プログラム言語論

亀山幸義

筑波大学 情報科学類

スタック機械 と PostScript 言語

目次

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。
  - メモリは、スタックのみ。
- 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.

このスタック機械の命令をあらわす OCaml の型:

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

例: `RCstI(10)`

簡単なスタック機械の意味

命令	実行前	実行後	説明 (参考)
`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))
 | _ → failwith "undefined transition"
```

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)
例: `reval [RCst 10; RCst 20; RAdd] [] ==> [30]`

課題

整数定数と足し算だけの言語で書かれたプログラム (式) は、スタック機械の命令列としては、どういう風にあらわされるか?
例 1. 10+(20+30) は、以下の命令列に翻訳するとよい。

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

例 2. (10+20)+30 は、以下の命令列に翻訳するとよい。

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

翻訳=プログラム変換=コンパイル (Compilation)

課題の答

「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」への翻訳をする関数：

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

例 1: `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]`

課題の答

例 2: `comp (Prim("+", Prim("+",CstI 10, CstI 20), CstI 30))`

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