λ騎士団

Chapter3 : Categorical Models of Simple Types

mmichish

目 次

1 Introduction **2**

2 Products and Cartesian Closure **2**

2.1 Diagram 4

2.2 Binary product 5

2.3 Tuples 6

2.4 Cartesian closed categories 8

2.5 Exercises 9

3 λ-Calculus with Constants and Products **9**

3.1 Ccc models 10

1 Introduction

圏論やるぞやるぞやるぞ（そこまでやらない）

圏論とは、異なる部分から同じ結果のコレクションを探すためのテクニックであり、これは数学の構造や数学的な関係性をより一般的に捉える手法である（超意訳)。圏論はプログラミング言語の意味論の研究に極めて強い影響を与え、正しいコンセプトや定義、構造、定理の発見を導いている。

Semantics of Programming Language の勉強をするツールとしての圏論は非常に有用である。多くの場合、「圏論的視点」はそのほかの基本論理よりも、コンピュータサイエンスの根本的な動機と非常によくマッチする。

ただ、圏論は非常に大きな主題であるし、それを開発すること自体は我々の目的ではない。そのため、この章はその主題の小さな一部に対しての斜め方向からのイントロ、という風に捉えるのがいい。 まずは、圏論とコンピュータサイエンスのアプリケーションとの基本的な関係から始める。application と abstraction というコンセプトは圏論的には cartesian closed category （デカルト閉圏）の構造として知られている.

どうせここからやってくだろうが、以下に我らが集合知、大正義 wikipedia 様にお問い合わせした結果を記載しておく。

> 圏論において、カテゴリーがデカルト閉（デカルトへい、英語: cartesian closed）であるとは、大雑把に言えば任意の二つの対象の直積上で定義される射が直積因子の一方で定義される射と自然に同一視できることである。デカルト閉な圏はラムダ計算の自然な設定ができるという点で数理論理学およびプログラミングの理論において特に重要である。デカルト閉圏の概念はモノイド圏に一般化される（モノイド閉圏を参照)。

何いってんだこいつ。ともかく、この本が真に self contained であるならば、ここから圏や射、直積などの説明があり、cartesian closed category の話があり、ラムダ計算との関係の話があるはず。

実際のところ流れとしては

1. Category の説明
2. Binary Product の説明
3. Tuple の話と Finite Product の話
4. Cartesian Closed Category の話
5. ラムダ計算と CCC の話 (← この辺までしか読めてない)

のような形になっている雰囲気。

2 Products and Cartesian Closure

category の定義を行う。この axiom はかなり一般的であり、数学やコンピュータサイエンスで簡単に見つけることができる。

Definition

category を $\boldsymbol{C}$ を構成するものは

- A collection $Ob(\boldsymbol{C})$ of objects and collection $Ar(\boldsymbol{C})$ of arrows.
- Operations

$$dom : Ar(\boldsymbol{C}) \to Ob(\boldsymbol{C})$$
$$codom : Ar(\boldsymbol{C}) \to Ob(\boldsymbol{C})$$

- A composition operator, which assigns to arrow f, g such that $dom(f) = codom(g)$ an arrow $f \circ g : dom(g) \to codom(f)$ such that the associativity axiom holds:

$$(f \circ g) \circ h = f \circ (g \circ h)$$

for any $h : \boldsymbol{A} \to \boldsymbol{B}, g : \boldsymbol{B} \to \boldsymbol{C}, f : \boldsymbol{C} \to \boldsymbol{D}$.

- For each object $\boldsymbol{A}$, an identity arrow $id_A : \boldsymbol{A} \to \boldsymbol{A}$ such that $id_B \circ f = f$ and $f \circ id_A = f$ for every $f : \boldsymbol{A} \to \boldsymbol{B}$

いくつか具体的な圏の例を見てみる。

Set 圏 (`object = Set, arrow = (total) function`)
よく知られた圏。
合成の定義は関数合成に対応し、自己写像も恒等写像に対応する.

$$f \circ g = x \mapsto f(g(x))$$
$$id_A = \backslash x \to x$$

Set' 圏 (`object = Set, arrow = inclusion`)
包含写像 (inclusion) : 集合 A, B が, $A \subseteq B$ であるとき、$x \in A$ を B の各元として扱う写像 $f(x) = x : A \to B$ のこと.
この圏では、object のペアの間に最大で 1 つの arrow が存在する (inclusion が存在する or not ということか？)
合成は通常の関数合成で、自己写像は $f(x) = x : A \to A$ かな？ inclusion とは実装は同じだけど codomain が違う感じ？(あまり自信なし)

Poset 圏 (`object = Poset, arrow = monotone`)
(復習) poset : 二項関係 $\sqsubseteq \subseteq P \times P$ が、transitive, reflexive, anti-symmetric を満たすような集合 P. (P, $\sqsubseteq_P$), (Q, $\sqsubseteq_Q$) などとして与えられる。
(復習) 単調写像 (monotone) : (P, $\sqsubseteq_P$), (Q, $\sqsubseteq_Q$) が与えられ、関数 $f : P \to Q$ が、$x \ \sqsubseteq_P \ y$ を満たす ($x, y \in P$) なら、$f(x) \ \sqsubseteq_Q \ f(y)$ を満たす場合、関数 f は monotone という。
合成の定義と自己写像は Set 圏と同じ.

Monoid と homomorphismn による圏 (`object = monoid, arrow = homomorphisms of monoid`)
Monoid 圏は別にあるので [1]、ここでは冗長だが上記のように書いておく
monoid : 集合、結合律を満たす二項演算、単位元の triple $(M, \cdot, e)$ から構成される代数的構造。
monoid 間の homomorphism f を考える。ここで、f が homomorphism であることは以下を満たす。(構造の保存)

$$f(x \cdot_M y) = f(x) \cdot_N f(y)$$
$$f(e_M) = e_N$$

(気持ち) ここでいう構造を保つとは、M (ないしは N) 上の単位元と二項演算を保つこと。M 側で演算を行ってから map しても ($f(x \cdot_M y)$), それぞれの様子を map してから N 側で演算しても ($f(x) \cdot_N f(y)$) 同じになる。単位元も map すると単位元になる。

ex) M=([A], ++, []), N=(N, +, 0), homomorphisms = length

$$\text{length}([a_0, \cdots, a_n] \mathbin{++} [b_0, \cdots, b_m]) = \text{length}([a_0, \cdots, a_n]) + \text{length}([b_0, \cdots, b_m])$$
$$\text{length}([]) = 0$$

ある圏 $\boldsymbol{C}$ が存在する時、別の圏 $\boldsymbol{D}$ の objects が圏 $\boldsymbol{C}$ の objects であり、圏 $\boldsymbol{D}$ の objects(A, B) について、$\boldsymbol{D}(A, B) \subseteq \boldsymbol{C}(A, B)$ である場合、$\boldsymbol{D}$ は $\boldsymbol{C}$ の subcategory と呼ぶ
ex. monoid の二項演算の条件に、commutative ($x \cdot y \ = \ y \cdot x$) が入れば、これは commutative monoid (可換モノイド) となり、commmutative monoid と homomorphisms で monoid 圏の subcategory を形成できる。(ほんまか？ちゃんと考えてない)

2.1 Diagram

おえかきのじかんだよー

圏 $\boldsymbol{C}$ の diagram は、頂点が objects, edges が arrow であるような有向グラフを書く。

edge の始点が domain, 終点が codomain であるように書く。

下記の図であれば、edge はそれぞれ射 $f : A \to B$ と $g : A \to B$ でラベルづけされている。

$$A \xrightarrow{f} B \xrightarrow{g} C$$

頂点として A, B のすべてのペアに対して、その間を結ぶ射 $f1, ..., fn$ と $g1, ..., gm$ が存在しており、

$$f_n \circ \cdots \circ f_2 \circ f_1 = g_m \circ \cdots \circ g_2 \circ g_1$$

を満たす時、その diagram は可換 (commute) であるという。

例えば以下の図では、iff. $q \circ f = g \circ p$ であれば可換である。

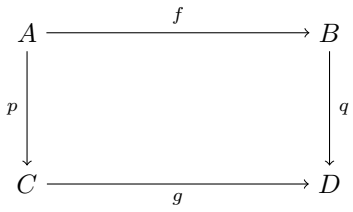
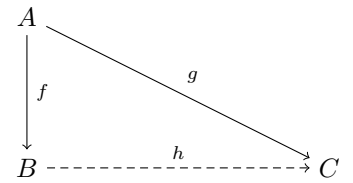


diagram を可換にするような射が存在することを強調するために、ドットラインで強調することもある。（この射は diagram を完成させる (completes the diagram) ともいう。）

例えば以下は、$h \circ f = g$ を満たすような射 $h : B \to C$ が存在していることを (強調して) 示している。



また、エクスクラメーションマークは、diagram を完成させる射が unique であることを強調するために利用される。

例えば以下は、$q \circ f = g \circ p$ であって、全ての $g : C \to D$ に対して、$g = \hat{f}$ であることを示している。

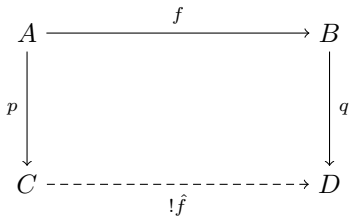
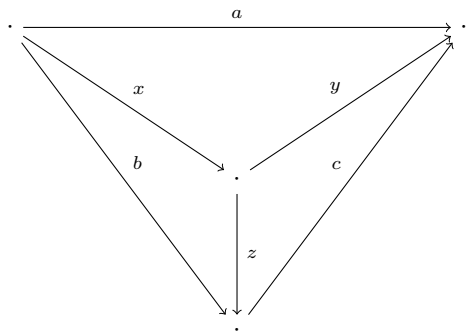


diagram の一部が可換であることを組み合わせて、diagram 全体の可換を言いたい (示すのは Exercies3.1 にあった)。例えば、以下の例では、それぞれの三角形が可換であれば下記が成り立つ。これは、diagram chase （図式追跡）などと呼ばれる。

$$a = y \circ x = (c \circ z) \circ x = c \circ (z \circ x) = c \circ b$$



2.2 Binary product

一般的に使い慣れた演算子を説明するための基盤という意味で category に興味がある。最もよく使われているものの一つとして、product と pairing operator があるらしい。

product： $A \times B$ of sets A and B is the set of ordered pairs (a, b) where $a \in A$ and $b \in B$

集合論の教科書では、$(a, b) = \{\{a\}, \{a, b\}\}$ として記述することがある。（ほんまか？ どういう気持ちや？ よくわからん）

別の表現の仕方もできるよね。重要なのは pair が fst, snd coodinate をもっており、それらによって unique に決定されるということ。ここで *projection* と呼ばれる関数 fst, snd を考える。

$$fst = (a, b) \mapsto a : A \times B \to A$$
$$snd = (a, b) \mapsto b : A \times B \to B$$
$$\forall x \in A \times B.\ x = (fst(x), snd(x))$$

つまり pair は, fst, snd によって一意に決まる。

しかし、構造の直積という考え方はもっと一般的である。ここで構造として *poset* を利用した直積を考える。$(P, \sqsubseteq_P)$ と $(Q, \sqsubseteq_Q)$ があり、それらの集合 P, Q の元での直積は $P \times Q$ であり、これは以下で定義される $\sqsubseteq_{P \times Q}$ を伴う。

$$(x, y) \sqsubseteq_{P \times Q} (x', y') \text{ iff. } x \sqsubseteq_P x' \wedge y \sqsubseteq_Q y'$$

つまり、pair という構造は、それぞれの元 P, Q における順序構造を保存するような $\sqsubseteq_{P \times Q}$ が定義される。projection はそれぞれの元に対する写像となり、構造を保っているため、monotone map であることが簡単にわかる。

同様に monoid の product($M \times N$) も考えてみる。この時、monoid は構造として二項演算を保持していたので、product の component ごとにその演算を行うことが可能で ($(x, y) \cdot (x', y') = (x \cdot x', y \cdot y')$)、そのように product の演算を定義することができる。この場合、もちろんその unit はその monoid 演算の unit の pair になる (e_M, e_N)。

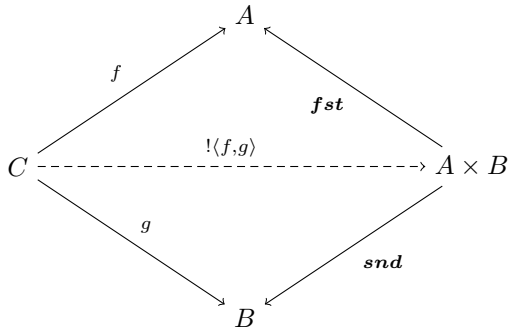
ここからは、ここまでみてきたようなオペレーションの共通の構造を圏論の言葉で表現していく。

実際に、これらは圏論における射で定義することができる !!!

Definition

いままでみてきたのはこれだぞ（）

$\boldsymbol{C}$ は圏、圏 C の object A, B の product の object($A \times B$)、$\boldsymbol{fst}$, $\boldsymbol{snd}$ があり、$f : C \to A,\ g : C \to B\ (if\ C \in Ob(\boldsymbol{C}))$ について, 以下の diagram を可換にする unique な射 $(!\langle f, g \rangle)$ が存在する。



パッと出されたけどこの説明ほぼテキストにない.

Product の要素となる値を移す関数 $f : A \to B$ 及び $f' A' \to B'$ が与えられていると、それらを利用して Product から Product に移す関数を構成できる.

$$f \times f' = \langle f \circ fst,\ f' \circ snd \rangle : A \times A' \to B \times B'$$

(ここから書き方の話)

計算のなかで一つ以上の product が出てくる場合には、異なる project の projection をちゃんと区別することが不可欠。タグ付けしましょうねという話。

（厳密に言えば、object A, B 及びその product である $A \times B$ を知る必要があるが、product operation （$\times$）と射影の定義がわかれば、object A, B がわかるだけで良い。）

下付き文字で $fst_{A,B}$, $snd_{A,B}$ と書くか、自明な場合は省略するといい。それでも計算によっては下付き文字が面倒だったり、それらを省略するとそれはそれで面倒なことになる。これらの理由から、square bracket で書くことにするよ。$(fst[A, B], snd[A, B])$ カンマを $\times$ に置き換えた表記をすることが時にはある（$fst[A \times B], snd[A \times B]$）が、これは $A \times B$ が A, B を一意に決定しない場合があるため、その時の記法？(どううういうこっちゃ?) object A は category 内の id_A を unique に決定し、シンボルが射か object かを簡単に判断できるため、id_A を A と記述する。もし、関数 $f : A \to B$ を考えた時、identity の equation に関しては以下のように記述する。

$$B \circ f = f$$
$$f \circ A = f$$

(ここまで書き方の話)

先程の圏が product と projection の特性を表現していることを証明する方法の例として、ある射 $h : C \to A \times B$ について

$$\langle fst \circ h, snd \circ h \rangle = h$$

これって上でコメントアウトした内容と同じでは？

しかし、product の定義から、$\langle fst \circ h,\ snd \circ h \rangle$ は unique である。したがって、Equation3.1 が成り立ったなければならない。（いや、Equation3.1ってなんだよ）。Equation3.1 のインスタンスから $\langle fst, snd \rangle = id$ を得ることができる？

category 内の object の structural identity という考え方は次のように定義される。

Definition

内容自体は p68 参照.

圏 $\boldsymbol{C}$ の objectA, B において、射 $f : A \to B$ を考える。もし、射 $g : B \to A$ が存在し、$g \circ f = id_A$ かつ $f \circ g = id_B$ を満たす場合、f は *isomorphism*(同型写像) という。A, B の間に isomorphism が存在する場合、A と B は同型であるといい、$A \cong B$ と書く。

product を含むより複雑な計算の例を考える。

3.1 ***Proposition*** In a category with products, $A \times (B \times C) \cong (A \times B) \times C$

Proof

$A \times (B \times C) \to (A \times B) \times C$ なる射 α と $(A \times B) \times C \to A \times (B \times C)$ なる射 β が $\alpha \circ \beta = id$ であることを示せば良い。(Note に書いた。⇒ TODO: ここに書く)

product は、isomorphism も以下のように一意に決定される。

3.2 ***Proposition***

圏 C と C の中の object$A$, B がある。この時, $A \times B$ が fst, snd という projections による product であり、$A \times' B$ が fst', snd' という projections による product であるときに、$A \times B \cong A \times' B$

Proof

(Note に書いた。⇒ TODO: ここに書く)

2.3 Tuples

product の考え方を、ordered pair (of objects) から list (of objects) に一般化しましょう。この定義を induction で行なっていきたいからまずは、degenerate case(0 object の場合) を考えたい。

Definition: 終対象

圏 $\boldsymbol{C}$ における *terminal object*($\mathbf{1}$) とは、全ての object A について、その terminal object に対して unique な射 $(t_A : A \to \mathbf{1})$ が存在するような object である。

ex) $\boldsymbol{Set}$ の場合、終対象は x

この例から分かる通り Category 内の終対象はいくつか存在し、それらは equal ではない。しかし、終対象の定義から以下に示す通り、終対象同士は同型になる。

Proof

ある Category の終対象 $\mathbf{1}, \mathbf{1'}$ を考える. 定義より、$\mathbf{1'}$ から $\mathbf{1}$ に対しての射 (t) および、その逆の射 (t') が unique に存在する。これらを合成して得られる射 $(t \circ t', t' \circ t)$ は、それぞれの終対象における identity$(id_1, id_{1'})$ と等しいため、$\mathbf{1} \cong \mathbf{1'}$ である。

射 $(p : \mathbf{1} \to A)$ を用いることで、終対象を持つ圏の object A の"要素"を表現することができる。それらの射は object A の *points* と呼ばれている。Set などの圏に置いて、通常の意味での集合 A の要素と、圏論的な意味での A の points は正確に対応する。終対象 $\mathbf{1}$ からある object$A$ に対して射がない場合には、object$A$ は empty object であるし、射があるのであれば、その object は non-empty である。後の section では、non-empty objects を simply-typed λ-calculus として解釈する。また、終対象は product の degenerate case になる。ここでようやく、finite product (list?) を持つ圏 $(\boldsymbol{C}, \times, \mathbf{1})$ について考える。このような圏の例として Set 圏を取り上げると、$(\boldsymbol{Set}, \times, \mathbf{1})$ であり、$\mathbf{1} = \{\emptyset\}$ である。ここで products は上記の triple であると言ったが、product と terminal object の choise には重要な意味はあまりないため、以降では簡単に $\boldsymbol{C}$ を finite products を含む category を指すことにする。(degenerate case と product を inductive に利用して定義する意味から明確に triple と書いたくらいの意味かな?)

3.3 ***Proposition***

finite product を伴う圏 $\boldsymbol{C}$ を考える。$\boldsymbol{C}$ に置いて終対象 $\mathbf{1}$ は product の unit である。すなわち、$\mathbf{1} \times A \cong A \times \mathbf{1} \cong A$
このような圏では、object $\times(A_1, ..., A_n) \to A_i$ を下記のように inductive に定義する。

$\times() = \mathbf{1}$
$\times(A) = \mathbf{1} \times A$
$\times(A_1, ..., A_n) = (\times(A_1, ..., A_{n-1})) \times A_n$

これに対して、projection も一般化する。

$\pi_i^n : \times(A_1, ..., A_n) \to A_i$
$\pi_n^n = snd$
$\pi_i^n = \pi_i^{n-1} \circ fst \ (i < n)$

である。pairing に関しても一般化する。

$\langle\rangle = t_1 : \mathbf{1} \to \mathbf{1}$
$\langle f \rangle = \langle t_1, f \rangle : A \to \times(A)$
$\langle f_1, ..., f_n \rangle = \langle \langle f_1, ..., f_{n-1} \rangle \times f_n \rangle : A \to \times(A_1, ..., A_n)$

その他に、finite production ということで、coodinate permute した場合を考える。$(1 \leq k < n)$
$A = \times(A_1, ..., A_k, A_{k+1}, ..., A_n)$ $A = \times(A_1, ..., A_{k+1}, A_k, ..., A_n)$
このような permute を行う射 τ_k^n は次のように定義できる。

$$\tau_k^n = \begin{cases} \langle \langle fst \circ fst, snd \rangle, snd \circ fst \rangle & if \ k+1 = n \\ \tau_k^{n-1} \times id_{A_n} & if \ k+1 < n \end{cases} \tag{1}$$

目的の直積部分 $(k+1 < n)$ までは id, 目的の部分 $(k+1 = n)$ で permute する関数になっている。$(\times(A) = 1 \times A$ が先頭とその次の要素の permute に効いてくる)

Lemma

$i \neq k \wedge i \neq k+1, 1 < k+1 \leq n$ のとき

1. $\pi_k^n[A] = \pi_{k+1}^n[A'] \circ \tau_k^n[A]$
2. $\pi_{k+1}^n[A] = \pi_k^n[A'] \circ \tau_k^n[A]$
3. $\pi_i^n[A] = \pi_i^n[A'] \circ \pi_k^n[A]$

この等式が成立するのは割と直感的にわかる。

1. τ_k^n で入れ替えた後の k+1 番目の projection は入れ替え前の k と同じ。
2. τ_k^n で入れ替えた後の k 番目の projection は入れ替え前の k+1 と同じ。
3. τ_k^n で入れ替えた後の、k, k+1 以外の projection は入れ替え前と同じ。

Proof

note に書いたがめんどいので省略 (やればできる)

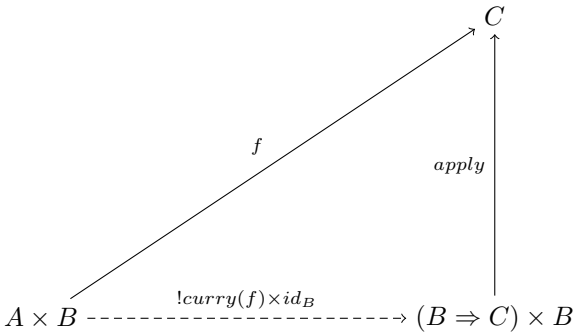
2.4 Cartesian closed categories

λ-calculus を圏論的に考えた場合、その構造は圏論的な関数空間に類似している. いくつかの category では、object$A$ から B への射の collection は、それ自身が category の object$(A \to B)$ である。簡単な例でいうと、Set 圏において、集合 A から B への関数はそれ自身が集合 B^A であるまたより面白い例としては poset と monotone function からなる圏である。monotone function 自体に natural ordering を考えた場合に、そのような order の上でのすべての monotone function は poset になる。これまでの議論で、product の圏論的特性は product の basic operator(pairing / projection) とそれらの equational property によって考えられるようになった。function の場合、basic operation に当たるものは application である。例えば、$f : A \to B$ と、$a \in A$ が与えられた時、結果として B の値を得るような f と a の binary operation(apply) が存在する。このような考えかたは environmental model の定義であるが、*exponential* の圏論的な定義に含まれるアイデアの一つでもある。ほかにも関数型言語を書く programer にとって馴染みのある考え方として、二引数関数 $f : A \times B \to C$ が与えられた時、この関数に $a \in A$ を適用した結果得られる $B \to C$ という関数 f' がある。この関数は f を「カリー化」した結果と言われる。

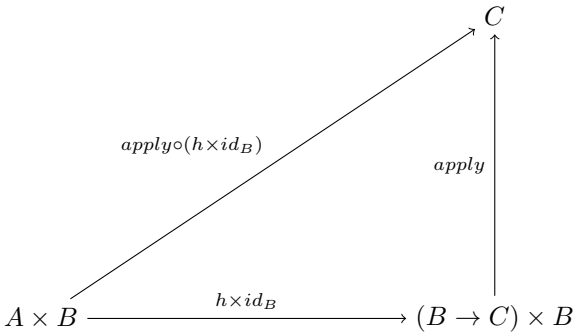
Definition: exponential

省略 (p73 参照)

気持ちとしては、$A \times B$ と C、そこの射 $f : A \times B \to C$ がある。その中で *exponential*、つまり $B \to C$ と B から C を得るような関数 *apply* を考える。前述より、$B \to C$ となるような射の集合はそれ自身が object $B \Rightarrow C$ なので、$(B \Rightarrow C) \times B$ から射 *apply* が C に向かって伸びる。このときに、この図式を可換にする射として、$A \times B$ から $(B \Rightarrow C) \times B$ への射 $!curry(f) \times B$ が与えられる。みたいな感じ？



また、A から $B \Rightarrow C$ への射 $h : A \to B \Rightarrow C$ が与えられたとすると、可換性から $f : A \times B \to C$ は, $apply \circ (h \times id_B)$ で与えられる。上述の式 $!curry(f) \times id_B$ より $h = curry(apply \circ (h \times id_B))$ が成り立つ。つまり、*exponential* の圏論的特性から、射 h は unique に決まる。



cartesianclosedcategory(ccc)

cartesian closed category (ccc) とは、$(\boldsymbol{C}, \times, \mathbf{1})$ という finite product と exponential operator$(\to)$ との組み合わせである、four-tuple$(\boldsymbol{C}, \to, \times, \mathbf{1})$ である。例えば、finite product$(Set, \times, \mathbf{1})$ を持つ圏（これはよくある）について、$A \to B = B^A$ という exponential ($A \to B$ という関数の集合) を加えたものは、cartesian closed category である。

applycation function は以下のように定義される。

$$apply(f, x) = f(x)$$

また、関数 $f : A \times B = C$ が与えられた時、

$$curry(f)(x)(y) = f(x, y)$$

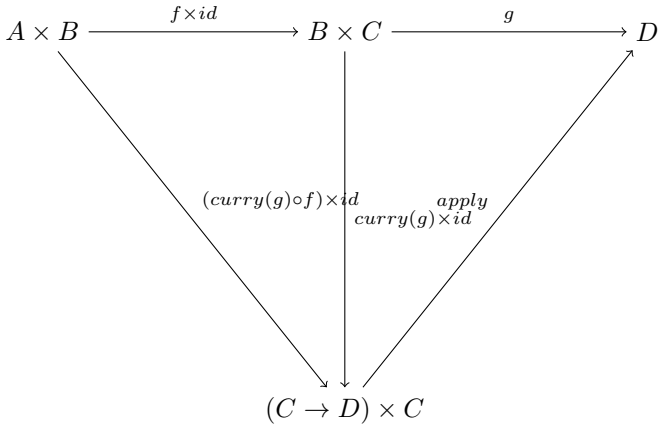
である。

これらの射が望ましい特性を持っていることを確認する。ccc の products は tuple であるが、$\to, \times, \mathbf{1}$ が重要ではない場合、または context から明らかな場合、ccc のことを単純に $\boldsymbol{C}$ とする。product と terminal object と同様に、exponential の選択は同型で unique であることを示すことができる。（ほんまか/示さんのかい）

exponential に関連する別の計算例として、$f : A \to B$ と $g : B \times C \to D$ という射が与えられた時、いかの equation を満たす。

$$curry(g) \circ f = curry(g \circ (f \times id_C))$$

なぜかというと、下の diagram についてそれぞれの inner diagram が可換だからである。



それゆえに外側の diagram も可換である。しかし、外側の diagram は exponentials の uniqueness を満たす必要がある。そのため、3.5 を見たす必要がある。(3.5 とは？)

2.5 Exercises

3 λ-Calculus with Constants and Products

2 章でやった、simply-typed λ-calculus を拡張する。今回拡張する内容は、Constant と Product である。(simply-typed λ-calculus with products and constants) これは、cartesian closed category でモデル化することができる。

いくつか記号を導入する

Σ_1 : collection of type constants 以下のような syntax で型が形成される。

$t ::= C \mid t \to t \mid t \times t \quad where\ C \in \Sigma_1$

Σ_0 : set of pairs (c, t) where c is a *term constant* and t is a *type* over Σ_1 constant に対しては、型との pair を取る？constant の評価が term のタイミングで行われ、変数のようにとりまわせないからだろうか？

if $(c, t) \in \Sigma_0$ and $(c, t') \in \Sigma_1$ then $t \equiv t'$

以上により、次のような Syntax を構成する

$M ::= c \mid x \mid \lambda x : t.\ M \mid MM \mid (M, M) \mid fst(M) \mid snd(M)$

typing rule については、2 章でやった simply-typed lambda calculus のものに加え、Table3.1(省略) のものを考える。2 章での simply-typed lambda calculus の rule における assignment, terms, types は今回の拡張である constants, products の計算の範囲を超えている。(どういういとの文章だ？超えているから同じように考えて問題ないという意味だろうか)

いくつか Table3.1 について補足をしておく。

- 任意の type assignment において、constant の型は term を考えるタイミングで型割り当てされ、signature は Σ_0 である。実質的に、constant は変数によく似ており、Σ_0 は type assignment H の暗黙の拡張であるが（つまり今までにすでにやってきていたこと)、それらは異なる Syntax class からきているため、constants は abstraction によって bind できない。 ($\lambda x : t$ の x は variable であり、constants をとりえない)
- $M : s,\ N : t$ と型付けされる場合、pair(M, N) の型は $s \times t$
- $M : s \times t$ と型付けされる場合、$fst(M) : s$ であり、$snd(M) : t$ である

simply-typed λ-calculus with constants and products における equation $(H \triangleright M = N : t)$ は基本的には simply-typed λ-calculus でやったものと同じだが、いくつか追加の equation がある (Table3.2)

- (M, N) の first, second condition はそれぞれ M, N により構成される。
- first, second coordinate が同じ M から構成される pair は、M そのものと等しい。
- pair の congruence rule (Exercises 3.19). 二つの pair の first, second coordinate がそれぞれ等しければ、その pair は等しい。

(かなり駆け足だが、ここで syntax の定義、equation の定義は終わり) (つぎに Model 側の話に入る)

3.1 Ccc models

λ-calculus with signature Σ の ccc model は、cartesian closed category $(\boldsymbol{C}, \to, \times, \mathbf{1})$ からなる。気持ちとしては、以前やった Type Frame, Full Frame などの際には, 解釈の方法として集合と関数を利用したが、今回は解釈の方法として ccc を利用しましょうという感じ。

先ずは型 $t := C \mid t \to\ t \mid t\ \times\ t$ についての interpretation を以下のように考える。

- それぞれの定数型 $C \in \Sigma_1$ は空でない object($\boldsymbol{C}[\![C]\!]$) に割り当てる.
- 関数型の interpretation($\boldsymbol{C}[\![s \to t]\!]$) はそれぞれの Object($\boldsymbol{C}[\![s]\!]$, $\boldsymbol{C}[\![t]\!]$) を、Object の間の射の collection($\to$) で結んだものに等しい.
- 直積型の interpretation($\boldsymbol{C}[\![s \times t]\!]$) は、それぞれの Object($\boldsymbol{C}[\![s]\!]$, $\boldsymbol{C}[\![t]\!]$) の直積 ($\times$) を取ったものに等しい.
- 上記性質からさらに、ある term constants (c) についての interpretation($\boldsymbol{C}[\![c]\!]$) は、終対象 (object) から、c の型に対応する object への射として表現される (ほんまか？)

表記を省くために、ここから先の議論では、Σ と $\boldsymbol{C}$ は固定とし、$\boldsymbol{C}[\![\,]\!]$ の代わりに $[\![\,\cdot\,]\!]$ と記述する。

$H = x_1 : t_1,\ \ldots\ , x_n : t_n$ が型割り当てであり、$H \vdash M : t$ となる。$A = \times([\![t_1]\!],\ \ldots\ , [\![t_n]\!])$ ($[\![t_n]\!]$ は Object に対応するため、$A = \times(A_1, ..., A_n)$ の定義 (A_n は object) より、この型割り当てにおいて A は $A = \times([\![t_1]\!], ..., [\![t_n]\!])$ と表現される.

型割り当て H において、$M : t$ の interpretation は射 $[\![H \triangleright M : t]\!] : A \to [\![t]\!]$ (object の直積からある object への射) として表現される。

この meaning function (おそらく, $[\![\,\cdot\,]\!]$ のこと) は再帰的に以下のように定義される

- Constants: $[\![H \triangleright c : t]\!] = [\![c]\!] \circ t_A\ \ where\ t = \Sigma_1(c)$
- Projection: $[\![H \triangleright x_i : t_i]\!] = \pi_i^n$
- Abstraction: $[\![H \triangleright \lambda x : u.N : u \to v]\!] = curry[\![H, x : u \to N : v]\!]$
- Application: $[\![H \triangleright L(N) : s]\!] = apply \circ \langle [\![H \triangleright L : t \to s]\!], [\![H \triangleright N : t]\!] \rangle$

- Pair: $[\![H \triangleright (L, N) : s \times t]\!] = \langle [\![H \triangleright L : s]\!], [\![H \triangleright N : t]\!] \rangle$
- First: $[\![H \triangleright fst(N) : s \times t]\!] = fst \circ [\![H \triangleright N : s \times t]\!]$
- Second: $[\![H \triangleright snd(N) : t]\!] = snd \circ [\![H \triangleright N : s \times t]\!]$

example)
$[\![\triangleright \lambda x : s.\lambda f : s \to t.f(x) : s \to (s \to t) \to t]\!]$
$= curry[\![x : s \triangleright \lambda f : s \to t.f(x) : (s \to t) \to t]\!]$ $(Abstraction)$
$= curry(curry[\![x : s, f : s \to t \triangleright f(x) : t]\!])$ $(Abstraction)$
$= curry(curry(apply \circ \langle [\![x : s, f : s \triangleright f : s \to t]\!], [\![x : s, f : s \triangleright x : s]\!] \rangle))$ $(Application)$
$= curry(curry(apply \circ \langle \pi_2^2, \pi_1^2 \rangle))$ $(Projection)$

我々はこの interpretation が equational rule に関して sound であることを示さねばならない。このモデルが theory T の equation を満たしていると仮定すると、equational axioms を利用して、導出可能なすべての equalities を満たすことを示す必要がある。equality の導出の高さに関する帰納法?によって証明する。各 rule が仮定のもとで、categorical interpretation に当てはまることを示すことから始める。ccc-model の場合、型割り当てに pair の順序が明示的に含まれるため、add, drop, permute rule の証明は type frame の時ほど簡単ではない.

permutation rule から行こう。そのために、Lemma3.5 を示す。これは元の型割り当てに関して、permutation した後の typing judgement の意味を記述するための補題である。

Lemma3.5

参考文献

[1] 猫番 — 圏としてのモノイド, http://eed3si9n.com/herding-cats/ja/Monoid-as-categories.html