $\begin{tikzcd} \mathcal{C} \arrow[r, bend left, "F"] \arrow[r, bend right, "G"'] \arrow[r, phantom, "\Downarrow\xi"] & \mathcal{D} \end{tikzcd}$. This would be consistent with the notation $\xi\colon F \Rightarrow G$, which is often used for natural transformations. We simply write $\xi\colon F \to G$ instead, for, after all, natural transformations are just morphisms in the functor category $[F, G]$.

Theorem 6. Cat is defined as follows:

- $\mathsf{Ob}(\mathsf{Cat})$ are small Categories $\mathcal{C}$ (that is, those for which $\mathsf{Ob}(\mathcal{C})$ is a set).

- $\mathsf{Hom}(\mathcal{C}, \mathcal{D})$ is the class of all functors from $\mathcal{C}$ to $\mathcal{D}$.

Cat is itself a category with $\mathsf{id}\colon \mathcal{C} \to \mathcal{C}$ being the identity functor and $F \circ G$ being functor composition $\mathcal{C} \xrightarrow{G} \mathcal{D} \xrightarrow{F} \mathcal{E}$.

Proof. trivial. □

Theorem 7. Given two categories $\mathcal{C}$ and $\mathcal{D}$, $[C \Rightarrow \mathcal{D}]$ (or $[\mathcal{C}, \mathcal{D}]$), defined as follows:

- $\mathsf{Ob}([\mathcal{C} \Rightarrow \mathcal{D}])$ are functors from $\mathcal{C}$ to $\mathcal{D}$
- $\mathsf{Hom}(F, G)$ are natural transformations $\xi\colon F \to G$.

is again a category.

Proof.

1. $\mathsf{id} \circ \xi = \xi$: For any $f\colon A \to B$

$$\begin{tikzcd}
FA \arrow[r, "\xi_A"] \arrow[rr, bend left, "\xi_A"] \arrow[d, "Ff"] & GA \arrow[r, "\mathsf{id}_A"] \arrow[d, "Gf"] & GA \arrow[d, "Gf"] \\
FB \arrow[r, "\xi_B"] \arrow[rr, bend right, "\xi_B"'] & GB \arrow[r, "\mathsf{id}_B"] & GB
\end{tikzcd}$$

2. $\xi \circ \mathsf{id} = \xi$
3. $\xi \circ (\theta \circ \sigma) = (\xi \circ \theta) \circ \sigma$

Properties 2 and 3 are analogous to proof. □

Pointwise composition of natural transformations ($(\xi \circ \theta)_A = \xi_A \circ \theta_A$) is called *vertical composition*:

$$\begin{tikzcd}
\mathcal{C} \arrow[r, bend left=60, "F"{name=F}] \arrow[r, "G"{name=G}] \arrow[r, bend right=60, "H"'{name=H}] & \mathcal{D}
\arrow[from=F, to=G, phantom, "\Downarrow\theta"]
\arrow[from=G, to=H, phantom, "\Downarrow\xi"]
\end{tikzcd}$$

Definition (Horizontal composition)**.** Given $\xi\colon F \to F'$ and $\theta\colon G \to G'$,

$$\xi \circ \theta\colon GF \to G'F'$$

is defined by the diagram:

$$\begin{tikzcd}
\mathcal{C} \arrow[r, bend left, "F"] \arrow[r, bend right, "F'"'] \arrow[r, phantom, "\Downarrow\xi"] & \mathcal{D} \arrow[r, bend left, "G"] \arrow[r, bend right, "G'"'] \arrow[r, phantom, "\Downarrow\theta"] & \mathcal{E}
\end{tikzcd}$$

Notation. Given $\xi\colon F \to G$, we can form:

$$H\xi\colon HF \to HG$$
$$\xi_U\colon FU \to GU$$

with

$$(H\xi)_A = H(\xi_A)$$
$$(\xi_U)_A = \xi_{UA}$$

Proposition. Given $\xi\colon F \to F'$ and $\theta\colon G \to G'$ then $\xi \circ \theta = (\theta_{F'}) \circ (G\xi)$

Example. $\mathsf{elems}_A : [A] \to \mathcal{P}(A)$ defined as follows:

$$\mathsf{elems}_A([l_1, \dots, l_2]) = \{l_1, \dots, l_n\}$$

yields a natural transformation $\mathsf{elems}\colon [\ ] \to \mathcal{P}$ of endofunctors over Sets.

Naturality: Let $f\colon A \to B$. Then

$$(\mathcal{P}f) \circ \mathsf{elems} \circ ([l_1, \dots, l_n]) = (\mathcal{P}f) \circ \{l_1, \dots, l_n\} = \{f(l_1), \dots, f(l_n)\}.$$

On the other hand:

$$(\mathsf{elems}_B \circ [f])[l_1, \dots, l_n] = \mathsf{elems}_B([f(l_1), \dots, f(l_n)]) = \{f(l_1), \dots, f(l_n)\}.$$

Notation (Natural transformation in two arguments)**.** The natural transformation

$$\tau_{A,B}\colon A \times TB \to T(A \times B)$$

can be defined as $\tau\colon F \to G$ where F and G are functors $F, G\colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, where $\mathcal{C} \times \mathcal{C}$ is the *product category*:

$$F(A, B) = A \times TB$$
$$G(A, B) = T(A \times B)$$

and similar definitions for morphisms.

Proposition. Let $F, G\colon \mathcal{C} \times D \to \mathcal{E}$, then $\xi_{A,B}\colon F \to G$ is natural (in A, B), iff

$$\begin{tikzcd}
F(A \times B) \arrow[r, "\xi_{A,B}"] \arrow[d, "F(f\times g)"] & G(A \times B) \arrow[d, "G(f\times g)"] \\
F(A' \times B') \arrow[r, "\xi_{A',B'}"] & G(A' \times B')
\end{tikzcd}$$

commutes for any $f\colon A \to A'$ and $g\colon B \to B'$.

We now can give a new (equivalent) definition of a monad.

Definition (Monad)**.** A monad on a category $\mathcal{C}$ consists of an *endofunctor* $T : \mathcal{C} \to \mathcal{C}$, and natural transformations

$$\underbrace{\eta\colon \mathsf{Id} \to T}_{unit}, \qquad \underbrace{\mu\colon T \circ T \to T}_{multiplication}$$

satisfying *triangle identities*:

$$\begin{tikzcd} TTTX \arrow[r, "\mu_{TX}"] \arrow[d, "T\mu_X"'] & TTX \arrow[d, "\mu_X"] \\ TTX \arrow[r, "\mu_X"] & TX \end{tikzcd} \qquad \begin{tikzcd} TX \arrow[r, "\eta_{TX}"] \arrow[dr, "\mathsf{id}_{TX}"'] & TTX \arrow[d, "\mu_X"] & TX \arrow[l, "T(\eta_X)"'] \arrow[dl, "\mathsf{id}_{TX}"] \\ & TX & \end{tikzcd}$$

i.e. the equations

$$\begin{aligned} \mu \circ \mu_T &= \mu \circ T\mu \\ \mu \circ \eta_T &= \mathsf{id} = \mu \circ T\eta. \end{aligned}$$

Proposition. Given a Kleisli-Triple $(T', \eta', \_^\star)$ satisfying the monad laws, one obtains a monad in the sense defined above in the following way:

$$\begin{aligned} Tf &= (\eta \circ f)^\star \quad \text{for } f\colon X \to Y \\ TX &= T'X \\ \eta_X &= \eta'_X \\ \mu_X &= (\mathsf{id}_{TX})^\star \end{aligned}$$

2.1.4 Examples of Monads

IO Monad

Example.

$$\begin{aligned} &\texttt{instance Monad IO} \\ &\texttt{getLine}\colon \texttt{IO String} \simeq 1 \to \texttt{IO String} \\ &\texttt{putString}\colon \texttt{String} \simeq \texttt{String} \to \texttt{IO}\,1 \end{aligned}$$

```
do x <- getLine; putStr $ x ++ "!"
```

Rough intuition: $\texttt{IO}\,A = \text{World} \to (A \times \text{World})$

$$\begin{aligned} &\texttt{getLine}\colon 1 \to (\texttt{World} \to (\texttt{String} \times \texttt{World})) \\ &\texttt{getLine}(x)(w) = (\mathsf{extrstr}(w), w) \\ &\texttt{putString}(s)(w) = (1, \mathsf{sendToWorld}(s, w)) \end{aligned}$$

State Monad

$$TS = S \to (X \times S) \simeq (X \times S)^S$$

This works in Sets, Cpos and more generally in Cartesian closed categories.

$$\begin{aligned}
&\eta_X : X \to (X \times S)^S \\
&\eta_X(x)(s) = \langle x, s\rangle \\
&f : X \to (Y \times S)^S \\
&f^\star : (X \times S)^S \to (Y \times S)^S \\
&f^\star(p)(s) = \mathsf{let}\langle x, s'\rangle = p(s)\ \mathsf{in}\ f(x)(s')
\end{aligned}$$

With let being defined like this:

$$\mathsf{let}\langle x, y\rangle = p\,\mathsf{in}\,q \quad = \quad q[\mathsf{fst}\,p/x, \mathsf{snd}\,/y]$$

The state monad supports the following operations:

$$\begin{aligned}
&\text{put} : S \to T1 && \text{put}(s)(s') = (*, s) \\
&\text{get} : 1 \to TS && \text{get}(*)(s) = (s, s)
\end{aligned}$$

Example (Writer Moand)**.**

$$TX = M \times X \quad \text{(where } M \text{ is a Monoid)}$$

Example (Reader Monad)**.**

$$TX = X^S$$

The Reader Monad is a submonad of the State monad:

$$\begin{aligned}
&\alpha_X : X^S \to (X \times S)^S \\
&\alpha_X(p)(s) = (p(s), s)
\end{aligned}$$

Theorem 8. $TX = X^S$ is a monad.

Continuation Monad In Sets: $TX = \underbrace{(X \to R)}_{\text{Continuation}} \to \underbrace{R}_{\text{Result}}$

$$\begin{aligned}
&\eta_X(x) = \lambda k.\ k(x) \\
&(f : X \to (R^Y \to R))^\star(p : R^X \to R) = \lambda k : Y \to R.\ p(\underbrace{\lambda x. f(x)(k)}_{X \to R})
\end{aligned}$$

The following lemma helps to prove that the continuation monad is indeed a monad in an abstract way.

Lemma. Let $F\colon \mathcal{C} \to \mathcal{D}$ be a functor and let T be a map $\mathsf{Ob}(\mathcal{C}) \to \mathsf{Ob}(\mathcal{C})$. Suppose that for any $X, Y \in \mathsf{Ob}(\mathcal{C})$, the hom-sets $\mathsf{Hom}(X, TY)$ and $\mathsf{Hom}(FX, FY)$ are isomorphic naturally in X. Then T is a monad with the following induced structure

$$\eta = \widecheck{\mathsf{id}} \qquad\qquad f^\star = \widecheck{\widehat{f}\,\widehat{\mathsf{id}}}$$

where $\widehat{f}\colon FX \to FY$ and $\widecheck{g}\colon X \to TY$ are the obvious isomorphic images of $f\colon X \to TY$ and $g\colon FX \to FY$ correspondingly.

Moreover, the Kleisli category of T is isomorphic to the full subcategory of $\mathcal{D}$ over the objects of the form FX.

Proof. The naturality condition means precisely that $\hat{f}(Fh) = \widehat{fh}$ for any $h\colon X \to Y$ and $f\colon Y \to TY$. This entails that $\widecheck{g(Fh)} = \widecheck{g}h$ for $g = \widecheck{f}$ and moreover,

$$f^\star g = \widecheck{\widehat{f}\,\widehat{\mathsf{id}}}g = \widecheck{\widehat{f}\,\widehat{\mathsf{id}}\,Fg} = \widecheck{\widehat{f}\,\widehat{\mathsf{id}\,g}} = \widecheck{\widehat{f}\,\widehat{g}}\,.$$

Therefore,

$$\eta^\star = \widecheck{\widehat{\widecheck{\mathsf{id}}}\,\widehat{\mathsf{id}}} = \widecheck{\widehat{\mathsf{id}}} = \mathsf{id}$$

$$f^\star\eta = \widecheck{\widehat{f}\,\widehat{\eta}} = \widecheck{\widehat{f}} = f$$

$$(f^\star g)^\star = \widecheck{(\widehat{f}\,\widehat{g})\,\widehat{\mathsf{id}}} = \widecheck{\widehat{f}\,(\widehat{g}\,\widehat{\mathsf{id}})} = f^\star\,\widecheck{\widehat{g}\,\widehat{\mathsf{id}}} = f^\star\,g^\star,$$

and we are done. $\square$

This can be instantiated as follows.

Example. For the state monad $TX = (X \times S)^S$, $\mathsf{Hom}_{\mathcal{C}}(X, TY) \cong \mathsf{Hom}_{\mathcal{C}}(X \times S, Y \times S)$.

For the continuation monad $TX = (X \to R) \to R$, $\mathsf{Hom}_{\mathcal{C}}(X, TY) \cong \mathsf{Hom}_{\mathcal{C}^{\mathsf{op}}}(R^X, R^Y) = \mathsf{Hom}_{\mathcal{C}}(R^Y, R^X)$.

2.1.5 Dualization, Bi-Functors, Cartesian Closure

Definition (Dual Category)**.** Given a category $\mathcal{C}$, the *dual category* $\mathcal{C}^{\mathsf{op}}$ is defined as follows:

- $\mathsf{Ob}(\mathcal{C}^{\mathsf{op}}) = \mathsf{Ob}(\mathcal{C})$;
- $\mathsf{Hom}_{\mathcal{C}^{\mathsf{op}}}(X, Y) = \mathsf{Hom}_{\mathcal{C}}(Y, X)$.

Example. Let $\mathcal{C}$ be a poset category, i.e. $\mathsf{Hom}_{\mathcal{C}}(X, Y) = \{\star\}$ iff $X \leqslant Y$. Then $\mathcal{C}^{\mathsf{op}}$ is the dually ordered poset: $\mathsf{Hom}_{\mathcal{C}^{\mathsf{op}}}(X, Y) = \{\star\}$ iff $X \geqslant Y$.

For example, we now can formally state that products are dual to coproducts.

Proposition. For every $\mathcal{C}$, a binary product $\mathcal{C}^{\mathsf{op}}$ is a binary coproduct of $\mathcal{C}^{\mathsf{op}}$.

Definition (Contravariant Functor)**.** A functor $F\colon \mathcal{C}^{\mathsf{op}} \to \mathcal{D}$ is said to be a *contravariant functor* from $\mathcal{C}$ to $\mathcal{D}$.

Small categories themselves form a category with finite products: the final object is the category of one object and one arrow, and a product of categories $\mathcal{C}$ and $\mathcal{D}$ is the category $\mathcal{C} \times \mathcal{D}$ with

- $\mathsf{Ob}(\mathcal{C} \times \mathcal{D}) = \mathsf{Ob}(\mathcal{C}) \times \mathsf{Ob}(\mathcal{D})$,
- $\mathsf{Hom}_{\mathcal{C}\times\mathcal{D}}((X,Y),(X',Y')) = \mathsf{Hom}_{\mathcal{C}}(X,X') \times \mathsf{Hom}_{\mathcal{D}}(Y,Y')$.

The category of all categories is not a category, more precisely, the locally small categories do not form a locally small category (but they form a category in a higher sense). Still, products of locally small categories make perfect sense regardless of this issue.

Definition (Bi-Functor)**.** A *bifunctor* is a functor $\mathcal{C} \times \mathcal{D} \to \mathcal{E}$ for which one also uses the notation $F(A,B)$ instead of $F(A \times B)$ and $F(f,g)$ instead of $F(f \times g)$.

Example (Product Functor)**.** Let $\mathcal{C}$ have binary products. Then $F\colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ sending A, B to $A \times B$ is a bi-functor with $F(f,g) = f \times g$.

Example (Hom-Functor)**.** The *hom-functor* is the bi-functor $\mathsf{Hom}(-,-)\colon \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \to \mathbf{Set}$.

Now, instead of saying that $\alpha\colon F \to G$ is a natural transformation, one often says that a family $\alpha_A\colon FA \to GA$ is *natural in* A, e.g. for bi-functors, $F\colon \mathcal{C} \times \mathcal{D} \to \mathcal{E}$, naturality of $\alpha_{A,B}\colon F(A \times B) \to G(A \times B)$ in A and B. Another example: associativity $\alpha_{A,B,C}\colon A \times (B \times C) \to (A \times B) \times C$ is natural in A, B, C.

Definition (Cartesian Closure)**.** A category $\mathcal{C}$ is *Cartesian closed* (CCC) if it is Cartesian, and for any objects B and C there is an object B^C, called an *exponential*, for which we have an isomorphism

$$\mathsf{curry}\colon \mathsf{Hom}(A \times B, C) \cong \mathsf{Hom}(A, C^B)$$

which is natural natural in A, meaning that

$$\begin{tikzcd}
\mathsf{Hom}(A \times B, C) \arrow[r, "\mathsf{curry}"] \arrow[d, "\mathsf{Hom}(f \times B, C)"'] & \mathsf{Hom}(A, C^B) \arrow[d, "\mathsf{Hom}(f, C^B)"] \\
\mathsf{Hom}(A' \times B, C) \arrow[r, "\mathsf{curry}"] & \mathsf{Hom}(A', C^B)
\end{tikzcd}$$

On the left side we go from $A \times B \to C$ to $A' \times B \to C'$ by post-composing with $f \times \mathsf{id}$ where $f\colon A' \to A$. On the right side we post-compose with f, i.e. the diagram expresses the following equation, where $g : A \times B \to C$:

$$(\mathsf{curry}\, g) \circ f = \mathsf{curry}(g \circ (f \times \mathsf{id}))$$

It is easy to see that the naturality condition for $\mathsf{uncurry} = \mathsf{curry}^{-1}$

$$\mathsf{uncurry}(g \circ f) = (\mathsf{uncurry}\, g) \circ (f \times \mathsf{id})$$

is derivable.

Again, we can define the evaluation transformation

$$\mathsf{ev} = \mathsf{uncurry}(\mathsf{id}\colon C^B \to C^B)\colon C^B \times B \to C.$$

Proposition. In any CCC $\mathcal{C}$, A^B extends to a bi-functor $(\text{–})^{(\text{–})}\colon \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \to \mathcal{C}$ sending $f\colon A' \to A$ and $g\colon B \to B'$ to

$$\mathsf{curry}(B^A \times A' \xrightarrow{\mathsf{id}\times f} B^A \times A \xrightarrow{\mathsf{ev}} B \xrightarrow{g} B')\colon B^A \to B'^{A'}.$$

Proposition. In any CCC curry and $\mathsf{uncurry}$ are natural in all parameters.

2.2 Tensorial Strength

We can generalize the call-by-value semantics of PCF along the following lines:

- replace $(\text{–})_\bot$ with T;
- replace "let" with the "do";
- replace $\lfloor - \rfloor$ with return.

This should work for any CCC with suitable carriers $[\![Bool]\!]$, $[\![Nat]\!]$ and a fixpoint operator $\mathit{fix}\colon (TA \to TA) \to TA$. Recall the semantics of types:

- $[\![1]\!] = 1$;
- $[\![A \times B]\!] = [\![A]\!] \times [\![B]\!]$;
- $[\![A \to B]\!] = [\![A]\!] \to T[\![B]\!]$.

Now, the semantics of a term in context $\Gamma \vdash t\colon A$ with $\Gamma = (x_1\colon A_1, \ldots, x_n\colon A_n)$ must be a morphism $[\![A_1]\!] \times \ldots \times [\![A_n]\!] \to T[\![A]\!]$. This works alright, and we could also incorporate the do-notation in the language (modulo replacing TX with X in the return types):

$$\frac{\Gamma \vdash p\colon A \qquad \Gamma, x\colon A \vdash q\colon B}{\Gamma \vdash \mathsf{do}\, x = p; q\colon B}$$

Here we have:

$$\begin{aligned} f &= [\![\Gamma \vdash p\colon A]\!]\colon [\![\Gamma]\!] \to T[\![A]\!] \\ g &= [\![\Gamma, x\colon A \vdash q\colon B]\!]\colon [\![\Gamma]\!] \times [\![A]\!] \to T[\![B]\!] \end{aligned}$$

from which we expect to obtain:

$$[\![\Gamma \vdash \mathsf{do}\, x = p; q\colon B]\!]\colon [\![\Gamma]\!] \to T[\![B]\!]$$

We would expect to have

$$[\![\Gamma]\!] \xrightarrow{\langle \mathsf{id}, f\rangle} [\![\Gamma]\!] \times T[\![A]\!] \xrightarrow{\ ?\ } T([\![\Gamma]\!] \times [\![A]\!]) \xrightarrow{g^\star} T[\![B]\!]$$

That is, we need means to incorporate the context Γ into a computation of type A.

2.2.1 Strong Monads

We arrive at the following notion.

Definition (Tensorial Strength)**.** A strong functor is a functor $F\colon \mathcal{C} \to \mathcal{D}$ between Cartesian categories $\mathcal{C}$ and $\mathcal{D}$, plus *strength*, which is a natural transformation $\tau_{A,B}\colon A \times FB \to F(A \times B)$, such that

$$\begin{tikzcd} 1 \times FX \arrow[r, "\mathsf{snd}"] \arrow[d, "\tau"'] & FX \\ F(1 \times X) \arrow[ur, "F\,\mathsf{snd}"'] & \end{tikzcd} \qquad \begin{tikzcd} (X \times Y) \times FZ \arrow[rr, "\tau"] \arrow[d, "assoc"'] & & F((X \times Y) \times Z) \arrow[d, "F\,assoc"] \\ X \times (Y \times FY) \arrow[r, "X \times \tau"] & X \times F(Y \times Z) \arrow[r, "\tau"] & F(X \times (Y \times Z)) \end{tikzcd}$$

Strong natural transformations are those that preserve strength in the obvious sense. Given a strong functor (F, τ), note that $(\mathrm{Id}, \mathsf{id}\colon X \times Y \to X \times Y)$ and $(FF, (F\tau)\tau\colon X \times FFY \to FF(X \times Y))$ are again strong functors.

Now, a monad is *strong* if it is strong as a functor and η, μ are strong natural transformations, concretely,

$$\begin{tikzcd} X \times Y \arrow[r, "\eta"] \arrow[d, "\mathsf{id} \times \eta"'] & T(X \times Y) \\ X \times TY \arrow[ur, "\tau"'] & \end{tikzcd} \qquad \begin{tikzcd} X \times TTY \arrow[rr, "\mathsf{id} \times \mu"] \arrow[d, "\tau"'] & & X \times TY \arrow[d, "\tau"] \\ T(X \times TY) \arrow[r, "T\tau"] & TT(X \times Y) \arrow[r, "\mu"] & T(X \times Y) \end{tikzcd}$$

The reason why we do not see strength when programming in Haskell is because Haskell functors $F\colon \mathcal{C} \to \mathcal{C}$ are indeed natural transformations $A^B \to FA^{FB}$ (as opposed to categorical functors $\mathsf{Hom}(A, B) \to \mathsf{Hom}(FA, FB)$). Categorically, this is in fact, a quite specific condition.

Definition (Functorial Strength)**.** An endofunctor $F\colon \mathcal{C} \to \mathcal{C}$ on a CCC $\mathcal{C}$ is functorially strong, if it comes with a *functorial strength*, i.e. a family of morphisms

$$\rho_{A,B}\colon B^A \to FB^{FA},$$

such that

$$\begin{tikzcd} \mathsf{Hom}(1 \times A, B) \arrow[r, "\cong"] \arrow[d, "\mathsf{curry}"'] & \mathsf{Hom}(A, B) \arrow[r, "F"] & \mathsf{Hom}(FA, FB) \arrow[r, "\cong"] & \mathsf{Hom}(1 \times FA, FB) \arrow[d, "\mathsf{curry}"] \\ \mathsf{Hom}(1, B^A) \arrow[rrr, "\mathsf{Hom}(1, \rho_{A,B})"'] & & & \mathsf{Hom}(1, FB^{FA}) \end{tikzcd}$$

Moreover, ρ must respect internal units ($\mathsf{curry}(\mathsf{snd})\colon 1 \to A^A$) and composition ($B^A \times C^B \to C^A$) in an obvious sense.

Analogously, we can internalise natural transformations and define "functorialy strong monad" as those functorially strong functors, for which there are internalized version of η and μ.

It turns out however that tensorial strength and functorial strength are equivalent:

$$\tau_{A,B} = \mathsf{uncurry}\big(A \xrightarrow{\mathsf{curry\,id}} (A \times B)^B \xrightarrow{\rho} T(A \times B)^{TB}\big),$$
$$\rho_{A,B} = \mathsf{curry}\big(B^A \times TA \xrightarrow{\tau} T(B^A \times A) \xrightarrow{T\,\mathsf{ev}} TB\big).$$

Example. Every endofunctor and every monad on *Set* are strong with the functorial strength being just the functorial action, because there is no distinction between hom-sets $\mathsf{Hom}(A, B)$ and exponentials B^A. Hence $\tau_{A,B}(x \in A, p \in TB) = (T\lambda y.\,\langle x, y\rangle)(p)$ (now we see, what this expression actually means!)

Every monad on predomains is thus also strong – this amounts to verifying that the above τ is continuous.

Categorically, the right setup for these considerations is *enriched categories*. These generalize standard categories by replacing *hom-sets* with *hom-objects* of a yet another category $\mathcal{V}$, in which the original category is said to be *enriched*. This produces the whole spectrum of derived notions: $\mathcal{V}$-functors, $\mathcal{V}$-natural transormations, $\mathcal{V}$-monads, etc. From this perspective our categories are *Set*-categories, i.e. categories enriched in *Set*. Every Cartesian closed category can be regarded as enriched over itself, because we can use exponentials A^B instead of hom-sets $\mathsf{Hom}(B, A)$. In that sense strong functors turn out to be precisely the enriched functors and strong monads turn out to be precisely the enriched monads. As a slogan: *in CCC strength is equivalent to enrichment*[2].

Is there non-strong monads? They are not easy to meet in the wild.

Example (Non-Strong Monad)**.** In the category of *two-sorted sets* $Set^2 = Set \times Set$ the monad $(X, Y) \mapsto (X, Y + X)$ is not strong.

2.2.2 Commutative Monads

We can classify computational effects according the equations they satisfy. Recall that the lifting monad satisfies the commutativity property:

$$\mathsf{let}\; x = p \;\mathsf{in}\; \mathsf{let}\; y = q \;\mathsf{in}\; \lfloor \langle x, y\rangle \rfloor = \mathsf{let}\; y = q \;\mathsf{in}\; \mathsf{let}\; x = p \;\mathsf{in}\; \lfloor \langle x, y\rangle \rfloor,$$

Definition (Commutative Monad)**.** A strong monad T is commutative if

$$\begin{tikzcd}
TA \times TB \arrow[r, "\tau"] \arrow[d, "\hat\tau"'] & T(TA \times B) \arrow[r, "T\hat\tau"] & TT(A \times B) \arrow[dd, "\mu"] \\
T(A \times TB) \arrow[d, "T\tau"'] & & \\
TT(A \times B) \arrow[rr, "\mu"] & & T(A \times B)
\end{tikzcd}$$

[2] Anders Kock. "Strong Functors and Monoidal Monads". In: *Archiv der Mathematik* 23.1 (1972), pp. 113–120.

This is the same as claiming

$$\mathsf{do}\, x = p;\ \mathsf{do}\, y = q;\ \mathsf{return}\langle x, y\rangle = \mathsf{do}\, y = q;\ \mathsf{do}\, x = p;\ \mathsf{return}\langle x, y\rangle.$$

Further important properties:

- *copyability:* $\mathsf{do}\, x = p;\ \mathsf{do}\, y = p;\ \mathsf{return}\langle x, y\rangle = \mathsf{do}\, x = p;\ \mathsf{return}\langle x, x\rangle$;
- *discardability:* $\mathsf{do}\, x = p;\ \mathsf{return}\,\star = \mathsf{return}\,\star$.

Example. Powerset monad is commutative, but neither copyable, nor discardable.

Example (Probabilistic Computations)**.** The following is a probability distribution monad on *Set*:

- $DX = \{d\colon X \to [0,1] \mid \sum d = 1\}$ (it follows that the set $\{x \mid d(x) \neq 0\}$ is countable);
- $(\eta x)(x) = 1$ and $(\eta x)(y) = 0$ if $x \neq y$ (*Dirac's distribution*);
- $(f\colon X \to DY)^\star(d\colon X \to [0,1])(y \in Y) = \sum_{x \in X} d(x) \cdot f(x)(y)$.

This monad is commutative and discardable, but not copyable.

2.3 Algebras and CPS-Transormations

Definition (Monad Algebras)**.** An *(Eilenberg-Moore) algebra* for a monad T, or a T-*algebra* is a tuple $(A, a\colon TA \to A)$ satisfying the following conditions:

$$\begin{tikzcd} A \arrow[r, "\eta_A"] \arrow[dr, equal] & TA \arrow[d, "a"] \\ & A \end{tikzcd} \qquad \begin{tikzcd} TTA \arrow[r, "Ta"] \arrow[d, "\mu_A"'] & TA \arrow[d, "a"] \\ TA \arrow[r, "a"] & A \end{tikzcd}$$

We call the object A of a T-algebra $(A, a\colon TA \to A)$ the *carrier* of the latter and the morphism $a\colon TA \to A$ the corresponding *structure*. As expected, morphisms of T-algebras are those morphisms of carrier that preserve the structure:

$$\begin{tikzcd} TA \arrow[r, "Th"] \arrow[d, "a"'] & TB \arrow[d, "b"] \\ A \arrow[r, "h"] & B \end{tikzcd}$$

We thus a category of T-algebras, of the Eilenberg-Moore category of T.

Example (Pointed Sets)**.** Let T be the maybe-monad $TX = X+1$. Then $(A, a\colon A + 1 \to A)$ is a T-algebra iff

$$\begin{tikzcd} A \arrow[r, "\mathsf{inl}"] \arrow[dr, equal] & A+1 \arrow[d, "a"] \\ & A \end{tikzcd} \qquad \begin{tikzcd} (A+1)+1 \arrow[r, "a+1"] \arrow[d, "{[\mathsf{id},\mathsf{inr}]}"'] & A+1 \arrow[d, "a"] \\ A+1 \arrow[r, "a"] & A \end{tikzcd}$$

The former diagram means precisely that a is of the form $[\mathsf{id}, p]$ for some $p\colon 1 \to A$ and the latter diagram commutes automatically. Therefore, to give a maybe-algebra over A is to give a morphism $1 \to A$, i.e. specify a *point* in A. A morphism of algebras $h\colon (A, a\colon A+1 \to A) \to (B, b\colon B+1 \to B)$ is exactly a morphism $h\colon A \to B$ of the carriers that respects the points.

Example (Monoids)**.** Let TX be the list monad over *Set*: $TX = X^\star$. It can be shown that the category of list-algebras is isomorphic to the category of monoids, defined as follows:

- objects are monoids $(M, \odot\colon M \times M \to M, e \in M)$;
- morphisms from $(M, \odot, e)$ to $(M', \odot', e')$ are those maps $h\colon M \to M'$, which preserve the monoid structure: $h(a \odot b) = h(a) \odot' h(b)$, $h(e) = e'$.

Definition (Free Algebras)**.** A free T-algebra on an object $A \in \mathsf{Ob}(\mathcal{C})$ is the tuple $(TA, \mu_A\colon TTA \to TA)$.

The axioms of T-algebras are automatics for free algebras.

Definition (Strong Monad Morphisms)**.** Given two monads S and T on the same category, a natural transformation $\alpha\colon S \to T$ is a monad morphism if

$$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, "\eta_X"'] & SX \arrow[d, "\alpha_X"] \\ & TX \end{tikzcd} \qquad \begin{tikzcd} SSX \arrow[r, "\alpha_{SX}"] \arrow[d, "\mu_X"'] & TSX \arrow[r, "T\alpha_X"] & TTX \arrow[d, "\mu_X"] \\ SX \arrow[rr, "\alpha_X"] & & TX \end{tikzcd}$$

A monad morphism between two strong monads is strong if it is a strong natural transformation.

Monad algebras, strong monad morphisms and continuations are connected in the following theorem.

Theorem 9 (Dubuc's Theorem[3,4])**.** Given a strong monad T, T-algebra structures over $(A, a\colon TA \to A)$ are in one-to-one correspondence with strong monad morphisms $\alpha\colon T \to (- \to A) \to A$ as follows:

- given $(A, a\colon TA \to A)$,

$$\alpha_X = \mathsf{curry}\Big(TX \times (X \to A) \xrightarrow{\cong} (X \to A) \times TX \xrightarrow{(T\,\mathsf{ev})\tau} TA \xrightarrow{a} A\Big);$$

- given $\alpha\colon T \to (- \to A) \to A$,

$$a = \Big(TA \xrightarrow{\langle \mathsf{id}, \mathsf{curry}\,\mathsf{snd}\rangle} TA \times (A \to A) \xrightarrow{\mathsf{uncurry}\,\alpha} A\Big).$$

[3] Eduardo J Dubuc. "Enriched semantics-structure (meta) adjointness". In: *Rev. Union Math. Argentina* 25 (1970), pp. 5–26.
[4] simplified version

If A is a free T-algebra $A = TR$ then $\alpha(p\colon TX)(f\colon X \to TR) = f^\star(p)$. Moreover, $\alpha(p\colon TR)(\eta\colon R \to TR) = \eta^\star(p) = p$. This can be illustrated with a series of Haskell programs. The program over the list monad

```
ex1 :: [Int]
ex1 = do
  a <- return 2
  b <- return 2
  return $ a+b
```

forms a list `[4]`. We can use just the same code for this purpose:

```
ex2 :: Cont String Int
ex2 = do
  a <- return 2
  b <- return 2
  return $ a+b
```

However, since the result type is `String`, in the end we will need to convert from `Int` to `String`, e.g. with `runCont ex2 show`. In contrast to the list monad we now can "escape" from the computation:

```
ex3 :: Cont String Int
ex3 = do
    cont (\r -> "escape")
    a <- return 2
    b <- return 2
    return $ a+b
```

Now, if we start with the program

```
ex4 :: [Int]
ex4 = do
  a <- [1,2]
  b <- [1,2]
  return $ a + b
```

which yields `[2,3,3,4]`, we can use the CPS-transform of the list monad to convert to the continuation monad:

```
i x = cont (\r -> x >>= r)

ex5 :: Cont [Int] Int
ex5 = do
  a <- i [1,2]
  b <- i [1,2]
  return $ a + b
```

Here `[Int]` is the free list-algebra on `Int` and `i` is the induced monad morphism. With `runCont ex5 return` we obtain `[42]` like in the original case of the list monad. But now we also can escape from the computation:

```
ex6 :: Cont [Int] Int
ex6 = do
  cont (\r -> [42])
  a <- i [1,2]
  b <- i [1,2]
  return $ a + b
```

The same can be achieved with the library function `callCC :: MonadCont m => ((a -> m b) -> m a) -> m a` (=*call with current continuation*):

```
ex7 :: Cont [Int] Int
ex7 = callCC $ \k -> do
  k 42
  a <- i [1,2]
  b <- i [1,2]
  return $ a + b
```

2.4 Free Objects and Adjoint Functors

Definition (Free Objects)**.** Given a functor $G\colon \mathcal{C} \to \mathcal{D}$, a *free* $\mathcal{C}$*-object on* $X \in \mathsf{Ob}(\mathcal{D})$ consists of an object $Y \in \mathsf{Ob}(\mathcal{C})$ together with a *morphism* $\eta_X\colon X \to GY$ in $\mathcal{D}$ such that for any other $Z \in \mathsf{Ob}(\mathcal{C})$ and morphism $f\colon X \to GZ$ in $\mathcal{D}$, there exists a unique $f^\dagger\colon Y \to Z$ in $\mathcal{C}$ such that

$$\begin{tikzcd} & GZ \arrow[d, "Gf^\dagger"] \\ X \arrow[ur, "\eta_X"] \arrow[r, "f"'] & GY \end{tikzcd}$$

Example (Exponentials)**.** Let $\mathcal{C} = \mathcal{D}$ and let $GX = X^A$. Then $\eta_X\colon X \to X \times A$ is a free object on A and $\mathsf{ev}(f \times A)\colon X \times A \to Z$ is the universal map induced by $f\colon X \to Z^A$.

Example (Free Monoids)**.** Let $\mathcal{C}$ be the category of monoids over $\mathcal{C}$ and let G be the obvious forgetful functor. Then $\eta\colon X \to X^*$ is a free monoid on X and for every $f\colon X \to Y$, $f^\dagger\colon X^\star \to Y$ is a unique extension of f to a monoid map from $X^\star$ to Y.

Example (Free Algebras)**.** Let $\mathcal{C}$ be the category of T-algebras over $\mathcal{D}$ and $G\colon \mathcal{C} \to \mathcal{D}$ a forgetful functor. Let $F\colon \mathcal{D} \to \mathcal{C}$ be the free T-algebra functor. Then $(FX, \eta_X\colon X \to GFX = TX)$ is the free object on X.

Definition (Adjointness)**.** A functor $F\colon \mathcal{D} \to \mathcal{C}$ is a *left adjoint* of $G\colon \mathcal{C} \to \mathcal{D}$ if $\mathsf{Hom}(FX, Y) \cong \mathsf{Hom}(X, GY)$ naturally in X and Y. This is written as $F \dashv G$ or $G \vdash F$ and G is called a *right adjoint* to F.

Theorem 10. A functor $G\colon \mathcal{C} \to \mathcal{D}$ has a left adjoint $F\colon \mathcal{D} \to \mathcal{C}$ iff there exist free algebras $(FX, \eta_X\colon X \to GFX)$ for every X:

- from an adjunction $\mathsf{Hom}(FX, Y) \cong \mathsf{Hom}(X, GY)$ we obtain a correspondence

$$(f\colon X \to GY) \mapsto (f^\dagger\colon FX \to Y)$$

such that $(\eta_X\colon X \to GFX)^\dagger = \mathsf{id}_{FX}$ for a suitable η_X;

- from free algebras $(FX, \eta_X\colon X \to GFX)$, we obtain the maps

$$(f\colon FX \to Y) \mapsto ((Gf)\eta\colon X \to GY),$$
$$(f\colon X \to GY) \mapsto (f^\dagger\colon FX \to Y).$$

Theorem 10 allows us to switch between two equivalent ways of defining categorical structures: by adjunctions or by free objects. The latter way is more fine grained, because we can speak about existence of *specific* free objects, while the adjoint formulation is only sensible when *all* free objects exist.

Example (Exponential)**.** Existence of exponentials, now can be reformulated as $(-) \times A \dashv (-)^A$. Theorem 10 show that this definition is equivalent the the definition via free objects.

By Theorem 10, we now see that $F \dashv G$ for F being the free T-algebra functor and G being the corresponding forgetful functor. This is called the *Eilenberg-Moore adjunction.* Because of Theorem 10, it is easy to see that we could just as well consider the category of free T-algebras instead of the category of all algebras. The resulting adjunction is called the *Kleisli adjunction.* The reason for it is the following

Proposition. The Kleisli category of a monad is isomorphic to the category of all free algebras of that monad. The relevant isomorphism is defined as follows:

- *(from Kleisli for free algebras):*

$$X \mapsto (TX, \mu_A), \qquad (f\colon X \to TY) \mapsto (f^\star\ TX \to TY);$$

- *(from free algebras to Kleisli):*

$$(TX, \mu_A) \mapsto X \qquad (f\colon (TX, \mu_X) \to (TY, \mu_Y)) \mapsto (f\eta\ X \to TY)$$

Bibliography

Barendregt, Hendrik Pieter. *The Lambda calculus: Its syntax and semantics.* Amsterdam: North-Holland, 1984.

Dubuc, Eduardo J. "Enriched semantics-structure (meta) adjointness". In: *Rev. Union Math. Argentina* 25 (1970), pp. 5–26.

Kock, Anders. "Strong Functors and Monoidal Monads". In: *Archiv der Mathematik* 23.1 (1972), pp. 113–120.