計算機科学実験3HSW レポート2

橘大佑
1029-31-6811
2019 年度入学

2021/7/4

工学部情報学科計算機科学コース
1029-31-6811
橘 大佑

SW レポート2

計算機科学実験及び演習 3
2021 年 7 月 4 日

Exercise 3.2.1

- プログラムの設計方針
  大域環境の拡張を行った。

- 実装の詳細な説明
  ii が 2、iii が 3、iv が 4 となるようにプログラムを変更した。インタープリタで iv+iii*ii と入力すると、4+3*2 と計算され、10 が表示された。

Exercise 3.2.2

- プログラムの設計方針
  cui.ml の read_eval_print を変更した。

- 実装の詳細な説明
  プログラムを実行した際にエラーを受け取ったら”Variable not bound: ”とメッセージを出力してインタプリタプロンプトに戻るように、 try with 構文で例外を受け取った際の処理を加えた。

Exercise 3.2.3

- プログラムの設計方針
  論理値演算のための二項演算子である && と || が SEKI,WA として表現されるように拡張を行った。

- 実装の詳細な説明
  lexer.ml に && と || をそれぞれ SEKI、WA という名前で定義した。また、parser.mly に SEKI、WA を token として追加した。論理和と論理積を加える前は演算子の優先順位は $*, +, <$ であったが、拡張後は $*, +, <, \&\&, ||$ となった。$\&\&, ||$ が右結合であることを考慮して、Expr から WExpr に遷移するように加筆し、WExpr からは左辺が SExpr、右辺が WExpr となる構造、または SExpr に遷移するよう WExpr を定義した。そして、SExpr からは左辺が LTExpr、右辺が SExpr となる構造、または LTExpr に遷移するよう SExpr を定義した。最後に、LTExpr からは左辺が LTExpr、右辺が PExpr となる構造、または PExpr に遷移するよう LTExpr を定義した。また、eval.ml の apply_prim op arg1 arg2 に op が Wa, または Seki であったときの記述を加えた。具体的には演算対象が真偽値であったときにそれぞれ論理積、論理和の演算を行うようにし、そうでない場合にはエラーを表示するようにした。最後に syntax.ml に論理積、論理和を表す binOp として、Seki、Wa を追加した。

Exercise 3.2.4

- プログラムの設計方針
  comment という再帰的なルールを lexer.mll に新しく定義することで、(*と*) で囲まれたコメントを読み飛ばすようにした。

- 実装の詳細な説明
  コメント開始記号”(*”を openComment、終端記号”*)”を closeComment と let 式で定義し、入れ子になっているコメントにも対応するために comment 関数ではコメントの深さを引数に取り、コメントの深さが 1 になるまで無視するようにした。

工学部情報学科計算機科学コース
1029-31-6811
橘 大佑

SW レポート2

計算機科学実験及び演習 3
2021 年 7 月 4 日

Exercise 3.3.1

- プログラムの設計方針
  MiniML1 インタプリタを拡張して，MiniML2 インタプリタを作成し、変数宣言を行えるようにした。

- 実装の詳細な説明
  syntax.ml に exp 型と program 型にコンストラクタを追加した。また、parser.mly に LET,IN,EQ を token に追加し、LetExpr を定義して Expr から LetExpr に遷移できるようにした。そして、lexer.ml に予約語と記号の追加を行った。最後に eval.ml の eval_exp に LetExp の記述を追加して、eval_decl に Decl を追加して、最初に束縛変数名、式をパターンマッチで取りだし、各式を評価するようにした。その値を使って現在の環境を拡張し、本体式を評価した。また、eval_decl では新たに束縛された変数、拡張後の環境と評価結果の組を返すようにした。let a = 2 in let b = 3 in a * b;; とインタープリタに入力すると 6 と出力された。

Exercise 3.4.1

- プログラムの設計方針
  MiniML3 インタプリタを作成し，高階関数が正しく動作することを確認した。以下に作成したコードを示す。これは MiniML2 では実現できなかった高階関数の例である。計算結果として 4 が表示された。

Listing 1: apply_one

```
let apply_one = fun f -> f 1 in let plus3 = fun x -> x + 3 in apply_one plus3;;
```

- 実装の詳細な説明
  syntax.ml の exp 型に新しいコンストラクタ FunExp と AppExp を追加した。そして、parser.mly に-¿と fun に対応するトークンとして RARROW と FUN を宣言して、関数定義式と関数適用式を構文解析するための規則を追加した。lexer.ml には予約語を追加し、-> をトークン FUN として切り出すように規則を追加した。最後に eval.ml には、関数が適用される際にはパラメータ名、関数本体の式に加え本体中のパラメータで束縛されていない変数（自由変数）の束縛情報（名前と値） の 3 つを組にしたデータを一般に 関数閉包・クロージャ (function closure) と呼び、これを関数を表す値として用いた。

Exercise 3.4.4

- プログラムの設計方針
  加算を繰り返して 4 による掛け算を行うプログラムを書き換えて階乗計算を行うプログラムを作成した。以下に作成したコードを示す。計算結果として 4!が計算され、24 が表示された。

Listing 2: fact

```
let makemult = fun maker -> fun x ->
  if x < 1 then 1 else x * maker maker (x + -1) in
let fact = fun x -> makemult makemult x in
fact 4;;
```

- 実装の詳細な説明
  fun において x が 1 以上のときに 4 をかけるのではなく x を乗算してから、x から 1 を引くという変更を行った。

Exercise 3.4.5

- プログラムの設計方針
  インタプリタを改造し，fun の代わりに dfun を使った関数は動的束縛を行うようにした。

- 実装の詳細な説明
  関数本体が関数式を評価した時点で評価される静的束縛とは対照的な、関数呼び出しがあった時点での環境をパラメータ・実引数で拡張した環境下で評価される動的束縛を実装した。

Exercise 3.4.6

- プログラムの設計方針
  以下のプログラムで， 二箇所の fun を dfun に置き換えて 4 通りのプログラムを実行し，その結果について説明する。

- 実装の詳細な説明

Listing 3: fact_1

```
let fact = fun n -> n + 1 in
let fact = fun n -> if n < 1 then 1 else n * fact (n + -1) in
fact 5
```

fact は関数式を評価した時点での評価となるので、fact = fun n -> n + 1 が採用される。fact 5 は 5 * fact 4 であり、fact 4 は 4 + 1 = 5 となるため、5× 5、つまり 25 と評価される。

Listing 4: fact_2

```
let fact = fun n -> n + 1 in
let fact = dfun n -> if n < 1 then 1 else n * fact (n + -1) in
fact 5
```

fact は関数呼び出しがあった時点での環境をパラメータ・実引数で拡張した環境下での評価となるので、fact = if n < 1 then 1 else n * fact (n + -1) が採用される。fact 5 は 5 * fact 4 であり、fact 4 は 4 * fact 3 となるため、fact 5 は 5 の階乗、つまり 120 と評価される。

Listing 5: fact_3

```
let fact = dfun n -> n + 1 in
let fact = fun n -> if n < 1 then 1 else n * fact (n + -1) in
fact 5
```

fact は関数式を評価した時点での評価となるので、fact = fun n -> n + 1 が採用される。fact 5 は 5 * fact 4 であり、fact 4 は 4 + 1 = 5 となるため、5× 5、つまり 25 と評価される。

Listing 6: fact_4

```
let fact = dfun n -> n + 1 in
let fact = dfun n -> if n < 1 then 1 else n * fact (n + -1) in
fact 5
```

fact は関数呼び出しがあった時点での環境をパラメータ・実引数で拡張した環境下での評価となるので、fact = if n < 1 then 1 else n * fact (n + -1) が採用される。fact 5 は 5 * fact 4 であり、fact 4 は 4 * fact 3 となるため、fact 5 は 5 の階乗、つまり 120 と評価される。

Exercise 3.5.1

- プログラムの設計方針
  syntax.ml にしたがって、parser.mly と lexer.mll を完成させ、MiniML4 インタプリタを作成し、テストした。以下に作成したコードを示す。これは MiniML3 では実現できなかった階乗を計算する再帰関数の例である。計算結果として 4!が計算され、24 が表示された。

Listing 7: fact

```
let rec fact = fun n -> if n < 1 then 1 else n*fact(n + -1) in fact 4
```

- 実装の詳細な説明
  syntax.ml には exp 型と program 型に新しいコンストラクタを追加した。また、eval.ml では関数閉包内の環境を参照型で保持するように ProcV を変更し、eval_exp に LetRecExp に関する記述を追加した。ここで、再帰関数を定義する際に、一旦ダミーの環境を作成し、関数閉包を作成した後に、その環境を更新する必要があるが、これを OCaml の参照を用いて実現した。lexer.ml には予約語”rec”を追加した。parser.mly には REC を token に追加し、toplevel に let rec に関する記述を追加し、Expr から LetRecExpr に遷移できるようにした。また、 RecDecl を eval.ml に実装する必要があるため、 eval.ml にも変更を加えた。

実装において工夫した点

できるだけ既存のものを参考にして実装を行うようにした。

感想

ocaml の基礎は理解できたのだが、そこからインタープリタ作成はかなりレベルが上がって難しかった。