ラムダ計算の復習を書いておく。Church 流（？）でコンストラクタみたいなやつをいちいち書く。少なくとも正確ではないし、多分こう考えたら普通の奴と同じになるだろう、みたいなのがいっぱいある。$\langle hoge; t \rangle$ とかいたら、$\langle hoge \rangle$ の元を t のように表すということ。

- 型なしラムダ計算
- 単純型付きラムダ計算
- 単純型付きラムダ計算 + 自然数 + 真偽値
- 単純型付きラムダ計算 + 積 + 和 + Unit + Void

■型なしラムダ計算　ラムダ項の定義としては

項の定義

$\langle term\ ;\ t\rangle ::= \langle variable\ ;\ x\rangle$
　| ‘Fun’ $\langle variable\rangle\ \langle term\rangle$
　| ‘App’ $\langle term\rangle\ \langle term\rangle$

これについては型理論はない。ラムダ項にどういうものが定義されていたかと言うと、

- 自由変数、 FV := **項の中にある束縛されていない変数** を返す関数
- 単純な代入 $(M\{x \leftarrow N\})$:= **項** M **の中に自由変数としてあらわれる変数** x **を** N **に置き換える**
- α 同値：項の間の関係 := 次で生成される同値関係
  - $\mathtt{Fun}\,x\,M =_\alpha \mathtt{Fun}\,y\,(M\{x \leftarrow y\})$ ただし M は自由変数に y をもたない。
- 簡約、 β 変換：項の間の関係 := **なんかいい感じの奴**
- 評価戦略：上の β 変換なる関係を実際に表現する、項から項への関数
- 値、正規形：ある条件を満たすラムダ項

これらに対して合流性やらが成り立つのだった。項の間の同値関係 $t_1 \equiv t_2$ を次の規則から生成する。

ラムダ項の等式

$$\frac{x_2 \notin \mathrm{FV}(t_2)}{\mathtt{Fun}\,x_1\,t_1 \equiv \mathtt{Fun}\,x_2\,(t_1\{x_1 \leftarrow t_2\})}\ \text{alpha}$$

$$\frac{x \notin \mathrm{FV}(t_2)}{\mathtt{App}\,(\mathtt{Fun}\,x\,t_1)\,t_2 \equiv t_2\{x \leftarrow t_2\}}\ \text{computation}$$

$$\frac{t_1 \equiv t_2}{\mathtt{Fun}\,x\,t_1 \equiv \mathtt{Fun}\,x\,t_2}\ \text{conversion-fun}$$

$$\frac{t_1 \equiv t_3}{\mathtt{App}\,t_1\,t_2 \equiv \mathtt{App}\,t_3\,t_2}\ \text{conversion-app-1}$$

$$\frac{t_2 \equiv t_3}{\mathtt{App}\,t_1\,t_2 \equiv \mathtt{App}\,t_1\,t_3}\ \text{conversion-app-2}$$

α, β 変換はこの同値と非常にうまくいくことが（定理としてであって定義ではなく）わかる。注意として、

- conversion-fun は関数型言語で作るときには人によっては使わないかも。スタックマシンへのコンパイルの場合は値としての関数は変に reduction せず写したり、 thunk と呼んで抽象化されたものはそれ以上変形しないことがある。
- β 変換は関係であって関数ではない、具体的に計算する方法を定めることが評価戦略を定めることにあたり、大体どの評価方法をつかっても同じものが返ってくる（いわゆる合流性）。

また、これに

$$\frac{}{t \equiv \mathtt{Fun}\,x\,(\mathtt{App}\,t\,x)}\ \text{computation-eta}$$

を付け加えることもあるが、外延的ラムダ計算という別の体系になる。

■単純型付きラムダ計算　ラムダ計算に型をつける。このとき、型に加えて新しくコンテキストというものが定義される。

項、型、コンテキストの定義

$\langle \textit{type ; } T \rangle ::= \langle \textit{type-variable ; } X \rangle$
　| ‘`Arr`’ $\langle \textit{type} \rangle \langle \textit{type} \rangle$

$\langle \textit{term ; } t \rangle ::= \langle \textit{term-variable ; } x \rangle$
　| ‘`Fun`’ $\langle \textit{variable} \rangle \langle \textit{type} \rangle \langle \textit{term} \rangle$
　| ‘`App`’ $\langle \textit{term} \rangle \langle \textit{term} \rangle$

$\langle \textit{context ; } \Gamma \rangle ::= \text{empty} \mid \langle \textit{context} \rangle , \langle \textit{term-variable} \rangle : \langle \textit{type} \rangle$

項に対しては相変わらず型なしラムダ計算と同じようなものが定義された。一方で型付けと言う新しい要素が増えた。型の間に自然に定義される等しさが当然ある。型付けとは、 Context (Γ) と Term (t) と Type (T) の間の関係 $\Gamma \vdash t : T$ であって次で生成されるもの。

型付け規則

$$\frac{}{\Gamma, x : T \vdash x : T}\ \text{variable}$$

$$\frac{\Gamma \vdash x_2 : T_2}{\Gamma, x_1 : T_1 \vdash x_2 : T_2}\ \text{weakning}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \texttt{Fun}\, x\, T_1\, t : \texttt{Arr}\, T_1\, T_2}\ \text{introduction}$$

$$\frac{\Gamma \vdash t_1 : \texttt{Arr}\, T_1\, T_2 \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash \texttt{App}\, t_1\, t_2 : T_2}\ \text{elimination}$$

ラムダ項の間の等式や簡約は大体型を無視した感じで定義されるため、型付けとは別に定義することができる。後でこの二つが強く関係してくるような体系がでてくる（定義時にお互いに依存するなど）型理論で成り立つかどうかを調べるのはだいたい以下のような感じ。有限ステップで停止する形で判定できることを決定できると表現する（あまりよくない表現で申し訳ない)。

- $\Gamma \vdash t : A$ であることの証明を与えられたとき、正しいか決定できる。
- $\Gamma \vdash t : A$ であるかどうか決定できる...type check decidability
- Γ と t に対して $\Gamma \vdash t : A$ なる A が存在するか決定できる...type inference
- Γ と A に対して $\Gamma \vdash t : A$ なる t が存在するか計算できる...type inhabitants
- $\Gamma \vdash t : A$ かつ $\Gamma \vdash t : B$ なら $A = B$ である...type uniqueness
- $\Gamma \vdash t : A$ かつ $t \to_\beta t'$ なら $\Gamma \vdash t' : A$...subject reduction
- $\vdash t : A$ なら t の簡約が停止し値になる

■単純型付きラムダ計算の拡張１　型付きラムダ計算をプログラムっぽく拡張する。拡張の方向性はいろいろあり、自然数と真偽値だけ追加する場合でもゆれがある。特に、

項、型、コンテキストの定義

$\langle type\ ;\ T\rangle ::= \ldots$
　| ‘`Nat`’
　| ‘`Bool`’

$\langle term\ ;\ t\rangle ::= \ldots$
　| ‘`O`’ | ‘`S`’ $\langle term\rangle$
　| ‘`match`’ $\langle term\rangle$ ‘`with`’ $\langle term\rangle$ ‘`or`’ $\langle variable\rangle$ $\langle term\rangle$
　| ‘`true`’ | ‘`false`’
　| ‘`if`’ $\langle term\rangle$ ‘`then`’ $\langle term\rangle$ ‘`else`’ $\langle term\rangle$

$\langle context\ ;\ \Gamma\ \rangle ::=$ empty | $\langle context\rangle$, $\langle term\text{-}variable\rangle : \langle type\rangle$

`Nat` という型は帰納的に定義され、コンストラクタが `O` と `S` でコンストラクタの除去のための項が `ifzero` である。

型付け規則

...

$$\frac{}{\vdash \mathtt{O} : \mathtt{Nat}}\ \text{nat-intro-O}$$

$$\frac{\Gamma \vdash M : \mathtt{Nat}}{\Gamma \vdash \mathtt{S}\, M : \mathtt{Nat}}\ \text{nat-intro-S}$$

nat-intro-

ラムダ項の間の等式や簡約は大体型を無視した感じで定義されるため、型付けとは別に定義することができる。後でこの二つが強く関係してくるような体系がでてくる（定義時にお互いに依存するなど）型理論で成り立つかどうかを調べるのはだいたい以下のような感じ。有限ステップで停止する形で判定できることを決定できると表現する（あまりよくない表現で申し訳ない）。

- $\Gamma \vdash t : A$ であることの証明を与えられたとき、正しいか決定できる。
- $\Gamma \vdash t : A$ であるかどうか決定できる...type check decidability
- Γ と t に対して $\Gamma \vdash t : A$ なる A が存在するか決定できる...type inference
- Γ と A に対して $\Gamma \vdash t : A$ なる t が存在するか計算できる...type inhabitants
- $\Gamma \vdash t : A$ かつ $\Gamma \vdash t : B$ なら $A = B$ である...type uniqueness
- $\Gamma \vdash t : A$ かつ $t \to_\beta t'$ なら $\Gamma \vdash t' : A$...subject reduction
- $\vdash t : A$ なら t の簡約が停止し値になる

■単純型付きラムダ計算の拡張達　あとで書く。積、和、Unit 、Void を足すことで Curry-Howard がいい感じに。自然数、真偽値、リストを足すことでプログラムっぽくなる。全部 Inductive に作れる。Curry-Howard 対応で見ておもしろくなるものをラムダ計算に付け足す。

項、型、コンテキストの定義

$\langle type \rangle ::= \langle type\text{-}variable \rangle$
| ‘`Arr`’ $\langle type \rangle$ $\langle type \rangle$
| ‘`Prod`’ $\langle type \rangle$ $\langle type \rangle$
| ‘`Sum`’ $\langle type \rangle$ $\langle type \rangle$
| ‘`Unit`’
| ‘`Void`’

$\langle term \rangle ::= \langle term\text{-}variable \rangle$
| ‘`Fun`’ $\langle variable \rangle$ $\langle type \rangle$ $\langle term \rangle$ | ‘`App`’ $\langle term \rangle$ $\langle term \rangle$
| ‘`Pair`’ $\langle term \rangle$ $\langle term \rangle$ | ‘`Pjl`’ $\langle term \rangle$ | ‘`Pjr`’ $\langle term \rangle$
| ‘`Inl`’ $\langle term \rangle$ | ‘`InE`’ $\langle term \rangle$ | ‘`Case`’ $\langle term \rangle$ $\langle variable \rangle$ $\langle term \rangle$ $\langle variable \rangle$ $\langle term \rangle$
| ‘`T`’

$\langle context \rangle ::=$ empty | $\langle context \rangle$, $\langle term\text{-}variable \rangle$: $\langle type \rangle$

型付け規則

$$\frac{}{\Gamma, x : T \vdash x : T}\ \text{variable} \qquad \frac{\Gamma \vdash x_2 : T_2}{\Gamma, x_1 : T_1 \vdash x_2 : T_2}\ \text{weakning}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \mathtt{Fun}\, x\, T_1\, t : \mathtt{Arr}\, T_1\, T_2}\ \text{Arr-intro} \qquad \frac{\Gamma \vdash t_1 : \mathtt{Arr}\, T_1\, T_2 \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash \mathtt{App}\, t_1\, t_2 : T_2}\ \text{Arr-elim}$$

$$\frac{\Gamma \vdash t_1 : T_1 \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \mathtt{Pair}\, t_1\, t_1 : \mathtt{Prod}\, T_1\, T_2}\ \text{Prod-intro} \qquad \frac{\Gamma \vdash t : \mathtt{Prod}\, T_1\, T_2}{\Gamma \vdash \mathtt{Pjl}\, t : T_1}\ \text{Prod-elim1} \qquad \frac{\Gamma \vdash t : \mathtt{Prod}\, T_1\, T_2}{\Gamma \vdash \mathtt{Pjr}\, t : T_2}\ \text{Prod-elim2}$$

$$\frac{\Gamma \vdash x : T_1}{\Gamma \vdash \mathtt{Inl}\, t : \mathtt{Sum}\, T_1\, T_2}\ \text{Sum-intro1} \qquad \frac{\Gamma \vdash x : T_2}{\Gamma \vdash \mathtt{Inr}\, t : \mathtt{Sum}\, T_1\, T_2}\ \text{Sum-intro2}$$

$$\frac{\Gamma \vdash t : \mathtt{Sum}\, T_1\, T_2 \quad \Gamma, x_1 : T_1 \vdash t_1 : T \quad \Gamma, x_2 : T_2 \vdash t_2 : T}{\Gamma \vdash \mathtt{Case}\, t\, x_1\, t_1\, x_2\, t_2 : T}\ \text{Sum-elim}$$

$$\frac{}{\Gamma \vdash \mathtt{T} : \mathtt{Unit}}\ \text{Unit-intro} \qquad \frac{}{\Gamma \vdash}\ \text{Unit-elim}$$

ラムダ項の等式

$$\frac{}{\mathtt{Fun}\, x_1\, t_1 \equiv \mathtt{Fun}\, x_2\, (t_1\{x_1 \leftarrow t_2\})}\ \text{Arr-alpha} \qquad \frac{}{\mathtt{App}\, (\mathtt{Fun}\, x\, t_1) t_2 \equiv t_2\{x \leftarrow t_2\}}\ \text{Arr-comp}$$

$$\frac{t_1 \equiv t_2}{\mathtt{Fun}\, x\, t_1 \equiv \mathtt{Fun}\, x\, t_2}\ \text{Arr-conv} \qquad \frac{t_1 \equiv t_3}{\mathtt{App}\, t_1\, t_2 \equiv \mathtt{App}\, t_3\, t_2}\ \text{Arr-conv} \qquad \frac{t_2 \equiv t_3}{\mathtt{App}\, t_1\, t_2 \equiv \mathtt{App}\, t_1\, t_3}\ \text{Arr-conv}$$

$$\frac{}{\mathtt{Pj1}\, (\mathtt{Pair}\, t_1 t_2) \equiv t_1}\ \text{Prod-comp1} \qquad \frac{}{\mathtt{Pj2}\, (\mathtt{Pair}\, t_1 t_2) \equiv t_2}\ \text{Prod-comp2}$$

$$\frac{}{\mathtt{Case}\, (\mathtt{Inl}\, t)\, x_1\, t_1\, x_2\, t_2 \equiv t_1\{x_1 \leftarrow t\}}\ \text{Sum-comp1} \qquad \frac{}{\mathtt{Case}\, (\mathtt{Inr}\, t)\, x_1\, t_1\, x_2\, t_2 \equiv t_2\{x_2 \leftarrow t\}}\ \text{Sum-comp1}$$

あと cov がいろいろ

この場合、 Coq と比較すると Product Type の induction が対応しないように見えるため、eliminater に対応するものに変えることが考えられる。こっちの方が match に近くて書きやすい。

項、型、コンテキストの定義

⟨*type*⟩ ::= ⟨*type-variable*⟩
| ‘`Arr`’ ⟨*type*⟩ ⟨*type*⟩
| ‘`Prod`’ ⟨*type*⟩ ⟨*type*⟩
| ‘`Sum`’ ⟨*type*⟩ ⟨*type*⟩
| ‘`Unit`’
| ‘`Void`’

⟨*term*⟩ ::= ⟨*term-variable*⟩
| ‘`Fun`’ ⟨*variable*⟩ ⟨*type*⟩ ⟨*term*⟩
| ‘`App`’ ⟨*term*⟩ ⟨*term*⟩
| ‘`Prod-c`’ ⟨*term*⟩ ⟨*term*⟩
| ‘`Prod-e`’ ⟨*term*⟩
| ‘`Sum-c1`’ ⟨*term*⟩
| ‘`Sum-c2`’ ⟨*term*⟩
| ‘`Sum-e`’ ⟨*term*⟩
| ‘`Unit-c`’
|

⟨*context-snippet*⟩ ::= ⟨*term-variable*⟩ : ⟨*type*⟩

⟨*context*⟩ ::= empty | ⟨*context*⟩ , ⟨*context-snippet*⟩

型付け規則

$$\frac{}{\Gamma, x : T \vdash x : T}\ \text{variable}$$

$$\frac{\Gamma \vdash x_2 : T_2}{\Gamma, x_1 : T_1 \vdash x_2 : T_2}\ \text{weakning}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \texttt{Fun}\, x\, T_1\, t : \texttt{Arr}\, T_1\, T_2}\ \text{Arr-introduction}$$

$$\frac{\Gamma \vdash t_1 : \texttt{Arr}\, T_1\, T_2 \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash \texttt{App}\, t_1\, t_2 : T_2}\ \text{Arr-elimination}$$