型なしラムダ計算のまとめ。これは形式化の一つである。このラムダ計算の形式化では変数を必要としているがこれを使わない方法もあり、例としては Debruijn などがある。また、単純な代入だけで α 同値をすますために新しい変数をとる関数を仮定しているが、捕縛回避代入と自由変数と束縛変数を決定できるので、この仮定を使わないこともできるはず。

項の定義

`<term>`	::=	t , M
	| `<variable>`	x
	| `Fun <variable> <term>`	
	| `App <term> <term>`	

右の奴はメタ変数で表すときにどう書くか

`<variable>` は人が勝手に決めてよいが、等しいかどうか判定できることと、新しい変数を用意できることを要求する。具体的には `isEquiv` : `variable` → `variable` → `Bool` があって等しさの判定に使えること（これを $=?$ で書いて場合分けに使う)、 `freshOf` : `list<variable>` → `variable` があって `list<variable>` に含まれないものを返せることが必要。これがあれば x : `variable` と L : `list<variable>` に対してこれに含まれているか判定できる。$x \in L$ と書いてしまう。

ラムダ項に対して以下のものが定義されていた。定理の記号は [!] とし、計算は停止しないもの、決定は停止するものに使う。(形式化されているっぽく書いているがこの言語に特に意味論はない。)

まず変数と代入について。

* 自由変数 (`freevalueOf`) : `term` → `list<variable>`
  := 変数が自由かどうか
* 束縛変数 (`boundedvalueOf`) : `term` → `list<variable>`
  := 変数が束縛されるかどうか

- 変数 (`valueOf`) : `term` → `list<variable>`
  := 変数が出現するかどうか
- 単純な代入 (`simpleSubst`) : `term` → `variable` → `term`
  := 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
  with notation $M\{x \leftarrow N\}$ as `simpleSubst` $M\,x\,N$

！変数の単純な代入は項の形（変数の名前だけ見ないで抽象と適用がどう表れているか）を変えず、特に項の大きさを変えない

- 捕縛を回避した代入 (`subst`) : `term` → `variable` → `term`
  := 項 M の中に自由変数としてあらわれる変数 x を N に置き換えるが、 N が M の他の自由変数を束縛しないようにするもの
  with notation $M[x \leftarrow N]$ as `subst` $M\,x\,N$
  これの構成は非自明なものになりがち。

次に α 同値について

- α 同値関係 (`isAlphaEquiv`) ： `term` → `term` → `Prop`

:= 次で生成される同値関係（閉包をとっていることに注意、この定義自体は small step な感じ）

$$\frac{(x_2 \notin \texttt{valueOf}\, M)}{\texttt{Fun}\, x_1\, M =_\alpha \texttt{Fun}\, x_2\, (M\{x_1 \leftarrow x_2\})}\ \text{alpha}$$

$$\frac{(x_1 =? x_2)}{x_1 =_\alpha x_2}\ \text{Var conversion}$$

$$\frac{M_1 =_\alpha M_2}{\texttt{Fun}\, x\, M_1 =_\alpha \texttt{Fun}\, x\, M_2}\ \text{Fun conversion}$$

$$\frac{M_1 =_\alpha M_1}{\texttt{App}\, M_1\, N =_\alpha \texttt{App}\, M_1\, N}\ \text{App conversion 1}$$

$$\frac{N_1 =_\alpha N_2}{\texttt{App}\, M\, N_1 =_\alpha \texttt{App}\, M\, N_2}\ \text{App conversion 2}$$

with notation $M =_\alpha N$ as `isAlphaEquiv` $M\, N$

- α 同値決定 (`isAlphaEquiv?`) : `term -> term -> Bool`
  := 次のように帰納的に定義
  M_1 と M_2 をとりこの構造に着目し以下のように場合分けする。
  - x_1 と x_2 のとき $x_1 =? x_2$ とする。
  - `Fun` $x_1\, M_1$ と `Fun` $x_2\, M_2$ のとき、`c` を `freshOf` $((\texttt{valueOf}\, M_1) + +(\texttt{valueOf}\, M_2))$ とおき $M_1\{x_1 \leftarrow c\} =_\alpha M_2\{x_2 \leftarrow c\}$ とする。
  - `App` $M_1\, M_2$ と `App` $N_1\, N_2$ のとき $M_1 =_\alpha N_1$ かつ $M_2 =_\alpha N_2$ とする。
  - それ以外は `false` とする。

  with notation $M_1 =_\alpha ? M_2$ as `isAlphaEquiv?` $M_1\, M_2$

! α 同値決定は停止し、 α 同値関係は α 同値決定で計算できる。

- α 同値に関する束縛変数の一律付け替え : `term` → `list<variable>` → `term`
  := 名前の通り、束縛変数を一律にリストに含まれないように付け替える

† これが必要となるのは捕縛回避代入を定義する際に「項の長さに関する帰納法を用いて再帰的に定義する方法」と「一度束縛変数を全ていい感じに付け替えることで行う方法」の二種類があり、後者では必要になるため。何て呼べばいいかわからなかった。

次に β 変換について。

- β 変換関係 (`isBetaConversion`) : `term` → `term` → `Prop`
  := 次で生成される関係

$$\frac{}{\texttt{App}\, (\texttt{Fun}\, x\, M)\, N \to_\beta M[x \leftarrow N]}\ \text{beta}$$

$$\frac{M \to_\beta N}{\texttt{Fun}\, x\, M \to_\beta \texttt{Fun}\, x\, N}\ \text{Fun conversion}$$

$$\frac{M_1 \to_\beta M_1}{\texttt{App}\, M_1\, N \to_\beta \texttt{App}\, M_1\, N}\ \text{App conversion 1}$$

$$\frac{N_1 \to_\beta N_2}{\texttt{App}\, M\, N_1 \to_\beta \texttt{App}\, M\, N_2}\ \text{App conversion 2}$$

with notation $M \to_\beta N$ as `isBetaConversion` $M\, N$

- β 簡約関係 (`isBetaReduce`) : `term` → `term` → `Prop`
  := β 変換関係 + α 同値関係の推移閉包

† 捕縛回避代入を使わずに単純な代入を使って β 同値を定義するときは、α 同値で変形できるようにしておく必要がある。その場合は α 同値の変形を一回と考えると上で定義した β 変換とは全然異なってしまう（正規形がない）など不具合があるので注意。

- 正規形 (`isNormal`) : `term` → `Prop`
  := β 変換関係においてこれ以上変換できないとき
- 正規形決定 (`isNormal?`) : `term` → `Bool`
  := 次のように帰納的に定義
  - x のとき true
  - $\texttt{Fun}\, x\, M$ のとき M に帰着
  - $\texttt{App}\,(\texttt{Fun}\, x\, M)\, N$ のとき false
  - $\texttt{App}\, M\, N$ のとき M と N に帰着

! β 簡約関係は合流性を持つ。

! 正規形は正規形決定で決定できる。

! 項の簡約の結果として得られる正規形は α 同値を除き一意である。

- β 変換 (`betaConversion`) : `term` → `option<term>`
  := 次のように帰納的に定義
  - x のとき `None`
  - $\texttt{Fun}\, x\, M$ のとき $\texttt{Fun}\, x\, (\texttt{conversion}\, M)$
  - $\texttt{App}\,(\texttt{Fun}\, x\, M)\, N$ のとき $M[x \leftarrow N]$
  - $\texttt{App}\, M\, N$ のとき
    - M が正規形なら $\texttt{App}\, M\, (\texttt{conversion}\, N)$
    - そうでなければ $\texttt{App}\, (\texttt{conversion}\, M)\, N$

† このような具体的な関数を作るのは評価戦略を決定することに対応しているため、人によっては β 変換の定義は異なる。ここでは最左最外の評価戦略をとった（そのために成り立つ性質もある）。また、`option` に値を持っているため本当はこれでは定義できていないが、そこはなあなあで。なぜ `option` かというと β 変換と合わせるためであるが、正規形が決定できることを考えると `term` → `term` としてもよい。

! β 変換は β 変換関係や簡約関係と compatible

- β 簡約、正規形の計算 (`normalize`) : `term` → `term`
  := 以下のように再帰的に定義
  - 正規形ならそのまま
  - 正規形でないなら `conversion` したものを `normalize`

! 正規形計算は（停止しないことがあるが、）停止すれば正規形が返ってくる。

! 項が簡約による正規形を持てば正規形計算は停止し正規形が返ってくる（最左最外簡約であることから）。

- β 同値 : `term` → `term` → `Prop`
  := β 変換の同値閉包。
- β 同値計算 : `term -> term -> Bool`

:= 項をそれぞれ `normalize` して α 同値か比べる。

! β 同値計算は停止すれば β 同値の計算になる。

! β 同値計算は共に正規形を持つ項の β 同値の決定になる。

† 注意として、一方が正規形を持ち他方が正規形を持たない場合は β 同値ではないが、それはこの方法では決定できない。また、共に正規形を持たないが β 同値である場合が存在するが、この場合もこの方法では決定できない。これらを含めて計算するには、β 変換を両方にかけて（これは停止する）これまで出てきた項と比べるなどする必要がある。

項の間の同値関係 $t_1 \equiv t_2$ を次の規則から生成する。これは α 同値関係と β 同値関係を含みかつ関数の外延性を得るので、外延的ラムダ計算と呼ばれる体系になる。

ラムダ項の等式

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

$$\frac{\mathtt{boundedvalueOf}\, t_1 \cap \mathtt{freevalueOf}\, t_2 = \emptyset}{\mathtt{App}\, (\mathtt{Fun}\, x\, t_1)\, t_2 \equiv t_1\{x \leftarrow t_2\}}\ \text{beta}$$

$$\frac{x_1 =?x_2}{x_1 \equiv x_2}\ \text{conversion Var}$$

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

$$\frac{t_1 \equiv t_3 \quad t_2 \equiv t_4}{\mathtt{App}\, t_1\, t_2 \equiv \mathtt{App}\, t_3\, t_4}\ \text{conversion-app}$$

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

conversion というときにはこれを指すことで後々楽になるかもしれないが、これを計算する関数を作るのはいろいろ大変。