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 (演習の答えなど)

亀山 幸義

1 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。

- $\texttt{Int}(10)$

  答.

$$\begin{aligned}\texttt{Int}(10) &\to \texttt{eval}\langle \texttt{Int}(10) \mid \texttt{init}\rangle \\ &\to \texttt{eval}\langle (\text{空}) \mid 10::\texttt{init}\rangle \\ &\to 10\end{aligned}$$

- $\texttt{Times}(\texttt{Plus}(\texttt{Int}(2),\texttt{Int}(3)),\texttt{Int}(4))$

  答.

$$\begin{aligned}\texttt{Int}(2),\texttt{Int}(3),\texttt{Plus},\texttt{Int}(4),\texttt{Times} &\to \texttt{eval}\langle \texttt{Int}(2),\texttt{Int}(3),\texttt{Plus},\texttt{Int}(4),\texttt{Times} \mid \texttt{init}\rangle \\ &\to \texttt{eval}\langle \texttt{Int}(3),\texttt{Plus},\texttt{Int}(4),\texttt{Times} \mid 2::\texttt{init}\rangle \\ &\to \texttt{eval}\langle \texttt{Plus},\texttt{Int}(4),\texttt{Times} \mid 3::2::\texttt{init}\rangle \\ &\to \texttt{eval}\langle \texttt{Int}(4),\texttt{Times} \mid 5::\texttt{init}\rangle \\ &\to \texttt{eval}\langle \texttt{Times} \mid 4::5::\texttt{init}\rangle \\ &\to \texttt{eval}\langle (\text{空}) \mid 20::\texttt{init}\rangle \\ &\to 20\end{aligned}$$

- $\texttt{Times}(\texttt{Int}(2),\texttt{Plus}(\texttt{Int}(3),\texttt{Int}(4)))$

  答.

$$\begin{aligned}\texttt{Int}(2),\texttt{Int}(3),\texttt{Int}(4),\texttt{Plus},\texttt{Times} &\to \texttt{eval}\langle \texttt{Int}(2),\texttt{Int}(3),\texttt{Int}(4),\texttt{Plus},\texttt{Times} \mid \texttt{init}\rangle \\ &\to \texttt{eval}\langle \texttt{Int}(3),\texttt{Int}(4),\texttt{Plus},\texttt{Times} \mid 2::\texttt{init}\rangle \\ &\to \texttt{eval}\langle \texttt{Int}(4),\texttt{Plus},\texttt{Times} \mid 3::2::\texttt{init}\rangle \\ &\to \texttt{eval}\langle \texttt{Plus},\texttt{Times} \mid 4::3::2::\texttt{init}\rangle \\ &\to \texttt{eval}\langle \texttt{Times} \mid 7::2::\texttt{init}\rangle \\ &\to \texttt{eval}\langle (\text{空}) \mid 14::\texttt{init}\rangle \\ &\to 14\end{aligned}$$

2 5 節の演習

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: $\texttt{Let}(x, \texttt{Int}(10), \texttt{Let}(y, \texttt{Int}(20), \texttt{Plus}(\texttt{Var}(x), \texttt{Times}(\texttt{Var}(y), \texttt{Int}(30)))))$

  ここで、$e_1 = \texttt{Plus}(\texttt{Var}(x), \texttt{Times}(\texttt{Var}(y), \texttt{Int}(30)))$ とする。

$$
\begin{aligned}
\texttt{Let}(x, \texttt{Int}(10), \texttt{Let}(y, \texttt{Int}(20), e_1)) &\to \texttt{eval}\langle \texttt{Let}(x, \texttt{Int}(10), \texttt{Let}(y, \texttt{Int}(20), e_1)) \mid \texttt{[]} \mid \texttt{init} \rangle \\
&\to \texttt{eval}\langle \texttt{Int}(10) \mid \texttt{[]} \mid \mathit{letin}(x, \texttt{Let}(y, \texttt{Int}(20), e_1), \texttt{[]})\texttt{::init} \rangle \\
&\to \texttt{apply}\langle \mathit{letin}(x, \texttt{Let}(y, \texttt{Int}(20), e_1), \texttt{[]})\texttt{::init} \mid 10 \rangle \\
&\to \texttt{eval}\langle \texttt{Let}(y, \texttt{Int}(20), e_1) \mid \texttt{[]}[x = 10] \mid \texttt{init} \rangle \\
&\to \texttt{eval}\langle \texttt{Int}(20) \mid \texttt{[]}[x = 10] \mid \mathit{letin}(y, e_1, \texttt{[]}[x = 10])\texttt{::init} \rangle \\
&\to \texttt{apply}\langle \mathit{letin}(y, e_1, \texttt{[]}[x = 10])\texttt{::init} \mid 20 \rangle \\
&\to \texttt{eval}\langle e_1 \mid \texttt{[]}[x = 10][y = 20] \mid \texttt{init} \rangle \\
&\to \texttt{eval}\langle \texttt{Var}(x) \mid \texttt{[]}[x = 10][y = 20] \mid \mathit{plus}1(\texttt{Times}(\cdots), E)\texttt{::init} \rangle \\
&\to \texttt{apply}\langle \mathit{plus}1(\texttt{Times}(\cdots), E)\texttt{::init} \mid 10 \rangle \\
&\to \cdots \to \texttt{apply}\langle \texttt{init} \mid 610 \rangle
\end{aligned}
$$

  ただし、$E = \texttt{[]}[x = 10][y = 20]$ と置いた。

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

  ここで、$e_2 = \texttt{Plus}(\texttt{Var}(x), \texttt{Int}(1))$, $e_3 = \texttt{Let}(y, \texttt{Int}(20), \texttt{App}(\texttt{Var}(f), \texttt{Var}(y)))$ とおく。

  また、$E_1 = \texttt{[]}[f = \texttt{Func}(\texttt{Lam}(x, e_2))][y = 20]$ とする。

$$
\begin{aligned}
\texttt{Let}(f, \texttt{Lam}(x, e_2), e_3) &\to \texttt{eval}\langle \texttt{Let}(f, \texttt{Lam}(x, e_2), e_3) \mid \texttt{[]} \mid \texttt{init} \rangle \\
&\to \texttt{eval}\langle e_3 \mid \texttt{[]}[f = \texttt{Func}(\texttt{Lam}(x, e_2))] \mid \texttt{init} \rangle \\
&\to \cdots \to \texttt{eval}\langle \texttt{App}(\texttt{Var}(f), \texttt{Var}(y)) \mid E_1 \mid \texttt{init} \rangle \\
&\to \texttt{eval}\langle \texttt{Var}(f) \mid E_1 \mid \mathit{apply}1(\texttt{Var}(y), E_1)\texttt{::init} \rangle \\
&\to \texttt{apply}\langle \mathit{apply}1(\texttt{Var}(y), E_1)\texttt{::init} \mid \texttt{Func}(\texttt{Lam}(x, e_2)) \rangle \\
&\to \texttt{eval}\langle \texttt{Var}(y) \mid E_1 \mid \mathit{apply}2(\texttt{Func}(\texttt{Lam}(x, e_2)))\texttt{::init} \rangle \\
&\to \texttt{apply}\langle \mathit{apply}2(\texttt{Func}(\texttt{Lam}(x, e_2)))\texttt{::init} \mid 20 \rangle \\
&\to \texttt{eval}\langle e_2 \mid \texttt{[]}[x = 20] \mid \texttt{init} \rangle \\
&\to \cdots \to \texttt{apply}\langle \texttt{init} \mid 21 \rangle \\
&\to 21
\end{aligned}
$$

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

  答. Env は関数呼び出しを持たないので、値呼びかどうかを論じることができない。(ただし、let(x,e,e') の形の式については、e を計算して値にしてから x に対する束縛を発生させていることから、「値呼び」に準じていると言える。)

  Fun は値呼びである。なぜなら、App(e,e') で 関数ポジションにある e だけでなく、引数ポジションにある e' も先に計算して、値にしてから、束縛を発生させているからである。

  補足. なお、Env や Fun は、止まらない計算や副作用 (プリント文など) を含まないので、値呼びでも名前呼びでも必要呼びでも、計算結果は同じである。つまり、計算結果に着目する限り区別できない。

- 静的束縛 vs 動的束縛

  答. Env は関数呼び出しを持たないので、静的束縛かどうかを論じることができない。Fun は関数呼び出しを持つが、関数本体の中に自由変数 (その関数において束縛されたのでない変数) を持たないので、静的束縛と動的束縛の差が生じない。差が生じるのは、この資料の末尾の「宿題」に書いたプログラム等を与えたときである。

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。

名前呼び計算にするためには、関数呼び出し App(e,e') において、e' を計算せずに、引数に束縛して (環境に x=e' の形の束縛を追加し) 関数本体を呼びだすように変更することが必要である。ただし、e' は変数を含むかもしれないので、現在の環境 E とセットにして環境に記録しなければいけない。この「セット」を $\mathtt{susp}(e', E)$ と表すことにする。

$$\mathtt{apply}\langle \mathit{apply1}(e, E) :: K \mid v\rangle \to \mathtt{eval}\langle e_1 \mid [][x = \mathtt{susp}(e, E)] \mid K\rangle$$

ただし、$v = \mathtt{Func}(\mathtt{Lam}(x, e_1))$ であったとする。さらに、環境に含まれている x=e における e の形が「値」から $\mathtt{susp}(e', E)$ の形に変わったので、その形を取り出す規則 ($\mathtt{Var}(x)$ に対する計算規則) を以下のように変更する。

$$\mathtt{eval}\langle \mathtt{Var}(x) \mid E \mid K\rangle \to \mathtt{eval}\langle e \mid E' \mid K\rangle \quad (\text{if } \mathtt{susp}(e, E') = \mathtt{lookup}(x, E))$$

また、apply2 に関する遷移規則は不要となる。

これらの変更により、名前呼び計算が実現できていることは各自確かめよ。

3 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。

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

ヒント。このプログラムを Func と同様に処理すると、動的束縛になってしまう。静的束縛にしたいなら、関数 f の中の変数 x が、外側の (値 10 が束縛される)x でなければならない。