プログラム言語論 資料 (修正版)

変数と関数を持つ言語とその意味論

亀山 幸義

1 言語 Arith に対する抽象機械 (つづき)

言語 Arith は、整数、足し算、かけ算だけからなるプログラム言語であり、前回の資料では、Arith に対する抽象機械 CK を定義した。

CK 機械は、スタックに基づく機械 (計算機) であり、「言語 Arith で記述されたプログラム (実際には、足し算とかけ算からなる式)」を受け取り、それを実行して、「計算結果 (実際には、整数)」を返すものであった。

CK 機械は、言語 Arith の意味を厳密に定めるという目的では満足の行くものであったが、実行性能は良くない。この「プログラム言語論」の授業は、コンパイラの授業ではないので、性能を追い求めることはしないが、授業で PostScript 言語に言及したので、ここで、CK 機械よりもう少し性能の良い抽象機械を導入しておく。

まず、言語 Arith のプログラムを、以下の構文の式に変換 (翻訳、あるいは、大袈裟に言えば「コンパイル」) しておく。

$$P ::= \mathtt{Int}(n) \mid P, P, \mathtt{Plus} \mid P, P, \mathtt{Times}$$

たとえば、$\mathtt{Plus}(\mathtt{Int}(1), \mathtt{Times}(\mathtt{Int}(2), \mathtt{Int}(3)))$ という Arith の式は、$\mathtt{Int}(1), \mathtt{Int}(2), \mathtt{Int}(3), \mathtt{Times}, \mathtt{Plus}$ という式に変換される。

抽象機械の状態: $\mathtt{eval}\langle P \mid K\rangle$ という状態のみとなる。P は上記の構文の式であり、K は整数をつんだスタックである。

抽象機械の動作 (状態遷移):

$$P \to \mathtt{eval}\langle P \mid \mathtt{init}\rangle \quad \text{(初期状態)}$$

$$\begin{aligned}
\mathtt{eval}\langle \mathtt{Int}(n), P \mid K\rangle &\to \mathtt{eval}\langle P \mid n::K\rangle \\
\mathtt{eval}\langle \mathtt{Plus}, P \mid n_2::n_1::K\rangle &\to \mathtt{eval}\langle P \mid n::K\rangle \quad (n_1 \dot{+} n_2 = n) \\
\mathtt{eval}\langle \mathtt{Times}, P \mid n_2::n_1::K\rangle &\to \mathtt{eval}\langle P \mid n::K\rangle \quad (n_1 \dot{*} n_2 = n)
\end{aligned}$$

$$\mathtt{eval}\langle (\text{空}) \mid n::K\rangle \to n \quad \text{(終了状態)}$$

この抽象機械は、前回の抽象機械より簡単になっており、実行性能も良いと考えられる。

演習 1.1 以下の式を P の式に変換した上で、上記抽象機械で実行せよ。

- *Int(10)*
- *Times(Plus(Int(2),Int(3)),Int(4))*
- *Times(Int(2),Plus(Int(3),Int(4)))*

2 変数と関数をもつ言語

以前に定義した「算術演算だけを持つ非常に小さなプログラム言語 (Arith)」に、いくつかの機能を追加して、それに対する意味論 (抽象機械による意味論) を見てみよう。

- 言語 Env: 変数と、変数に対する束縛 (let 式) を追加した言語である。対応する抽象機械は、実行時に「変数が現在どういう値を持っているか」の情報 (環境 E) が必要になる (CK 機械が CEK 機械となる)。
- 言語 Fun: 関数定義と関数呼出しを追加した言語である。ただし、関数定義はトップレベルのみとし、また、関数をデータとして、他の関数の引数にしたり変数に格納したりすることはできないものとする。(いわゆる first-order function のみを持つ言語である。)
- 言語 Lam: 関数を動的に定義し、それをデータとして他の関数の引数にしたり変数に格納したりできる言語である。(いわゆる higher-order function を持つ言語である。)

これらの言語では、C 言語や Java 言語における $x = x + 1$; のように、変数の値を変更する機能は持たせない。これらへの対応は、(将来) 「状態 (S)」を抽象機械に追加することにより行われる。

3 構文の定義

それぞれの言語の抽象構文 (e_0:Arith, e_1:Env, e_2:Fun, Lam)

$$e_0 ::= \texttt{Int}(e_0) \mid \texttt{Plus}(e_0, e_0) \mid \texttt{Times}(e_0, e_0)$$
$$e_1 ::= \texttt{Int}(e_1) \mid \texttt{Plus}(e_1, e_1) \mid \texttt{Times}(e_1, e_1) \mid \texttt{Var}(x) \mid \texttt{Let}(x, e_1, e_1)$$
$$e_2 ::= \texttt{Int}(e_2) \mid \texttt{Plus}(e_2, e_2) \mid \texttt{Times}(e_2, e_2) \mid \texttt{Var}(x) \mid \texttt{Let}(x, e_2, e_2) \mid \texttt{Lam}(x, e_2) \mid \texttt{App}(e_2, e_2)$$

ここで，$\texttt{Let}(x, e_1, e_2)$ は，let x=e_1 in e_2 (ML 言語) や，(let ((x e_1)) e_2) (Lisp/Scheme 言語) を意味する構文であり，$x = e_1$ のもとで e_2 を計算するという意味の式である．たとえば $\texttt{Let}(x, \texttt{Int}(10), \texttt{Var}(x))$ を計算すると 10 になることを想定している．

Fun は上記の構文 e_2 に、以下の制限をつけたものとする。

- $\texttt{Let}(f, \texttt{Lam}(x, e_1), \texttt{Let}(g, \texttt{Lam}(y, e_2), e_3))$ のように、すべての関数定義が式の冒頭にある．
- それぞれの関数の定義の本体は、閉じている (関数の引数と，他の関数の呼び出し以外には，自由な変数を持たない)．

※ これは，きれいな制限ではないが、ここでは、導入する構文の量を最小限にするため、Lam の構文を流用して Fun にも使っている。Lam にはこのような制限は一切ない。

Env のプログラム例 2: $\texttt{Let}(x, \texttt{Int}(10), \texttt{Let}(y, \texttt{Int}(20), \texttt{Plus}(\texttt{Var}(x), \texttt{Times}(\texttt{Var}(y), \texttt{Int}(30)))))$
Func のプログラム例: $\texttt{Let}(f, \texttt{Lam}(x, \texttt{Plus}(\texttt{Var}(x), \texttt{Int}(1))), \texttt{Let}(y, \texttt{Int}(20), \texttt{App}(\texttt{Var}(f), \texttt{App}(\texttt{Var}(f), \texttt{Var}(y)))))$
Lam のプログラム例: $\texttt{Let}(x, \texttt{Int}(10), \texttt{Let}(f, \texttt{Lam}(\texttt{Var}(y), \texttt{Plus}(\texttt{Var}(x), \texttt{Var}(y))), \texttt{Let}(x, \texttt{Int}(20), \texttt{App}(\texttt{Var}(f), \texttt{Int}(30)))))$

4 抽象機械

言語 Env 等に対応した抽象機械として、CK 機械に環境 (E) を加えた CEK 機械を提議する。

- C ... control string
- E ... environment (環境、つまり、変数が現在どういう値を持っているかを決めるもの)
- K ... continuation

直感的には、E は、x=10,y=20,z=30 といった情報を表す。

抽象機械の状態: $\texttt{eval}\langle C \mid E \mid K\rangle$ と $\texttt{apply}\langle K \mid v\rangle$ の 2 種類の状態がある。

言語 Lam では、計算結果が整数とは限らない。(計算をするとその結果が関数になる、ということがあるので) そこで、先回りして、「計算結果」を意味する「値 (value)」 という概念を以下で定義する。

$$
\begin{aligned}
v_1 &::= n \\
v_2 &::= n \mid \texttt{Func}(e_2) \\
v_3 &::= n \mid \texttt{Closure}(x, e_2, E)
\end{aligned}
$$

v_1 は 言語 Env に対応する機械での値, v_2 は 言語 Func に対応する機械での値, v_3 は 言語 Lam に対応する機械での値である。

$\texttt{Func}(e)$ は、式 e が関数のときのみ使い、「計算結果が関数 e である」ことを意味する。$\texttt{Closure}(x, e_2, E)$ は関数クロージャであるが、別途説明する。

環境 E に対する操作:

- $\texttt{[]}$: 空の環境 (どの変数も値を持っていない環境をあらわす)。
- $E[x = v]$: 環境 E に、$x = v$ という束縛を追加した環境。
- $\texttt{lookup}(x, E)$: 環境 E における変数 x の値 (変数 x の値が E に含まれていなければエラーとする)。

Env に対する CEK 機械の動作 (状態遷移):

$$
e \to \texttt{eval}\langle e \mid \texttt{[]} \mid \texttt{init}\rangle
$$

$$
\begin{aligned}
\texttt{eval}\langle \texttt{Var}(x) \mid E \mid K\rangle &\to \texttt{apply}\langle K \mid v\rangle \quad (v = \texttt{lookup}(x, E)) \\
\texttt{eval}\langle \texttt{Int}(n) \mid E \mid K\rangle &\to \texttt{apply}\langle K \mid n\rangle \\
\texttt{eval}\langle \texttt{Plus}(e, e') \mid E \mid K\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{plus}1(e', E) :: K\rangle \\
\texttt{eval}\langle \texttt{Times}(e, e') \mid E \mid K\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{times}1(e', E) :: K\rangle \\
\texttt{eval}\langle \texttt{Let}(x, e, e') \mid E \mid K\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{letin}(x, e', E) :: K\rangle
\end{aligned}
$$

$$
\begin{aligned}
\texttt{apply}\langle \mathit{plus}1(e, E) :: K \mid v\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{plus}2(v) :: K\rangle \\
\texttt{apply}\langle \mathit{plus}2(n_1) :: K \mid n_2\rangle &\to \texttt{apply}\langle K \mid n\rangle \quad (n_1 \dot{+} n_2 = n) \\
\texttt{apply}\langle \mathit{times}1(e, E) :: K \mid v\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{times}2(v) :: K\rangle \\
\texttt{apply}\langle \mathit{times}2(n_1) :: K \mid n_2\rangle &\to \texttt{apply}\langle K \mid n\rangle \quad (n_1 \dot{*} n_2 = n) \\
\texttt{apply}\langle \mathit{letin}(x, e, E) :: K \mid v\rangle &\to \texttt{eval}\langle e \mid E[x = v] \mid K\rangle \\
\texttt{apply}\langle \texttt{init} \mid v\rangle &\to v \quad \text{(最終結果)}
\end{aligned}
$$

Func に対する抽象機械の動作 (追加分のみ):

$$
\begin{aligned}
\texttt{eval}\langle \texttt{Let}(f, \texttt{Lam}(x, e_1), e_2) \mid E \mid K\rangle &\to \texttt{eval}\langle e_2 \mid E[f = \texttt{Func}(\texttt{Lam}(x, e_1))] \mid K\rangle \\
\texttt{eval}\langle \texttt{App}(e, e') \mid E \mid K\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{apply}1(e', E) :: K\rangle \\
\texttt{apply}\langle \mathit{apply}1(e, E) :: K \mid v\rangle &\to \texttt{eval}\langle e \mid E \mid \mathit{apply}2(v) :: K\rangle \\
\texttt{apply}\langle \mathit{apply}2(\texttt{Func}(\texttt{Lam}(x, e))) :: K \mid v\rangle &\to \texttt{eval}\langle e \mid \texttt{[]}[x = v] \mid K\rangle
\end{aligned}
$$

5 演習

1. 以下のプログラムに対する CEK 機械の状態遷移を書きなさい。(ステップ数が非常に多い場合は、適宜省略して書いてもよい。)

- Env のプログラム例: $\texttt{Let}(x, \texttt{Int}(10), \texttt{Let}(y, \texttt{Int}(20), \texttt{Plus}(\texttt{Var}(x), \texttt{Times}(\texttt{Var}(y), \texttt{Int}(30)))))$
- Func のプログラム例: $\texttt{Let}(f, \texttt{Lam}(x, \texttt{Plus}(\texttt{Var}(x), \texttt{Int}(1))), \texttt{Let}(y, \texttt{Int}(20), \texttt{App}(\texttt{Var}(f), \texttt{App}(\texttt{Var}(f), \texttt{Var}(y)))))$

2. 上記の CEK 機械は、Env と Func に対する「1 つの」意味論を決めている。それは、以下の観点では、どれに該当するか？(比較が意味を持たない観点も含むことに注意)

- 値呼び vs 名前呼び vs 必要呼び
- 静的束縛 vs 動的束縛

3. (発展課題) また、CEK 機械の状態遷移を変更して、上記のうち対応していない意味論に対応させるにはどうしたらよいか考えてみよ。

6 宿題

以下の Lam のプログラム例は、Func に対する抽象機械ではうまく処理できない。その理由を考えよ。また、どう対応させたらよいか考えてみよ。

$\texttt{Let}(x, \texttt{Int}(10), \texttt{Let}(f, \texttt{Lam}(y, \texttt{Plus}(\texttt{Var}(x), \texttt{Var}(y))), \texttt{Let}(x, \texttt{Int}(20), \texttt{App}(\texttt{Var}(f), \texttt{Int}(30)))))$