プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 4

実行時の環境 (environment)

変数の束縛を表す (変数とその値の対応): 例: x=10, y=20, z=30
ここでは、OCaml のリストを使って表現する。

(空の環境) <-> []
x=10, y=20 <-> ["x",10); ("y",20)]

OCaml の組 (タプル) とリスト

- 組 (a, b, c)
- リスト: [a; b; c]

組とリストの違い:

ok (10,"abc",true) ng [10;"abc";true]
ng (10,20)=(20,30,40) ok [10;20]=[20;30;40]

変数と変数束縛のある言語の意味

```
let rec eval e env =
 match e with
 | CstI i    -> i
 | Var x     -> lookup env x
 | Let(x, erhs, ebody)
     -> let xval = eval erhs env
        let env1 = (x, xval) :: env
         eval ebody env1
 | Prim("+", e1, e2) -> (eval e1 env) + (eval e2 env)
 | _ -> failwith "unknown expression"
```


環境の操作

初期値 (空の環境) [ ]
環境の延長 (extend) a :: b

("x",10) :: [("y",20); ("z",30)]
-> [("x",10); ("y",20); ("z",30)]

環境からの値の取り出し (lookup)

lookup [("x",10); ("y",20); ("x",30)] "x"
-> 10 (30 ではない)

後から入れたものが優先される.

評価の例

- let x=10 in x+20
- let x=10 in let x=30 in x+20
  - x+20 を評価する時の環境は [("x",30); ("x",10)]
- let x=10 in (let x=30 in x+20) + x * 5
  - x+20 を評価する時の環境は [("x",30); ("x",10)]
  - x*5 を評価する時の環境は [("x",10)]

簡単なスタック機械

- 1 本のスタックをもつ抽象機械:
  - プログラムは命令列として与える。
  - スタック以プ外にはメモリはない。
- 命令 (in BNF):

$$r ::= \texttt{RCstI}\ i \mid \texttt{RAdd} \mid \texttt{RDup} \mid \texttt{RSwap}$$

i は整数定数とする.

```
type rinstr =
   | RCstI of int | RAdd | RDup | RSwap
```


簡単なスタック機械の意味

Instruction	before	after	Explanation
RCst i	s	s,i	Push
RAdd	s,i1,i2	s,(i1+i2)	Add
RDup	s,i	s,i,i	Duplicate
RSwap	s,i1,i2	s,i2,i1	Swap

- 例 1. [RCst 10; RCst 20; RCst 30; RAdd; RAdd]
  - 10+(20+30) の計算に相当する.
- 例 2. [RCst 10; RCst 20; RAdd; RCst 30; RAdd]
  - (10+20)+30 の計算に相当する.

逆ポーランド記法 (reverse Polish form)

簡単なスタック機械の評価器

主要部分のみ (詳細は教科書またはコードを参照のこと)

```
let rec reval inss stack =
 match (inss, stack) with
 | ([]       , v :: _)    -> v
 | (RCst i::r, s)         -> reval r (i::s)
 | (RAdd  ::r, i2::i1::s) -> reval r ((i1+i2)::s)
 | (RDup  ::r, i1::s)     -> reval r (i1::(i1::s))
 | (RSwap ::r, i2::i1::s) -> reval r (i1::(i2::s))
 ...
```

inss は命令列 (rinstr 型のリスト)
stack はスタック (リストで表現する)

先週の宿題

スタック機械があるとき、整数定数と足し算だけの言語はどう処理できるか考えてみてください。
例: 10+(20+30) を以下の命令列に翻訳すればよい.

- `[RCst 10; RCst 20; RCst 30; RAdd; RAdd]`

翻訳=プログラム変換=コンパイル (Compilation)

宿題の答

「整数定数と足し算だけの言語」から「スタック機械の命令列」への翻訳:

```
let rec comp e =
 match e with
 | CstI i    ->  [(RCstI i)]
 | Prim("+", e1, e2) -> (comp e1) @ (comp e2) @ [RAdd]
```

例: `comp (Prim("+", CstI 10, Prim("+", CstI 20, CstI 30)))`

- `(comp (CstI 10))@(comp (Prim("+", CstI 20, CstI 30)))@[RAdd]`
- `[(RCstI 10)]@((comp (CstI 20))@(comp (CstI 30))@[RAdd])@[RAdd]`
- `[(RCstI 10)]@([(RCstI 20)]@[(RCstI 30)]@[RAdd])@[RAdd]`
- `[(RCstI 10);(RCstI 20);(RCstI 30);RAdd;RAdd]`

コンパイラ= 翻訳系

ここでは、以下の 2 言語間の変換:

- ソース: 整数と足し算からなる言語の項
- ターゲット: スタック機械の命令列

多くの場合,

- ソース: 高レベル言語のプログラム
- ターゲット: 低レベル言語 (機械語など) のプログラム
- コンパイルの過程で，プログラムを解析して，高性能プログラムへ変換することが多い (最適化)

発展課題

テキスト第 2 章では,「整数定数，四則演算，変数，変数束縛をもつ言語のプログラム」から「拡張されたスタック機械の命令列」への変換についても論じているが，具体的な実装は示していない．これをどう実現すればいいか，考えよ.

例: `let z=17 in z+ z` を以下の命令列の翻訳したい。

- `[SCstI 17; SVar 0; SVar 1; SAdd; SSwap; SPop]`
- `SVar n` は、スタックトップから n 番目の値を取ってきて、スタックにプッシュする命令。
- 注意点: z が 2 回出現するが、翻訳された命令列では SVar 0 と SVar 1 という異なるインデックスで表現。

命令の構文:

```
type sinstr =
   | SCstI of int | SVar of int | SAdd | SPop | SSwap
```