TAPL ch.23
Universal Types

住井・松田研究室 B4 三上 陽向

内容

内容

23.1 モチベーション

22.7 で扱ったように，無数の “doubling” 関数を
単純型付きラムダ計算で書くことができる.

doubleNat = λf:Nat → Nat. λx:Nat. f (f x);

doubleRcd = λf:{l:Bool} → {l:Bool}. λx:{l:Bool}. f (f x);

doubleFun = λf:(Nat → Nat) → (Nat → Nat). λx:Nat → Nat. f (f x);

これらの関数は異なった型の引数に適用することができ，
まったく同じ動作をする.
(これらのプログラムは型注釈を除けば一致)

23.1 モチベーション

Abstraction Principle（抽象化原則）

プログラム内の重要な機能は、それぞれ1か所でのみ実装されるべきである。同じような機能が異なるコード片で実行されている場合、変化する部分を抽象化することにより統合することが一般的に有益である。

23.1 モチベーション

doubling operation を同一プログラム内で異なる型の引数に対して実行したいとき，それぞれの型 **T** に応じて **doubleT** を区別して定義しなければならない．

double = λf:doubleT → doubleT. λx:doubleT. f (f x);

このような切り貼りのプログラミングはソフトウェア工学の基本原則 (抽象化原則) に反する．

変化する部分は型

→　項から型を抽象化し，具体的な型注釈をあとで付加するような仕組みが必要

内容

23.2 polymorphism の種類

複数の型で単一のコードを使用できるようにする型システムは，総称して polymorphic システムと呼ばれる.

現代の言語にはいくつかの種類の polymorphism がみられる.

この節における polymorphism の分類は，Strachey (1967) および Cardelli and Wegner (1985) による.

23.2 polymorphism の種類

Parametric polymorphism

本章の主題.

実際の型の代わりに変数を使用して、コードを「一般的に」型付けし、その後必要に応じて具体的な型でインスタンス化する.

impredicative / first-class polymorphism

本章で扱う. プログラミング言語で人気が高まっており, MLのような言語の強力なモジュールシステムの技術的基盤を形成する.

ML-style / let-polymorphism

polymorphism をlet束縛に制限し, 多相的な値を引数にとることを禁止する代わりに, 自動の型再構築 (22章) による便利で自然な形式を提供する.

23.2 polymorphism の種類

Ad-hoc polymorphism

多相的な値が異なる型で”viewed”するたびに異なる挙動をするようにする.

一般的な例は overloading であり，これは1つの関数シンボルを複数の実装に関連づけ，コンパイラまたは実行時システムが引数の型に基づいて適切な実装を選択する.

Intentional polymorphism

Ad-hoc. のより強力な形式として知られている.

実行時に型に関する制限された計算が可能.

型情報に対する任意のパターンマッチを実行時に許可.

高度な実装技術が可能:

tag-free GC, “unboxed” 関数引数，polymorphic marshaling, “flattened” データ構造など

23.2 polymorphism の種類

Subtype polymorphism (15章)

subsumption 規則を使用して単一の項に多くの型を付与

項の挙動に関する情報を選択に「忘れる」ことが可能になる

$$\frac{\Gamma \vdash \mathbf{t} : \mathbf{S} \qquad \mathbf{S} <: \mathbf{T}}{\Gamma \vdash \mathbf{t} : \mathbf{T}} \quad \text{(T - SUB)}$$

$$\frac{\vdash \{x=0,\ y=1\} : \{x{:}Nat,\ y{:}Nat\} \qquad \{x{:}Nat,\ y{:}Nat\} <: \{x{:}Nat\}}{\vdash \{x=0,\ y=1\} : \{x{:}Nat\}} \quad \text{(T-SUB)}$$

(λr:{x:Nat}. r.x) {x = 0, y = 1} //型チェックを通る

23.2 polymorphism の種類

これらのカテゴリーは互いに排他的でない.
異なる polymorphism を混ぜて使うこともできる.

polymorphism という言葉自体,プログラミング言語のコミュニティ間で若干の混乱を引き起こす:

関数型プログラマー (MLやHaskell などを使用・設計する人々)

→ polymorphism ≒ parametric polymorphism

オブジェクト指向プログラマー

→ polymorphism ≒ subtype polymorphism

(parametric polymorphism には *genericity* や *generic* という用語がよく用いられる)

内容

23.3 System F

この章で使うシステムは一般に *System F* と呼ばれている.

- Jean-Yves Girard (1972) により論理の証明理論の文脈で発見.
- John Reynolds (1974) によりほぼ同じ能力を持つ型システムが独立して開発され，polymorphic lambda-calculus と命名された
- このシステムは 基礎研究のための手段として，また言語設計の基礎として広く使用されてきた.

23.3 System F

この章で使うシステムは一般に *System F* と呼ばれている.

System F は *second-order lambda-calculus* とも呼ばれる:

quantification (量化) を individuals [terms] だけでなく predicates [types] にも許す *Curry-Howard correspondence* を通じて second-order (二階) intuitionistic logic に対応するため

quantification (量化) https://ja.wikipedia.org/wiki/量化

言語や論理学において、論理式が適用される(または満足される)議論領域(ドメイン)の個体の「量」を指定すること。述語論理の基本量化子：∀, ∃

Curry-Howard correspondence (カリー = ハワード同型対応) https://ja.wikipedia.org/wiki/カリー=ハワード同型対応

計算機プログラムと数学的証明との間の直接的な対応関係。「プログラム＝証明」(proofs-as-programs)・「型＝命題」(formulae-as-types) などとしても知られる。

intuitionistic logic (直感主義論理) https://ja.wikipedia.org/wiki/直感主義論理

証明が存在する命題のみを真とする論理. 排中律・二重否定除去は許されない

second-order predicate logic (二階述語論理) https://ja.wikipedia.org/wiki/二階述語論理

ドメインに属する個々の値だけでなく, 個体の集合も変項の値として量化できるよう一階述語論理を拡張した論理体系

23.3 System F

System F の定義は単純型付きラムダ計算（$\lambda_{\to}$）の率直な拡張.

項から型を抽象化し、それを後から埋め込む仕組み:

type abstraction (型抽象) **λX. t**

type application (or *instantiation*, 型適用) **t [T]**

簡約規則

$(\lambda X.\, t_{12})\ [T_2] \to [X \mapsto T_2]\ t_{12}$ **(E-TappTabs)**

23.3 System F

type abstraction (型抽象)　　　　$\lambda X.\ t$

type application (or *instantiation*, 型適用)　$t\ [T]$

簡約規則　$(\lambda X.\ t_{12})\ [T_2] \rightarrow [X \mapsto T_2]\ t_{12}$　　(E-TappTabs)

例えば，多相的な高等関数

$$id = \lambda X.\ \lambda x : X.\ x;$$

を id[Nat] としてNatに適用すると，E-TappTabs により

$$[X \mapsto Nat]\ (\lambda x : X.\ x) \ = \ \lambda x : Nat.\ x$$

となる．

23.3 System F

最後に，多相抽象の型を定義する．

λx:Nat.x の型を **Nat→Nat** としたように，**id** のような多相関数を分類するためにはドメインが型であるような "arrow type" が必要．

例えば，**id** では最終的な型は適用する引数に依存したように，多相抽象と適用の型付け規則は依存関係を表現するように書かれなければならない．これらの規則は項レベルの抽象と適用の規則に類似している．

$$\frac{\Gamma, X \vdash t_2 : T_2}{\Gamma \vdash \lambda X.t_2 : \forall X.T_2} \qquad \text{(T-TAbs)}$$

$$\frac{\Gamma \vdash t_1 : \forall X.T_{12}}{\Gamma \vdash t_1\ [T_2] : [X \mapsto T_2]T_{12}} \qquad \text{(T-TApp)}$$

☆ **id** の型は **∀X. X→X** と書ける．

23.3 System F

注意

- 型変数 X は t の部分導出で使用する文脈 Γ に含まれるべき
- 5.3.4の規約により，いかなる変数名もすでに Γ に束縛された すべての名前と異なるように選ばれるべき
- ラムダで束縛された型変数は、この条件を満たすように任意に 名前を変更可能
- 現時点での Γ 内での型変数の唯一の役割は，スコープを追跡し，同じ型変数名が使用されないようにすること （26章，29章でさらに拡張される）

23.3 System F

次のページに，多相ラムダ計算の定義を示す.

ただし,

- $\lambda_{\rightarrow}$との違いはハイライトされている.
- 純粋な計算体系のみ定義されている.
- 型構成子(レコードなど)，基本型(Nat や Bool など)，項言語の拡張(let や fix など) は省略されている.
- これらは 23.4 examples で拡張される

23.3 System F

→ ∀ *Based on λ→ (9-1)*

Syntax

t ::=		*terms:*
	x	*variable*
	$\lambda x{:}T.t$	*abstraction*
	$t\ t$	*application*
	$\lambda X.t$	*type abstraction*
	$t\ [T]$	*type application*
v ::=		*values:*
	$\lambda x{:}T.t$	*abstraction value*
	$\lambda X.t$	*type abstraction value*
T ::=		*types:*
	X	*type variable*
	$T \to T$	*type of functions*
	$\forall X.T$	*universal type*
Γ ::=		*contexts:*
	$\varnothing$	*empty context*
	$\Gamma, x{:}T$	*term variable binding*
	Γ, X	*type variable binding*

Evaluation $\boxed{t \longrightarrow t'}$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ [T_2] \longrightarrow t_1'\ [T_2]} \quad \text{(E-TApp)}$$

$$(\lambda X.t_{12})\ [T_2] \longrightarrow [X \mapsto T_2]t_{12} \quad \text{(E-TappTabs)}$$

Typing $\boxed{\Gamma \vdash t : T}$

$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T} \quad \text{(T-Var)}$$

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.t_2 : T_1 \to T_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

$$\frac{\Gamma, X \vdash t_2 : T_2}{\Gamma \vdash \lambda X.t_2 : \forall X.T_2} \quad \text{(T-TAbs)}$$

$$\frac{\Gamma \vdash t_1 : \forall X.T_{12}}{\Gamma \vdash t_1\ [T_2] : [X \mapsto T_2]T_{12}} \quad \text{(T-TApp)}$$

Figure 23-1: Polymorphic lambda-calculus (System F)

内容

23.4 例

Polymorphism を使用したプログラミングのいくつかの例

<u>Warm–ups</u>

identity function

doubling function

self-application function

quadrupling function

23.4.1 Exercise

型付け規則を使い，それぞれの項が記載されたような型を持つことを確認する.

23.4 例

Warm–Ups and 23.4.1 Exercise

id = λX. λx:X. x

 # id : ∀X. X → X

$$\cfrac{\cfrac{\cfrac{x:X \in x:X,\ X}{X,\ x:X \vdash x : X}\ (\text{T-VAR})}{X \vdash \lambda x:X.\ x : X \to X}\ (\text{T-ABS})}{\emptyset \vdash \lambda X.\ \lambda x:X.\ x : \forall X.\ X \to X}\ (\text{T-TABS})$$

23.4 例

Warm–Ups and 23.4.1 Exercise

id [Nat]

 # <fun> : Nat → Nat

$$\dfrac{\emptyset \vdash id : \forall X.X \to X}{\emptyset \vdash id\,[Nat] : \underbrace{[X \mapsto Nat]X \to X}_{= Nat \to Nat}}\ (\text{T-TApp})$$

23.4 例

Warm–Ups and 23.4.1 Exercise

id [Nat] 0

 # 0 : Nat

$$\frac{\emptyset \vdash \mathrm{id}\,[\mathrm{Nat}] : \mathrm{Nat} \to \mathrm{Nat} \qquad \emptyset \vdash 0 : \mathrm{Nat}}{\emptyset \vdash \mathrm{id}\,[\mathrm{Nat}]\ 0 : \mathrm{Nat}}\ (\text{T-App})$$

23.4 例

Warm–Ups and 23.4.1 Exercise

double : λX. λf : X → X. λa : X. f (f a)

 # double : ∀X. (X → X) → X → X

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{f:X\to X\in\Gamma}{\Gamma\vdash f:X\to X}\,\text{(T-VAR)} \qquad \dfrac{\dfrac{f:X\to X\in\Gamma}{\Gamma\vdash f:X\to X}\,\text{(T-VAR)} \quad \dfrac{a:X\in\Gamma}{\Gamma\vdash a:X}\,\text{(T-VAR)}}{\Gamma\vdash f\;a : X}\,\text{(T-APP)}}{\underbrace{X,\, f:X\to X,\, a:X}_{\Gamma}\vdash f\,(f\,a) : X}\,\text{(T-APP)}}{X,\, f:X\to X\vdash \lambda a:X.\, f\,(f\,a) : X\to X}\,\text{(T-ABS)}}{X\vdash \lambda f:X\to X.\,\lambda a:X.\, f\,(f\,a) : (X\to X)\to X\to X}\,\text{(T-ABS)}}{\emptyset\vdash \lambda X.\,\lambda f:X\to X.\,\lambda a:X.\, f\,(f\,a) : \forall X.\,(X\to X)\to X\to X}\,\text{(T-TABS)}
$$

23.4 例

Warm–Ups and 23.4.1 Exercise

double : ∀X. (X → X) → X → X

doubleNat = double [Nat]

 # doubleNat : (Nat → Nat) → Nat → Nat

doubleNatArrowNat = double [Nat → Nat]

 # doubleNatArrowNat :((Nat → Nat) → Nat → Nat) →
(Nat → Nat) → Nat → Nat

double [Nat] (λx : Nat. succ (succ (x))) 3

 # 7 : Nat

導出木省略.

23.4 例

Warm–Ups and 23.4.1 Exercise

λx. x x は Polymorphism により型付け可能になる.

selfApp : λx : ∀X. X → X. x [∀X. X → X] x

 # selfApp : (∀X. X → X) → (∀X. X → X)

$$[X \to (\forall X. X \to X)]\,(X \to X).$$

$$\dfrac{\dfrac{\dfrac{\dfrac{x:\forall X.X\to X \in \forall X.X\to X}{x:\forall X.X\to X \vdash x:\forall X.\underbrace{X\to X}_{T_{12}}}\ (\text{T-VAR})}{x:\forall X.X\to X \vdash x\,[\underbrace{\forall X.X\to X}_{T_2}] : (\forall X.X\to X)\to\forall X.X\to X}\ (\text{T-TAPP}) \qquad \dfrac{x:\forall X.X\to X \in x:\forall X.X\to X}{x:\forall X.X\to X \vdash x:\forall X.X\to X}\ (\text{T-VAR})}{x:\forall X.X\to X \vdash x\,[\forall X.X\to X]\,x : \forall X.X\to X}\ (\text{T-APP})}{\emptyset \vdash \lambda x:\forall X.X\to X.\ x\,[\forall X.X\to X]\,x : (\forall X.X\to X)\to(\forall X.X\to X)}\ (\text{T-ABS})$$

23.4 例

Warm–Ups and 23.4.1 Exercise

quadruple : λx :double [X → X] (double [X])

 # quadruple : ∀X. (X → X) → X → X

$$
\dfrac{
\dfrac{X \vdash double : \forall X.(X\to X)\to X\to X}{X \vdash double\ [X\to X] : ((X\to X)\to X\to X)\to (X\to X)\to X\to X}\ \text{(T-TAPP)}
\qquad
\dfrac{X \vdash double : \forall X.(X\to X)\to X\to X}{X \vdash double\ [X] : (X\to X)\to X\to X}\ \text{(T-TAPP)}
}{
\dfrac{X \vdash double\ [X\to X]\ (double\ [X]) : (X\to X)\to X\to X}{\emptyset \vdash \lambda X.\ double\ [X\to X]\ (double\ [X]) : \forall X.(X\to X)\to X\to X}\ \text{(T-TABS)}
}\ \text{(T-APP)}
$$

23.4 例

Polymorphic Lists

多相型を用いたリスト操作について考える.

初めてリストを導入したとき (11章12節), いかなる型のリストにも同様の操作が適用できるように, 特別な推論規則を用意した.

一方, 多相型を使用することによりまったく同じ制約での演算を行うことができる.

23.4 例

Polymorphic Lists

型コンストラクタ **List** と，次のような型を持つリスト操作のためのプリミティブな項があるとする.

nil : ∀X. List X

cons : ∀X. X → List X → List X

isnil : ∀X. List X → Bool

head : ∀X. List X → X

tail : ∀X. List X → List X

23.4 例

Polymorphic Lists

nil, cons, isnil, head, tail を用いると，**map** 関数は次のように表現される.

map = λX. λY.
λf : X → Y.
(fix (λm : (List X) → (List Y).
λl : List X.
if isnil [X] l
then nil [Y]
else cons [Y] (f (head [X] l))
(m (tail [X] l))))

23.4 例

23.4.2 Exercise

map : ∀X. ∀Y. (X → Y) → List X → List Y であることの確認

直感的には，マッピング前後のリストの型を多相型X, Yで抽象化し，**fix** に **l** を適用すると，

- **l == nil** [X]ならば **nil** [Y]
- そうでなければ **f : X → Y** を先頭要素に適用し，残りについても **fix** により再帰的に**f** が適用される.

したがって，**map** は **X → Y** 型の関数 **f** と **List X** 型のリストを受け取り，**List Y** 型のリストを生成して停止するから，

map : ∀X. ∀Y. (X → Y) → List X → List Y

は直感的には正しいといえる.

23.4 例

23.4.3 Exercise [Recommended]

23.4 例

23.4.4 Exercise

第一引数がX型を持つ要素の比較関数であるような，多相型を用いたソート関数を書け.

sort : ∀X. (X → X → Bool) → (List X) → List X

23.4.4 Exercise

比較関数 cmp, 要素 a, 挿入するリスト l を受け取り, cmp を a と l の各要素に再帰的に適用し, cmp a (head l) == true となるとき head の前に a を挿入するような insert 関数を先に定義する.

```
insert = λX. λcmp: X → X → Bool. λa: X.
           (fix (λm: List X → List X.
                 λl: List X.
                     if isnil [X] l then cons [X] a (nil [X])
                     else if cmp a (head [X] l)
                                        then cons [X] a l
                     else cons [X] (head [X] l) (m (tail [X] l)) ))
```

23.4.4 Exercise

再帰的に挿入ソートを実行する．ソート済みリストの適切な位置に要素を挿入していく操作が再帰的に実行される．

```
# insert : ∀X. (X → X → Bool) → X → List X → List X

sort = λX. λcmp: X → X → Bool.
           (fix (λm: List X → List X.
                 λl : List X.
                      if isnil [X] l then nil [X]
                      else insert [X] cmp
                               (head [X] l) (m (tail [X] l ))))
```

http://pllab.is.ocha.ac.jp/harukam/okiba/tapl/tapl_23.html#sec23_4

23.4 例

Church Encodings

Church Encodings が *System F* でも実現可能であることを確認する.

[$\lambda_{\to}$] $\mathrm{tru} = \lambda t.\ \lambda f.\ t$

$\mathrm{fls} = \lambda t.\ \lambda f.\ f$

[System F]

$\mathrm{Cbool} : \forall X.\ X \to X \to X$

$\mathrm{tru} : \mathrm{CBool} = \lambda X.\ \lambda t : X.\ \lambda f : X.\ t$

$\mathrm{fls} : \mathrm{CBool} = \lambda X.\ \lambda t : X.\ \lambda f : X.\ f$

$\mathrm{not} = \lambda b : \mathrm{Cbool}.\ \lambda X.\ \lambda t : X.\ \lambda f : X.\ b\ [X]\ f\ t$

$\#\ \mathrm{not} : \mathrm{CBool} \to \mathrm{CBool}$

23.4 例

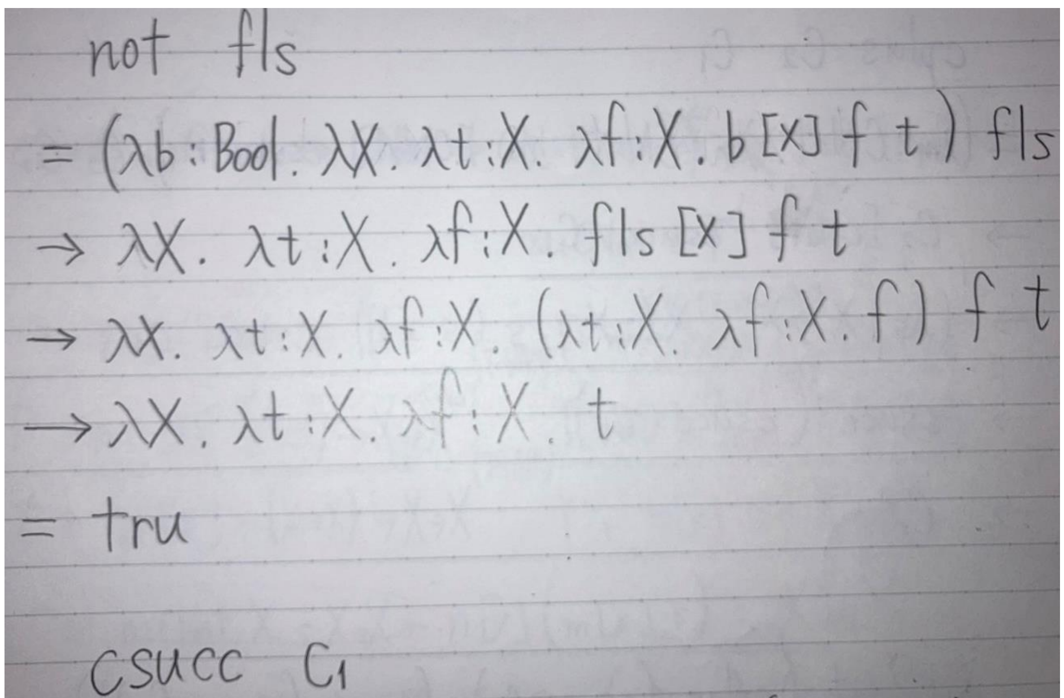
Church Encodings

[System F]

tru : CBool = λX. λt : X. λf : X. t

fls : CBool = λX. λt : X. λf : X. f

not = λb : Cbool. λX. λt : X. λf : X. b [X] f t



23.4.5 Exercise [Recommended]

23.4 例

Church Encodings

[$\lambda_{\to}$] $c_0 = \lambda s.\ \lambda z.\ z$

$c_1 = \lambda s.\ \lambda z.\ s\ z$

$c_2 = \lambda s.\ \lambda z.\ s\ (s\ z)$

[System F]

$CNat : \forall X.\ (X \to X) \to X \to X$

$c_0 : CNat = \lambda X.\ \ \lambda s : X \to X.\ \ \lambda z : X.\ \ z$

$c_1 : CNat = \lambda X.\ \ \lambda s : X \to X.\ \ \lambda z : X.\ \ s\ z$

$c_2 : CNat = \lambda X.\ \ \lambda s : X \to X.\ \ \lambda z : X.\ \ s\ (s\ z)$

23.4 例

Church Encodings

[System F]

$csucc = \lambda n : CNat.\ \ \lambda X.\ \ \lambda s : X \to X.\ \ \lambda z : X.\ \ s\ (n\ [X]\ s\ z)$

$\#\ csucc : CNat \to CNat$

$cplus = \lambda m : CNat.\ \ \lambda n : CNat.\ \ m\ [CNat]\ csucc\ n$

$\#\ cplus : CNat \to CNat \to CNat$

$cnat2nat = \lambda m : CNat.\ \ m\ [Nat]\ (\lambda x : Nat.\ \ succ(x))\ 0$

$\#\ cnat2nat : CNat \to Nat$

$cnat2nat\ (cplus\ (csucc\ c_0)\ (csucc\ (csucc\ c_0)))$

$\#\ 3 : Nat$

23.4 例

Church Encodings

[System F]

$\mathrm{csucc} = \lambda n : \mathrm{CNat}.\ \lambda X.\ \lambda s : X \to X.\ \lambda z : X.\ s\ (n\ [X]\ s\ z)$

$\#\ \mathrm{csucc} : \mathrm{CNat} \to \mathrm{CNat}$

$\mathrm{csucc}\ C_1$

$= (\lambda n : \mathrm{CNat}.\ \lambda X.\ \lambda s : X \to X.\ \lambda z : X.\ s\ (n\ [X]\ s\ z))\ C_1$

$\to \lambda X.\ \lambda s : X \to X.\ \lambda z : X.\ s\ (C_1\ [X]\ s\ z)$

$\to \lambda X.\ \lambda s : X \to X.\ \lambda z : X.\ s\ ((\lambda s : X \to X.\ \lambda z : X.\ s\ z)\ s\ z)$

$\to \lambda X.\ \lambda s : X \to X.\ \lambda z : X.\ s\ (s\ z)$

23.4 例

Church Encodings

[System F]

cplus = λm : CNat. λn : CNat. m [CNat] csucc n

 # cplus : CNat → CNat → CNat

$$
\begin{aligned}
& \mathrm{cplus}\ C_2\ C_1 \\
&= (\lambda m : \mathrm{CNat}.\ \lambda n : \mathrm{CNat}.\ m\ [\mathrm{CNat}]\ \mathrm{csucc}\ n)\ C_2\ C_1 \\
&\rightarrow C_2\ [\mathrm{CNat}]\ \mathrm{csucc}\ C_1 \\
&\rightarrow (\lambda s : X \rightarrow X.\ \lambda z : X.\ s\ (s\ z))\ \mathrm{csucc}\ C_1 \\
&\rightarrow \mathrm{csucc}\ (\mathrm{csucc}\ C_1) \\
&\rightarrow C_3
\end{aligned}
$$

23.4 例

Church Encodings

[System F]

cnat2nat = λm : CNat. m [Nat] (λx : Nat. succ(x)) 0

 # cnat2nat : CNat → Nat

$$
\begin{aligned}
& \mathrm{cnat2nat}\ (\mathrm{cplus}\ (\mathrm{csucc}\ c_0)\ (\mathrm{csucc}\ (\mathrm{csucc}\ c_0))) \\
\to\ & \mathrm{cnat2nat}\ (\mathrm{cplus}\ c_1\ c_2) \\
\to\ & \mathrm{cnat2nat}\ c_3 \\
=\ & (\lambda m : \mathrm{CNat}.\ m\ [\mathrm{Nat}]\ (\lambda x : \mathrm{Nat}.\ \mathrm{succ}(x))\ 0)\ c_3 \\
\to\ & c_3\ [\mathrm{Nat}]\ (\lambda x : \mathrm{Nat}.\ \mathrm{succ}(x))\ 0 \\
\to\ & (\lambda s : \mathrm{Nat} \to \mathrm{Nat}.\ \lambda z : \mathrm{Nat}.\ s\ (s\ (s\ z)))\ (\lambda x : \mathrm{Nat}.\ \mathrm{succ}(x))\ 0 \\
\to\ & (\lambda x : \mathrm{Nat}.\ \mathrm{succ}(x))\ (\ \cdots\ (\lambda x : \mathrm{Nat}.\ \mathrm{succ}(x))\ 0) \\
\to\ & 3
\end{aligned}
$$

23.4.6 Exercise [Recommended]

23.4 例

23.4.7 Exercise

ctimes = λm : CNat. λn : CNat. λX. λs : X → X. n [X] (m [X] s)

cexp = λm : CNat. λn : CNat. λX. n [X → X] (m [X])

これらの項が,

 # ctimes : CNat → CNat → CNat

 # cexp : CNat → CNat → CNat

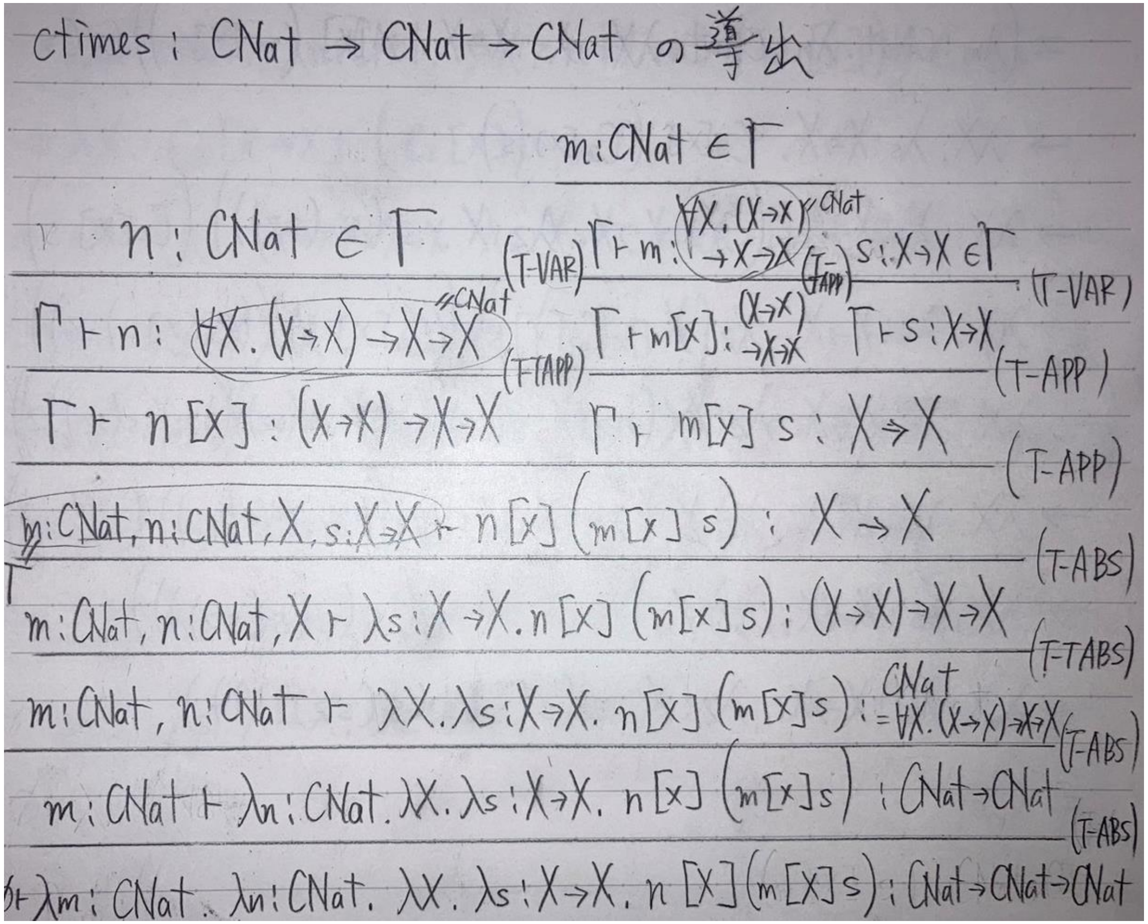
の型を持つことを検証し, これらが乗算関数・べき乗関数になっていることを informal に確認する.

23.4 例

23.4.7 Exercise

ctimes = λm : CNat. λn : CNat. λX. λs : X → X. n [X] (m [X] s)

 # ctimes : CNat → CNat → CNat



23.4 例

23.4.7 Exercise

ctimes = λm : CNat. λn : CNat. λX. λs : X → X. n [X] (m [X] s)

 # ctimes : CNat → CNat → CNat

$$\text{ctimes}\ C_2\ C_3$$
$$= (\lambda m{:}\text{CNat}.\ \lambda n{:}\text{CNat}.\ \lambda X.\ \lambda s{:}X\to X.\ n\ [X]\ (m\ [X]\ s))\ C_2\ C_3$$
$$\to \lambda X.\ \lambda s{:}X\to X.\ C_3\ [X]\ (C_2\ [X]\ s)$$
$$\to \lambda X.\ \lambda s{:}X\to X.\ (\lambda s{:}X\to X.\ \lambda z{:}X.\ s\ (s\ (s\ z)))\ (C_2\ [X]\ s)$$
$$\to \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\ (C_2\ [X]\ s)\ ((C_2\ [X]\ s)\ ((C_2\ [X]\ s)\ z))$$
$$\to \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\ (\lambda z{:}X.\ s\ (s\ z))\ ((\lambda z{:}X.\ s\ (s\ z))\ ((\lambda z{:}X.\ s\ (s\ z))\ z))$$
$$\to \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\ (\lambda z{:}X.\ s\ (s\ z))\ ((\lambda z{:}X.\ s\ (s\ z))\ (s\ (s\ z)))$$
$$\to \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\ (\lambda z{:}X.\ s\ (s\ z))\ (s\ (s\ (s\ (s\ z))))$$
$$\to \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\ s\ (s\ (s\ (s\ (s\ (s\ z)))))$$
$$= C_6$$

23.4 例

23.4.7 Exercise

cexp = λm : CNat. λn : CNat. λX. n [X → X] (m [X])

 # cexp : CNat → CNat → CNat

導出木省略．画像以降の簡約列を書けず…

$\mathrm{cexp}\ C_2\ C_2$

$= (\lambda m{:}\mathrm{CNat}.\ \lambda n{:}\mathrm{CNat}.\ \lambda X.\ n\,[X\to X]\ (m\,[X]))\ C_2\ C_2$

$\to \lambda X.\ C_2\,[X\to X]\ (C_2\,[X])$

$= \lambda X.\ (\lambda s{:}(X\to X)\to X\to X.\ \lambda z{:}X\to X.\ s\ (s\ z))\ (C_2\,[X])$

$\to \lambda X.\ \lambda z{:}X\to X.\ C_2\,[X]\ (C_2\,[X]\ z)$

$\to \lambda X.\ \lambda z{:}X\to X.\ (\lambda s{:}X\to X.\ \lambda z{:}X.\ s\ (s\ z))\ (C_2\,[X]\ z)$

$\to \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\ s\ (s\ (C_2\,[X]\ z))$

$\to \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\ s\ (s\ ((\lambda s{:}X\to X.\ \lambda z{:}X.\ s\ (s z))\ z)$

$\to \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.$

23.4 例

23.4.8 Exercise [Recommended]

23.4.9 Exercise [Recommended]

23.4.10 Exercise

k = λx. λy. x および **i = λx. x** とすると,

vpred = λn. λs. λz. n (λp. λq. q (p s)) (k z) i

は, untyped Church numeral における predecessor である.

適切な型抽象・型適用を追加することにより, これが *System F* により型付けされることを示し, これがどのように機能するか確認する.

23.4 例

23.4.10 Exercise

.

$k = \lambda x.\ \lambda y.\ x$, $i = \lambda x.\ x$ のとき

$$vpred = \lambda n.\ \lambda s.\ \lambda z.\ n\ (\lambda p.\ \lambda q.\ q\ (p\ s))\ (k\ z)\ i$$

System F では…

$$vpred = \lambda n{:}CNat.\ \lambda X.\ \lambda s{:}X \to X.\ \lambda z{:}X.$$
$$(n\ [(X \to X) \to X]$$
$$(\lambda p{:}(X \to X) \to X.\ \lambda q{:}X \to X.\ q\ (p\ s))$$
$$(\lambda x{:}X \to X.\ z))$$
$$(\lambda x{:}X.\ x)$$

23.4.10 Exercise

$$
\begin{aligned}
\text{Vpred } C_1 \to\ & \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\\
& (C_1\,[(X\to X)\to X]\\
& \quad(\lambda p{:}(X\to X)\to X.\ \lambda q{:}X\to X.\ q\,(p\,s))\\
& \quad(\lambda x{:}X\to X.\ z))\\
& \quad(\lambda x{:}X.\ x)
\end{aligned}
$$

23.4 例

23.4.10 Exercise

$$C_1\,[\,(X \to X) \to X\,]$$

$$\to \lambda s:(X\to X)\to X \to (X\to X)\to X.\ \lambda z:(X\to X)\to X.\ s\ z$$

より

$$C_1[(X\to X)\to X]\,(\lambda p:(X\to X)\to X.\ \lambda q:X\to X.\ q\,(p\ s))(\lambda x:X\to X.\ z)$$

$$\to \lambda p:(X\to X)\to X.\ \lambda q:X\to X.\ (q\ (p\ s))\ (\lambda x:X\to X.\ z)$$

であり,

23.4.10 Exercise

$$
\begin{aligned}
\mathrm{upred}\ C_1 \to^{*}\ & \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\\
& (\lambda p{:}(X\to X)\to X.\ \lambda q{:}X\to X.\ (q\ (p\ s))\ (\lambda x{:}X\to X.\ z))\ (\lambda x{:}X.\ x)\\
\to\ & \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\\
& (\lambda x{:}X.\ x)\ ((\lambda x{:}X\to X.\ z)\ s)\\
\to\ & \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\\
& ((\lambda x{:}X\to X.\ z)\ s)\\
\to\ & \lambda X.\ \lambda s{:}X\to X.\ \lambda z{:}X.\ z\\
=\ & C_0
\end{aligned}
$$

23.4 例

Encoding Lists

List X = ∀R. (X → R → R) → R → R

nil = λX. (λR. λc : X → R → R. λn : R. n) as List X

\# nil : ∀X. List X

cons = λX. λhd : X. λtl : List X.

(λR. λc : X → R → R. λn : R. c hd (tl [R] c n)) as List X

\# cons : ∀X. X → List X → List X

isnil = λX. λl : List X. l [Bool] (λhd : X. λtl : Bool. false) true

\# isnil : ∀X. List X → Bool

23.4 例

Encoding Lists

head の定義は少しややこしい

empty list を与えたときの挙動をどうするか？

型Xを与えると Unit → X の関数を生成するdiverge

diverge = λX. λ_ : Unit. fix (λx : X. x)

 # diverge : ∀X. Unit → X

diverge [X] unit を head nil の結果として使用する.

head = λX. λl : List X. l [X] (λhd : X. λtl : X. hd) (diverge [X] unit)

 # head : ∀X. List X → X

これは非空リストを受け取っても無限ループする.

23.4 例

Encoding Lists

l が nil のとき diverge [X] unit (発散)
そうでないときは unit を受け取って先頭要素を返すように定義を修正.

head =
λX. λl : List X.
(l [Unit → X] (λhd : X. λtl : Unit → X. hd) (diverge [X]))
unit
 # head : ∀X. List X → X

23.4 例

Encoding Lists

tail 関数のためにPair X Y 型とpairに対する操作を導入.

Pair X Y = ∀R. (X→Y→R)→R

pair : ∀X. ∀Y. X→Y→Pair X Y

fst : ∀X. ∀Y. Pair X Y→X

snd : ∀X. ∀Y. Pair X Y→X

23.4 例

Encoding Lists

tail = λX. λl : List X.
(fst [List X] [List X] (
l [Pair (List X) (List X)]
(λhd : X. λtl : Pair (List X) (List X).
pair [List X] [List X]
(snd [List X] [List X] tl)
(cons [X] hd (snd [List X] [List X] tl)))
(pair [List X] [List X] (nil [X] (nil [X])))))

 # tail : ∀X. List X → List X

23.4.11 Exercise

厳密に言えば、Encoding Lists はpure System Fでは表現されていない。なぜなら、**head** が空のリストに適用されたときの返り値を構築するために **fix** 演算子を使ったからである。

空リストを受け取った時に，発散の代わりに返される別のパラメータを引数とする **head** 関数を書け。

head = λX. λdefault : X. λl : List X.
l [X] (λhd : X. λtl : X. hd) default;;

23.4.12 Exercise [Recommended]

END.