型なしラムダ項の簡約機 brdct の実装

May 16, 2017

村田 康佑

https://gist.github.com/MurataKosuke/7ce38f4231f04750b19ea086c6e9756b

brdct の使い方

- `./brdct (簡約したいλ項)`
  - `$ ./brdct "(\x.xy)(\z.(\x.xz)z)"`
- 代表的なラムダ項はマクロを定義している.
  - `&add`, `&mult`, `&exp`, `&true`, `&false`, `&S`, `&K`, `&I`, `&Y` など
  - `&12` のように，`&`と自然数を組み合わせれば Church Number を入力できる
- `./brdct (簡約したいλ項) --latex`
  - `--latex` オプションをつけると，LaTeX 形式で出力できる

型なしラムダ計算の簡約機実装

- 使用言語：Haskell
  - コードはそんなに長くない
- 簡約より，ラムダ項の表示やパーサーを書くほうが難
  - ラムダ計算の中核部分は単純：実装が簡単
  - 関数型言語なので，ほぼ定義をそのまま書くだけ
  - 一方，ラムダ項の省略表記などは複雑
    - $\lambda x.\lambda y.((M_1 M_2) M_3)$ を，$\lambda xy.M_1 M_2 M_3$ と略記.

型なしラムダ項のベータ簡約実装(1)

- ラムダ項の AST を与える BNF：

$$\langle \mathrm{Term} \rangle = \langle \mathrm{Var} \rangle \mid (\lambda \langle \mathrm{Var} \rangle . \langle \mathrm{Term} \rangle) \mid (\langle \mathrm{Term} \rangle \langle \mathrm{Term} \rangle)$$

- ラムダ項のデータ構造

```
1  data Term =
2      Var String          -- 変数
3    | Abs String Term     -- 関数抽象 (λ v.M)
4    | App Term Term       -- 関数適用 (MN)
```

  - ほぼ BNF そのまま

型なしラムダ項のベータ簡約実装 (2)

- 代入

```haskell
-- 代入
subst :: Term -> String -> Term -> Term
subst (Var x) y z   = if (x == y) then z else (Var x)
subst (App m n) y l = (App (subst m y l) (subst n y l))
subst (Abs x m) y n =
  if (x == y)
     then (Abs x m)  else
       if (isFVar x n) && (isFVar y m)
       then (Abs x1 (subst (subst m x (Var x1)) y n))
       else (Abs x (subst m y n))
     where
       x1 = newVar x ((fVar m) ++ (fVar n) ++ [y])
```

  - 愚直な実装なので，(多分) 最適化の余地がある

型なしラムダ項のベータ簡約実装 (3)

- 最も左のベータ基を探して簡約

```haskell
-- 一番左にあるベータ基を探して簡約
brdctl :: Term -> (Maybe Term)
brdctl (Var x) = Nothing
brdctl (Abs x m) = brdctl m >>= (\m1 -> Just (Abs x m1))
brdctl (App (Abs x m) n) = Just (subst m x n)
brdctl (App m n) =
  if n1 /= Nothing then n1 else n2
    where
      n1 = brdctl m >>= (\m1 -> Just (App m1 n))
      n2 = brdctl n >>= (\m2 -> Just (App m m2))
```

  - 「ベータ基がない」という事態に備えて Maybe モナド使用
  - モナドの bind($>>=$) で計算を接続

勉強になったこと

- Haskell コンパイラを使ってまともなものをつくったのははじめて
  - 実は Main 関数を自分で書いたのは初めて
  - コマンドライン引数の取り方など勉強になった
- モナドの bind 演算子 $(\gg=)$ への理解が深まった
  - 文脈付きの計算を接続するというイメージが得られた

TODO

- Infinite Reduction への対応
  - 型なしラムダ式が Infinite Reduction となるかどうかは決定不能
  - しかし明らかに対応できるもの ($\omega\omega$ とか) については対応したい
  - 上限ステップ数をユーザーが指定できるように
- パーサーの完成度を上げる
  - パーシングエラー処理
- 注目しているベータ基に下線を引くモードを実装
- 簡約戦略をユーザーが選択できるようにする
  - 現在は最左簡約 (left most reduction) のみ
  - Call by Name, Call by Value など実装予定
  - できれば，インタラクティブに簡約するベータ基を指定できるモードも