プログラム言語論

亀山幸義

筑波大学コンピュータサイエンス専攻

筑波大学 情報科学類 講義

関数型プログラム言語の特徴

- 関数プログラミング: 単一代入，再帰呼出し，高階関数，データ型の活用
- 単一代入⇒副作用の分離・明示
- 高階関数と静的束縛⇒関数クロージャ
- データ型⇒ヒープ
- 再帰呼出し⇒末尾再帰

復習: 関数呼出しの処理

C 言語など、値呼び、静的束縛の言語での $f(e)$ の処理 (f の仮引数を x とする。)

- まず e を計算して、その値 v を得る。
- $[x = v]$ という束縛等を含むスタックフレームを作り、スタックに push する。
- f の処理を行う。(通常は、スタックのトップに、f が返す値を積んで終了する。)
- 先ほど push したスタックフレームを pop する。

関数呼出しを開始するたびに、スタックには 1 つフレームが積まれ、呼出された関数の計算を終了するたびに、スタックから 1 つフレームがはずされる。

関数クロージャ

静的束縛で、関数を第一級のデータとする言語で必要。
関数の定義式と、環境 (あるいはスタックフレームへのポインタ) をセットにしたもの。
関数 (の定義式) に自由変数を含むとき、その値を得るための環境を付ける。

```
  (let  x = 10 in
     fun y -> x + y) 20
==>
  FunClos(fun y-> x+y, [x=10]) 20
==>
  x+y  in [x=10, y=20]
==>
  30
```

スタックとヒープ

スタック

- stack は、プログラム上のブロック構造に対応する。
- stack には、ブロックに局所的な変数の値などが格納される。
- ブロックが終了した後も行き残るデータは、stack に置くことはできない。

ヒープ

- heap は、ブロック構造に対応しないデータを格納する。
- 関数クロージャ、構造を持つデータ型 (ペア、組、レコードなど)、オブジェクト指向言語のオブジェクトなど。

ヒープの管理

- C 言語では、プログラマの責任 (malloc, free 関数)
- 多くの関数型言語やオブジェクト指向言語では、処理系の責任 (ごみ集め,Garbage Collection)

追記: 「局所変数の値が、構造を持つデータ」の場合、データそのものはヒープに取られ、スタック上の局所変数からはヒープ上の領域へのポインタがはられる。

末尾再帰

関数呼び出しが「最後」(末尾) になるもののこと。
末尾呼出しでない例:

```
let rec f x = if x=0 then 0 else f(x-1) + 10
```

末尾呼出しの例:

```
let rec f x = if x=0 then 1 else f x
```

すべての関数呼び出しが末尾再帰であれば、呼び出しにあたって現在のスタックフレームを保存する必要はない。
⇒手続き型言語における「ループ」と同様の、効率的な処理が可能。

末尾再帰の場合の関数呼出しの処理

$f(e)$ の処理 (f の仮引数を x とする。)

- まず e を計算して、その値 v を得る。
- $[x = v]$ という束縛等を含むスタックフレームを作り、現在のスタックに上書きする。
- f の処理を行う。(通常は、スタックのトップに、f が返す値を積んで終了する。)
- f の処理が終わっても、スタックを pop しない。

関数 $f0$ から $f1$ を (末尾再帰でなく) 呼んで、$f1$ から末尾再帰で $f2$ を呼んだとき、$f2$ の計算が終了したら、$f1$ の処理に戻るのではなく、$f0$ の処理に戻る。