ML 演習 第 5 回

2007/07/03

飯塚 大輔, 後藤 哲志, 前田 俊行

http://www.yl.is.s.u-tokyo.ac.jp/~sgotou/lecture/caml-enshu

今回の内容

- 言語処理系の実装 (1)

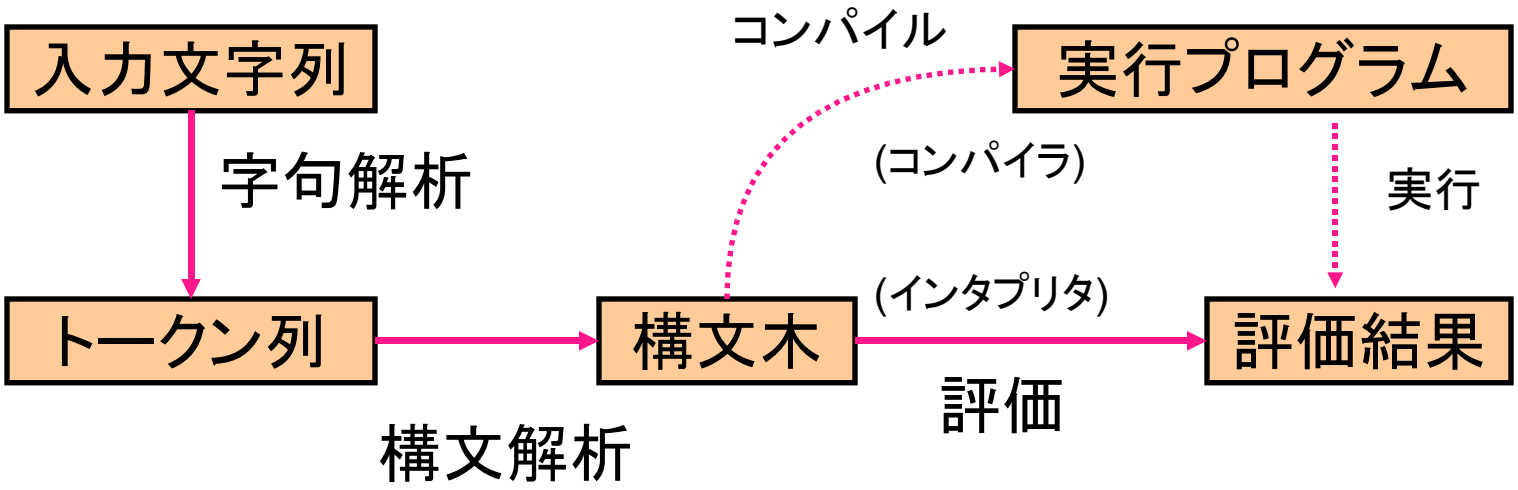
  形無し関数型言語のインタプリタの作成

言語処理系の作成

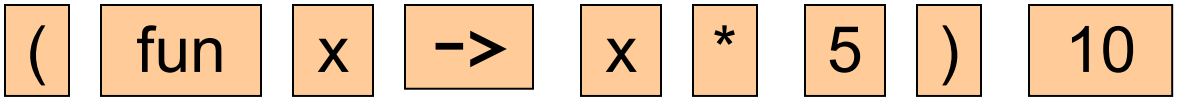
- 今後3回の予定
  - 第5回: 基本的なインタプリタの作成
    - 形無し関数型言語の処理系の作成
  - 第6回: インタプリタの様々な拡張
    - 式の評価順序に関する考察
  - 第7回: 言語処理系と型システム
    - ML 風の型推論の実装

言語処理系の構造

- 入力: プログラム文字列
- 出力: 評価結果 or 実行プログラム



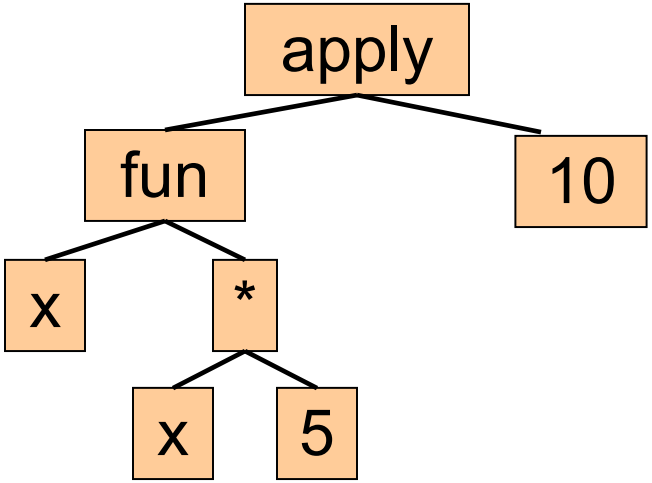
字句解析

- 入力文字列を「単語」に切り出す
  - 例: (fun x -> x * 5) 10
  - 出力: 

  - ツール: lex, flex, ocamllex etc…

構文解析 (1)

- 字句の列から文法解釈して構文木に
  - 例: (fun x -> x * 5) 10
  - 出力:



構文解析 (2)

- ツール: yacc, bison, ocamlyacc, etc…
- 今回は字句・構文解析はこちらで提供します
  - モジュール
    `MiniMLLexer`: 字句解析
    `MiniMLParser`: 構文解析
    `MiniMLReader`: 読み込み関数の定義

今回の言語 Mini-ML の構文

- O'Caml の小さな subset
  - データ型: int, bool, 関数, pair, list
  - 構文: 定数, 加減乗除, =, pair, ::,
    if, fun, 関数適用, match,
    let, let rec

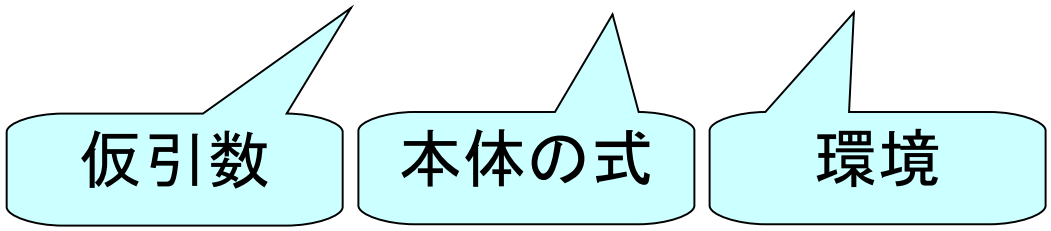
  (詳しくは別紙参照)

  - 今回は型チェックは動的に行う
    (Scheme 風に)

Mini-ML の値

- miniML.ml中のmlvalue型
  - 整数 (0, 1, 2, ...)　　`Int x`
  - 真偽値 (true, false)　`Bool b`
  - リスト　　`Nil, Cons(x, xs)`
  - ペア　　`Pair(x1, x2)`
  - 関数(クロージャ)

```
Closure([pattern, exp], env)
```



Mini-ML の式 (1/2)

- miniML.ml中のexpr型
  - 定数値　　`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)`
  - ペアの生成　`PairExp(e1, e2)`

Mini-ML の式 (2/2)

- if 式　　`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*`)`
- ペアパターン　　`PairPtn(`*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 in let 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 で環境を拡張する式の例:
    - App, MatchExp, LetExp, LetRecExp, …

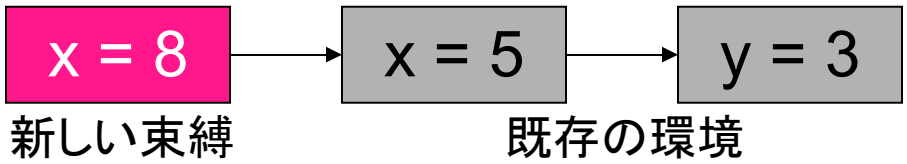
今回のインタプリタでの環境の表現

- 束縛 (string * mlvalue) のリスト

環境の実装

- 環境から変数の値を得る
  - get (miniMLInterp.ml)
- 環境を拡張する
  - 例：関数 eval の LetExp の節

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



式の評価: let 式

- let x = e_1 in e_2
  - まず e_1 を評価する
  - 次に変数 x に e_1 の評価結果を束縛して環境を拡張する
  - e_2 を拡張した環境で評価する

式の評価の例: let 式

- let x = 3 in
  let x = x + 1 in x / 2 の評価

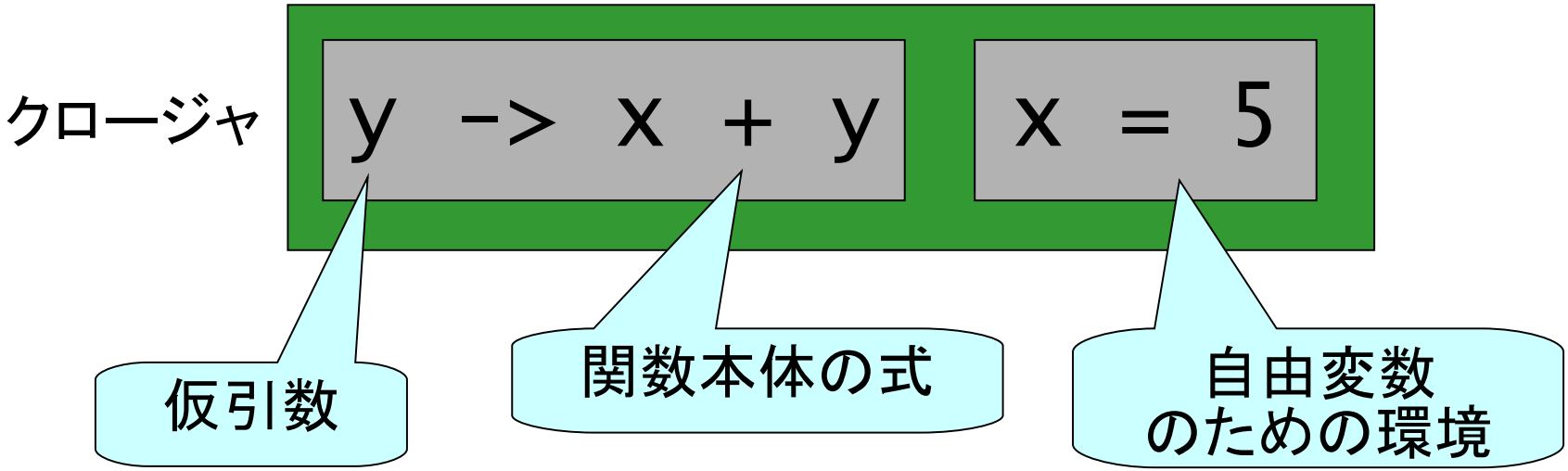
  - まず x = 3 の環境下で
    x + 1 を評価 → 4
  - x を 4 に束縛して環境を拡張
  - x = 4 → x = 3 の下で
    x / 2 を評価 → 2

式の評価: λ抽象

- fun x -> e の評価
- クロージャを評価結果として返す
  - クロージャ
    = 関数の仮引数, 関数本体, 環境 の組
  - 関数本体には自由変数があるので
    λ抽象を評価した時点での環境を
    保存しておく必要がある

式の評価の例: λ抽象

- `let x = 5 in (fun y -> x + y)` の評価結果

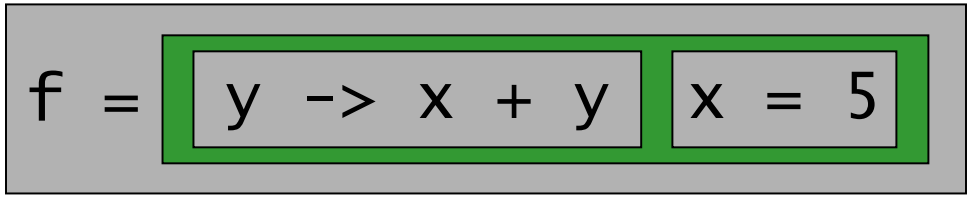


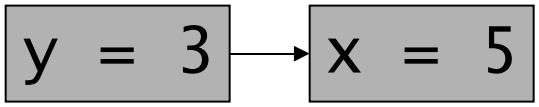
式の評価: 関数適用

- e_1 e_2 の評価
- まず、e_1、e_2 をそれぞれ評価する
  - e_1 の評価結果がクロージャでなかったらエラー
- 次に、クロージャに保存された環境を拡張する
  - 具体的にはクロージャ内の仮引数
    e_2 の評価結果に新たに束縛する
- 最後に、拡張された環境でクロージャ内の関数本体を評価する

式の評価の例: 関数適用

- 今、環境が次の通りだったとする



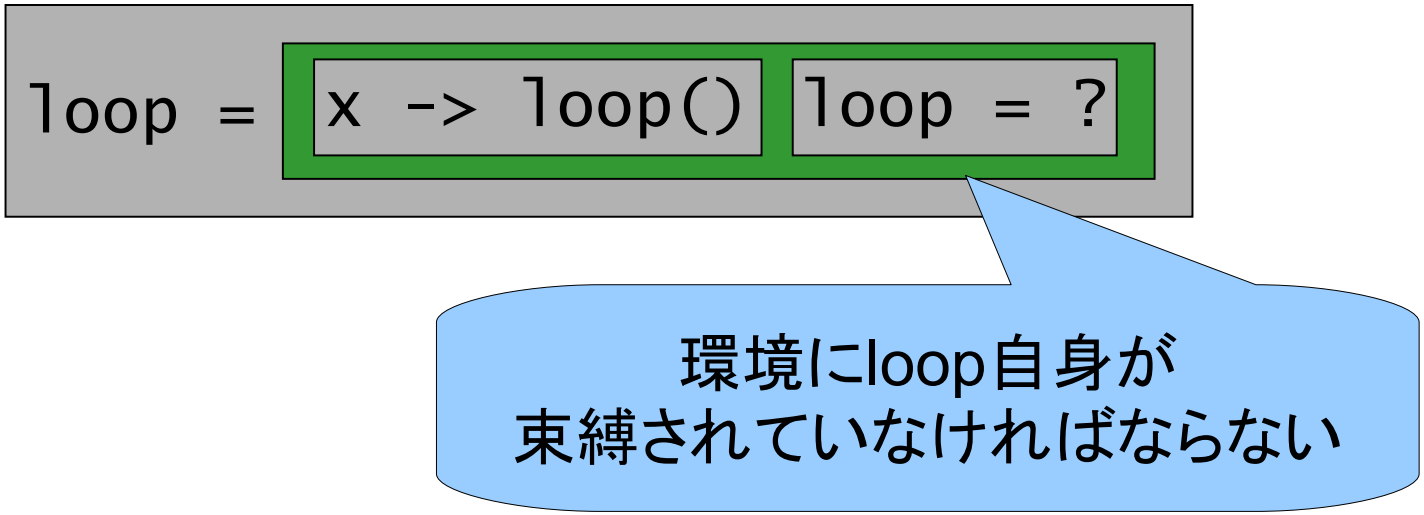
- `let x = 3 in f x` の評価結果
  - 実引数 x の評価結果 3 を y に束縛
  - 環境 

で x + y を評価

let rec式の評価の問題

- 自分自身が束縛された環境を
  参照できなければならない

- (例) let rec loop = (fun x → loop ()) in loop ()

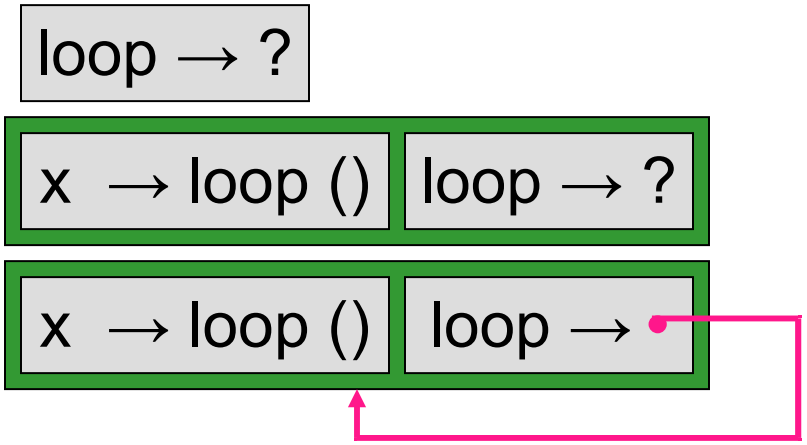


let rec 式の評価の実装の一例

- 先に環境を拡張
- 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

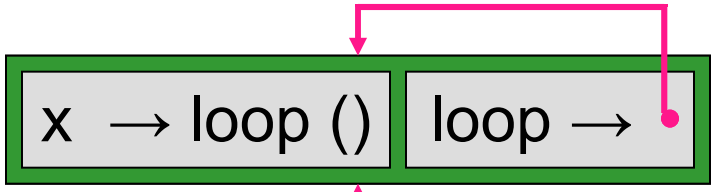
式の評価の例: let rec 式 (1/2)

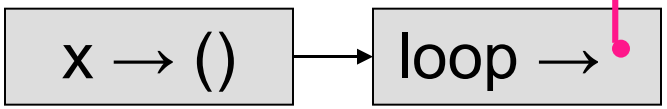
- (例) let rec loop = (fun x → loop ()) in …
  - まず環境を拡張
  - 引数を評価
  - 値を？に代入



  - referenceを使えば実現できる

式の評価の例: let rec 式 (2/2)

- (例) let rec loop = …… in loop ()
  - 

←これに () を適用
  - 環境を取り出して…
  - 

引数で拡張して…
  - 本体 “loop ()” を評価 = 振り出しに戻る

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

構文解析モジュール (1)

- 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
    - ファイルは演習のページから

      http://yl.is.s.u-tokyo.ac.jp/~sgotou/lecture/caml-enshu/sources/lecture5/
  - 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

構文解析モジュール (2)

- 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

第 5 回 課題

締め切り: 7/17 13:00

課題1 (必須)

- 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
  - 実装方針はここまでの説明を参照
  - LetRecはreferenceを使えば比較的楽に実装できる
    - miniML.ml を変更してもよい
  - もちろんreferenceを使わなくてもよい

課題2 (必須)

- 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) 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
- λ 抽象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