ML 演習 第6回

おおいわ

July 3, 2000

ロードマップ

- 型無し ML 風言語 (第6回)
  - 環境操作とパターンマッチ
- 型付き風言語 (第7回)
  - unification と型推論
- Prolog 風言語 (第8回)
  - unification の別の応用

今回の内容

- Mini-ML の型無し subset の実装
  - 環境操作
  - 関数適用と closure
  - パターンマッチの実装
- 構文解釈 (おまけ)
  - ocamllex, ocamlyacc

Mini-ML の型

- ML の subset (type mlvalue: miniML.ml)
  - 整数 (0, 1, 2, ...)　　Int *x*
  - 論理値 (true, false)　　Bool *b*
  - リスト　　Nil
    Cons(*x*, *xs*)
  - 関数　　Closure([*pattern*, *exp*], *env*)

Mini-ML の式 (1)

- Caml 上での表現: type expr
  - 定数式　　Const(*x* : mlvalue)
  - 変数参照　　Var(*x* : string)
  - 整数演算　　Plus(*e1*, *e2*)
    Minus(...), Times(...), Div(...)
  - 等値比較　　Equal(*e1*, *e2*)
  - リストの生成　　ConsExp(*e1*, *e2*)

Mini-ML の式 (2)

- if 文　　IfExp (*e1*, *e2*, *e3*)
- λ抽象　　LambdaExp [IdentPtn *id*, *e*]
- 関数適用　　App(e1, e2)
- match　　MatchExp(*e*, *match_list*)
  - match_list: [*pattern1*, *exp1*; *pattern2*, *exp2*; ...]
- let 束縛　　LetExp([IdentPtn *id*, *e1*], *e2*)
  LetRecExp ([IdentPtn *id*, *e1*], *e2*)

optional 課題のための拡張。
とりあえず気にしなくてよい。

Mini-ML のパターン言語

- 定数パターン　　ConstPtn(*x*)
- 変数束縛パターン　IdentPtn(*ident*)
- 任意パターン　　IdentPtn(“_”)
- リストパターン　　ConsPtn(*ptn1*, *ptn2*)

環境

- 自由変数と値の間の束縛関係を記憶
  - 評価の進行に応じて拡張される

例: let x = 5 and y = 3 in x + y

  - この下線部を評価している時点での環境は
    { x → 5, y → 3 }
  - Mini-ML で環境を拡張する構文の例:
    - App, MatchExp, LetExp, LetRecExp, ...

環境の表現

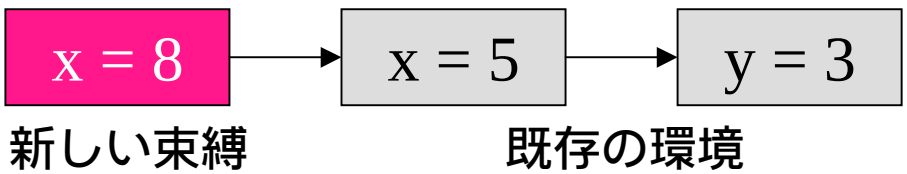
- 束縛 (string * mlvalue ref) のリスト
  - Scheme と違い、一回束縛した値は書き換わることが無いので、原理的には mlvalue でいいのだが、LetRecExp の実装の都合上 mlvalue ref の方が都合がいいのでこうしてあります。

環境の実装

- get (miniMLInterp.ml)

- eval (LetExp の節)

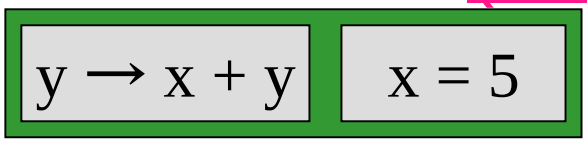
```
let rec eval env LetExp([IdentPtn id, e1], e2) =
  let v1 = eval env e1 in
  eval ( (id, ref v1) :: env ) e2
```

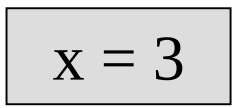


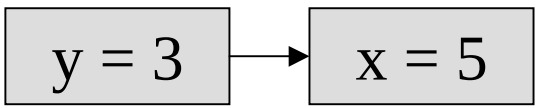
式の評価 (1)

- λ抽象
  - その時点での環境を保存
- 関数適用
  - 実引数を現在の環境で評価
  - 仮引数を実引数の評価結果に束縛し、それで保存されていた環境を拡張
  - 関数本体を拡張した環境で評価

式の評価 (2)

- f := let x = 5 in (fun y -> x + y)
  - f : 

- let x = 3 in f x
  - この時点での環境 : 

  - 実引数 “x” の評価結果 3 を y に束縛
  - 環境 

で “x + y” を評価

式の評価 (3)

- Let 式:
  - まず値を計算
  - 変数に束縛して環境を拡張
  - in 節の式を拡張された環境で評価

式の評価 (4)

- LetRec 式:
  - 先に環境を拡張 (中身の値は参照禁止)
  - 拡張された環境で式を評価
  - その値を束縛
  - in 節の式を拡張された環境で評価

→ 自分自身が束縛された環境を参照

パターンマッチ (1)

- パターンと値を見比べて束縛を作る
- データ構造パターン (ConsPtn) とのマッチは内部を再帰的に調査
  - 例:

    ConsPtn (IdentPtn x, ConstPtn (Nil)) と

    Cons (Int 1, Nil)

    → 結果は { x → 1 }

パターンマッチ (2)

ConsPtn (IdentPtn x, ConstPtn (Nil))

Cons (Int 1, Nil)

1. トップのデータ構造の比較:
   ConsPtn ←→ Cons : 内部が合えば合致
2. 第1要素の比較:
   IdentPtn x ←→ Int 1 : x を 1 に束縛
3. 第2要素の比較:
   ConstPtn (Nil) ←→ Nil : 合致

構文解析 (1)

- ocamllex
  - 文字列から「単語」を切り出す
    - 例: miniMLLexer.mll
- ocamlyacc
  - 「単語」列を構文解析
    - 例: miniMLParser.mly

構文解析 (2)

- Mini-ML 用パーサの使い方:
  - .cmo ファイルを3つ読み込む
    - miniMLReader.ml のコメント参照
    - ファイルは演習の URL を参照
  - なお、miniML.ml の定義を変更した場合、Makefile を用いて再コンパイルの必要がある場合があります。

構文解析 (3)

- 例

```
# let exp = mlexp_of_string “fun x -> x + 1”;;
- : MiniML.expr =
  LambdaExp
    [IdentPat “x”, Plus (Var “x”, Const (Int 1))]
# eval [] (mlexp_of_string “5 + 3”);;
- : MiniML.mlvalue = Int 8
```

- fun x y → x + y や let f x = x + 3 などは LambdaExp などの組み合わせに展開されます。

課題1

- miniMLInterp.ml のインタプリタに、関数適用 (App) と LetRec 式 (LetRecExp) に対する実装を追加せよ。
  - それぞれ ... になっている所を埋めてください。
  - 実装方針はここまでの説明を参照。

課題2

- パターンと値をとって、pattern match時に生じる束縛を生成する関数
  match_pattern :
    pattern → mlvalue →
      (string * mlvalue ref) list
  を作成し、
  eval に MatchExp に対する実装を追加せよ。

課題3 (optional)

- LetExp, LetRecExp の実装をパターンと and 節に対応させよ。
  - 束縛のタイミングに要注意。
  - [IdentPtn id, e1] と書いてあったところに [pattern1, exp1; pattern2, exp2] という形で複数パターンが与えられます。
- Lambda 式を複数パターン選択 (function 式) に対応させよ。
  - もちろん実際は App の書き換えの方が重要。

提出方法

- 〆切: 2000年7月17日 (月) 24:00
- 提出先: ml-report@yl.is.s.u-tokyo.ac.jp
- 題名: Report 6 学生証番号