SoftwareReport2

1029-28-1547 Yuya Sumie

平成 30 年 7 月 5 日

1　このレポートについて

このレポートでは Exercise ごとにプログラムの設計方針や実装の詳細な説明を記述していくことにする

2　Exercise 3.2.1

大域環境に関わるファイルは main.ml である
“ii” “iii” “iv” という値を定義するためには 31-33 行目のように空の環境を `Environment.extend` 関数を使って拡張した

3　Exercise 3.2.2

入力出力を司るのは main.ml である
main.ml の環境を読み込む関数 `read_eval_print` を改良して、エラーが起こってもプログラムの実行を終了しないようにした
具体的には `read_eval_print` 関数を try 文を使い、構文解析段階や評価段階など各所でエラーが起こった場合、それを捕捉してエラー文を表示し、環境は変えず、新しい入力を待つようにした (main.ml 15-21 行目)

4　Exercise 3.2.3

新しい演算子の導入に関わるのは、lexer と parser,eval.ml であるのでそこに | | , && の定義を追加する lexer では　&& を AND , | | を OR という名前で、処理するということを記述した (lexer.mll 34,35 行目)
parser では 2 つの演算子をどのように、またどのような優先度で評価するかを記述している．OR のほうが強弱として弱いのでそれを反映させるため、A && B や A | | B は式の一種なので Expr を評価したものの

中に ORExpr を追加し、ORExpr の中に ANDExpr に置換される構文規則を入れた。両側の引数は、LTExpr クラスの式が入るので ANDExpr の評価が LTExpr となる構文規則を入れ、&&と || の位置付けを定義できた (parser.mly 41,59-65 行目)
eval.ml では二項演算子の評価関数 `apply_prim` のなかに、And と Or の評価方法を追加した。(eval.ml 35-38 行目)
さらに、&&または || の右側の評価が再帰などにより停止しなくなった場合でも左側の引数で演算全体の評価ができる場合 (false && ... や true && ... の場合) も評価できるように `eval_exp` の中に処理を追加した.(eval.ml 46-52 行目)

5 Exercise 3.2.4

lexer に “(*” と” *)” というトークンを定義し、それを読み込んだ時の動作を定義するのが必要である (lexer.mll 44-57 行目) 入れ子構造にも対応するため `level` という変数 (何回コメントが入力されたか) を導入し、“(*” が入力されるたびにインクリメントする.
“*)” が入力されると、`level` が 0 の場合、コメント範囲を終了され、そうでない場合は `level` をデクリメントする.

6 Exercise 3.3.1

let 式の導入にあたって、構文的には、プログラムの部分と式の部分に let 式を追加しなければならない
そのためには syntax.ml の改良が必要である
階層的には大元のプログラムの部分と式の部分なので、そこに LetExp と Decl を追加した (syntax.ml 17,37 行目)
また、それに対応して parser.mly(8,26,34,78,79 行目) や lexer.mll(9,10,32 行目) では “let” “in” “EQ” のトークンを追加し、Decl や LetExp コンストラクタに入力する引数も解析の中から読み込めるように改良する eval.ml では let 式で宣言された式を現環境で評価して、それをもとに、環境を拡張し新たな環境を作る関数を記述する (eval.ml 61-65 行目)

7 Exercise 3.4.1

- fun 式の導入

let 式の導入と同様に、構文解析のフェイズにおいて必要な部分を追加する (syntax.ml 21,22 行目 parser.mly 10,37,53-55,84-86,91,92 行目 lexer.mll 12,33 行目)
parser.mly について fun 式は式の一種であり、適用式は、評価後は整数値と真偽値となるので、Mult の引数にすることができるので、そのレベルに AppExpr を置く
eval.ml では関数閉包・クロージャという機構を使い、実装する値の一種として、パラメータ名、関数式、変数と束縛情報を組とするもの (これが関数閉包・クロージャ) を定義する
評価の仕方 (eval_exp) は教科書を参照 (67-78 行目)
教科書に書かれているところ以外の変更点は `string_of_exval` 関数の中に、関数式を引数にとるときは、評価後、標準出力に、と出力するというパターンを追加した部分である (23 行目)

8 Exercise 3.5.1

- 再帰的関数定義の導入

まずは、構文解析用に追加される記述を追加する (syntax.ml 29,39 行目 parser.mly 30,39,95 行目 lexer.mll 14 行目). parser.mly では、再帰的関数は、式レベルとプログラムレベルに置く.
eval.ml ではバックパッチという機構が採用されている.
再帰関数では、宣言しようとしている関数のなかに、その関数が含まれているので環境の束縛情報などが ML[3] のままではうまくいかない. そこで、はじめはダミーの環境を作成し、それで関数閉包を作成する. そののち、環境が拡張されるとその環境でダミー環境を更新するというものである.OCaml では参照を用いて実装されており、ML[4] のために ProcV の引数の値を環境から環境の参照に変更している (11 行目)
`eval_exp` の LetRecExp を評価する部分では、ダミー環境をまず作り、現在の環境を拡張した環境 `newenv` を作ると、破壊的代入を行い、環境を更新する.(79-87 行目)
それに伴い、ProcV を使うほかの式の評価の場合に参照から環境を求めなければならなくなったので、そこを変更し (68,76 行目)、eval_decl の RecDecl の評価の仕方も定義した (94-99 行目)