ソフトウェア基礎論配布資料 (12)
第一級継続

五十嵐 淳
京都大学 大学院情報学研究科知能情報学専攻
e-mail: igarashi@kuis.kyoto-u.ac.jp

平成 24 年 1 月 29 日

前章で扱った，継続を評価判断中に明示した操作的意味論での判断形式 $\boldsymbol{v_1} \Rightarrow \boldsymbol{k} \Downarrow \boldsymbol{v_2}$ は，計算の中間結果である値 $\boldsymbol{v_1}$ と，最終結果 $\boldsymbol{v_2}$ を，継続 $\boldsymbol{k}$ を関係づけている．与えられた継続 $\boldsymbol{k}$ と値 $\boldsymbol{v_1}$ に対して $\boldsymbol{v_2}$ は一意に決まることから，継続 $\boldsymbol{k}$ はある意味，関数を表現していると考えることができる．

本章では，プログラム中の任意の場所での継続を，関数として取り出す (もしくは捕捉する，ともいう) ためのプログラミング機構を導入する．取り出された継続は，あたかも普通の `fun` で構成される関数と同様に，適用したり，他の関数の引数として渡したり，リストに格納したりすることが可能である．このように，ある対象 (ここでは継続) が，ふつうの値と同様に制限なく扱えることを，その対象は**第一級の値・実体**(first-class value) である，といい，このように，関数として扱える継続を **(第) 一級継続**(first-class continuation) という．

ここでは，まず，EvalContML1 のような継続を評価判断中に明示した操作的意味論を関数，リストを扱う ML4 まで拡張し[1]，その上で，継続を関数値化する `letcc` 構文を導入する．第一級継続は，プログラム中の計算を中断して別の場所から再開する，といった,「計算の制御」に有用である．典型的な使用例として，繰り返し計算の途中からの脱出などを見る．

1 ML4 の継続に基づく操作的意味論

始めに ML4 に対する，継続に基づく操作的意味論を示す．基本的なアイデアは EvalContML1 と同様である．ただし，ML4 では，元の評価判断に環境が現れるのに伴い，継続 (が行う計算) の中身に式の評価が含まれる場合には，その式を評価するための環境も継続情報の一部として含める必要があるため，継続の BNF 定義は変更点が多くなっている．

[1] リストは第一級継続と必ずしも関連する機構ではない，いわば直交した機構だが，リストと第一級継続を組み合わせて意味のある応用例が記述できるため，ML3 ではなく ML4 に (第一級) 継続を導入する．

$i \in \mathbf{int}$
$b \in \mathbf{bool}$
$x, y \in \mathbf{Var}$
$v \in \mathbf{Value} ::= i \mid b \mid (\mathcal{E})[\texttt{fun } x \to e] \mid (\mathcal{E})[\texttt{rec } x \texttt{ = fun } y \to e] \mid [] \mid v :: v$
$\mathcal{E} \in \mathbf{Env} ::= \cdot \mid \mathcal{E}, x = v$
$e \in \mathbf{Exp} ::= i \mid b \mid x \mid e \; op \; e \mid \texttt{if } e \texttt{ then } e \texttt{ else } e \mid \texttt{let } x \texttt{ = } e \texttt{ in } e \mid \texttt{fun } x \to e \mid e \; e$
$\quad \mid \texttt{let rec } x \texttt{ = fun } y \to e \texttt{ in } e$
$\quad \mid [] \mid e :: e \mid \texttt{match } e \texttt{ with } [] \to e \texttt{ | } x :: y \to e$
$op \in \mathbf{Prim} ::= \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{<}$
$k \in \mathbf{Cont} ::= \_ \mid \{\mathcal{E} \vdash \_ \; op \; e\} \gg k \mid \{v \; op \; \_\} \gg k \mid \{\mathcal{E} \vdash \texttt{if } \_ \texttt{ then } e \texttt{ else } e\} \gg k$
$\quad \mid \{\mathcal{E} \vdash \texttt{let } x \texttt{ = } \_ \texttt{ in } e\} \gg k \mid \{\mathcal{E} \vdash \_ \; e\} \gg k \mid \{v \; \_\} \gg k$
$\quad \mid \{\mathcal{E} \vdash \_ :: e\} \gg k \mid \{v :: \_\} \gg k \mid \{\mathcal{E} \vdash \texttt{match } \_ \texttt{ with } [] \to e \texttt{ | } x :: y \to e\} \gg k$

$$\frac{i \Rightarrow k \Downarrow v}{\mathcal{E} \vdash i \gg k \Downarrow v} \quad \text{(E-Int)}$$

$$\frac{b \Rightarrow k \Downarrow v}{\mathcal{E} \vdash b \gg k \Downarrow v} \quad \text{(E-Bool)}$$

$$\frac{\mathcal{E} \vdash e_1 \gg \{\mathcal{E} \vdash \texttt{if } \_ \texttt{ then } e_2 \texttt{ else } e_3\} \gg k \Downarrow v}{\mathcal{E} \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \gg k \Downarrow v} \quad \text{(E-If)}$$

$$\frac{\mathcal{E} \vdash e_1 \gg \{\mathcal{E} \vdash \_ \; op \; e_2\} \gg k \Downarrow v}{\mathcal{E} \vdash e_1 \; op \; e_2 \gg k \Downarrow v} \quad \text{(E-BinOp)}$$

$$\frac{(\mathcal{E}(x) = v_1) \qquad v_1 \Rightarrow k \Downarrow v_2}{\mathcal{E} \vdash x \gg k \Downarrow v_2} \quad \text{(E-Var)}$$

$$\frac{\mathcal{E} \vdash e_1 \gg \{\mathcal{E} \vdash \texttt{let } x \texttt{ = } \_ \texttt{ in } e_2\} \gg k \Downarrow v}{\mathcal{E} \vdash \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 \gg k \Downarrow v} \quad \text{(E-Let)}$$

$$\frac{(\mathcal{E})[\texttt{fun } x \to e] \Rightarrow k \Downarrow v}{\mathcal{E} \vdash \texttt{fun } x \to e \gg k \Downarrow v} \quad \text{(E-Fun)}$$

$$\frac{\mathcal{E} \vdash e_1 \gg \{\mathcal{E} \vdash \_ \; e_2\} \gg k \Downarrow v}{\mathcal{E} \vdash e_1 \; e_2 \gg k \Downarrow v} \quad \text{(E-App)}$$

$$\frac{\mathcal{E}, x = (\mathcal{E})[\texttt{rec } x \texttt{ = fun } y \to e_1] \vdash e_2 \gg k \Downarrow v}{\mathcal{E} \vdash \texttt{let rec } x \texttt{ = fun } y \to e_1 \texttt{ in } e_2 \gg k \Downarrow v} \quad \text{(E-LetRec)}$$

$$\frac{\texttt{[]} \Rightarrow \boldsymbol{k} \Downarrow \boldsymbol{v}}{\boldsymbol{\mathcal{E}} \vdash \texttt{[]} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}} \quad \text{(E-Nil)}$$

$$\frac{\boldsymbol{\mathcal{E}} \vdash \boldsymbol{e_1} \gg \{\boldsymbol{\mathcal{E}} \vdash \texttt{\_ ::} \ \boldsymbol{e_2}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}}{\boldsymbol{\mathcal{E}} \vdash \boldsymbol{e_1} \texttt{ ::} \ \boldsymbol{e_2} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}} \quad \text{(E-Cons)}$$

$$\frac{\boldsymbol{\mathcal{E}} \vdash \boldsymbol{e_1} \gg \{\boldsymbol{\mathcal{E}} \vdash \texttt{match \_ with []} \rightarrow \boldsymbol{e_2} \texttt{ |} \ \boldsymbol{x} \texttt{ ::} \ \boldsymbol{y} \rightarrow \boldsymbol{e_3}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}}{\boldsymbol{\mathcal{E}} \vdash \texttt{match} \ \boldsymbol{e_1} \ \texttt{with []} \rightarrow \boldsymbol{e_2} \texttt{ |} \ \boldsymbol{x} \texttt{ ::} \ \boldsymbol{y} \rightarrow \boldsymbol{e_3} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}} \quad \text{(E-Match)}$$

$$\frac{}{\boldsymbol{v} \Rightarrow \texttt{\_} \Downarrow \boldsymbol{v}} \quad \text{(C-Ret)}$$

$$\frac{\boldsymbol{\mathcal{E}} \vdash \boldsymbol{e} \gg \{\boldsymbol{v_1} \ \boldsymbol{op} \ \texttt{\_}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v_2}}{\boldsymbol{v_1} \Rightarrow \{\boldsymbol{\mathcal{E}} \vdash \texttt{\_} \ \boldsymbol{op} \ \boldsymbol{e}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v_2}} \quad \text{(C-EvalR)}$$

$$\frac{\boldsymbol{i_1} \ \texttt{plus} \ \boldsymbol{i_2} \ \texttt{is} \ \boldsymbol{i_3} \qquad \boldsymbol{i_3} \Rightarrow \boldsymbol{k} \Downarrow \boldsymbol{v}}{\boldsymbol{i_2} \Rightarrow \{\boldsymbol{i_1} \texttt{ + \_}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}} \quad \text{(C-Plus)}$$

$$\frac{\boldsymbol{i_1} \ \texttt{minus} \ \boldsymbol{i_2} \ \texttt{is} \ \boldsymbol{i_3} \qquad \boldsymbol{i_3} \Rightarrow \boldsymbol{k} \Downarrow \boldsymbol{v}}{\boldsymbol{i_2} \Rightarrow \{\boldsymbol{i_1} \texttt{ - \_}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}} \quad \text{(C-Minus)}$$

$$\frac{\boldsymbol{i_1} \ \texttt{times} \ \boldsymbol{i_2} \ \texttt{is} \ \boldsymbol{i_3} \qquad \boldsymbol{i_3} \Rightarrow \boldsymbol{k} \Downarrow \boldsymbol{v}}{\boldsymbol{i_2} \Rightarrow \{\boldsymbol{i_1} \texttt{ * \_}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}} \quad \text{(C-Times)}$$

$$\frac{\boldsymbol{i_1} \ \texttt{less than} \ \boldsymbol{i_2} \ \texttt{is} \ \boldsymbol{b_3} \qquad \boldsymbol{b_3} \Rightarrow \boldsymbol{k} \Downarrow \boldsymbol{v}}{\boldsymbol{i_2} \Rightarrow \{\boldsymbol{i_1} \texttt{ < \_}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}} \quad \text{(C-Lt)}$$

$$\frac{\boldsymbol{\mathcal{E}} \vdash \boldsymbol{e_1} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}}{\texttt{true} \Rightarrow \{\boldsymbol{\mathcal{E}} \vdash \texttt{if \_ then} \ \boldsymbol{e_1} \ \texttt{else} \ \boldsymbol{e_2}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}} \quad \text{(C-IfT)}$$

$$\frac{\boldsymbol{\mathcal{E}} \vdash \boldsymbol{e_2} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}}{\texttt{false} \Rightarrow \{\boldsymbol{\mathcal{E}} \vdash \texttt{if \_ then} \ \boldsymbol{e_1} \ \texttt{else} \ \boldsymbol{e_2}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}} \quad \text{(C-IfF)}$$

$$\frac{\boldsymbol{\mathcal{E}}, \boldsymbol{x} = \boldsymbol{v_1} \vdash \boldsymbol{e} \gg \boldsymbol{k} \Downarrow \boldsymbol{v_2}}{\boldsymbol{v_1} \Rightarrow \{\boldsymbol{\mathcal{E}} \vdash \texttt{let} \ \boldsymbol{x} \texttt{ = \_ in} \ \boldsymbol{e}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v_2}} \quad \text{(C-LetBody)}$$

$$\frac{\boldsymbol{\mathcal{E}} \vdash \boldsymbol{e} \gg \{\boldsymbol{v_1} \texttt{ \_}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}}{\boldsymbol{v_1} \Rightarrow \{\boldsymbol{\mathcal{E}} \vdash \texttt{\_} \ \boldsymbol{e}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}} \quad \text{(C-EvalArg)}$$

$$\frac{\boldsymbol{\mathcal{E}}, \boldsymbol{x} = \boldsymbol{v_1} \vdash \boldsymbol{e} \gg \boldsymbol{k} \Downarrow \boldsymbol{v_2}}{\boldsymbol{v_1} \Rightarrow \{(\boldsymbol{\mathcal{E}})\texttt{[fun} \ \boldsymbol{x} \rightarrow \boldsymbol{e}\texttt{]} \ \texttt{\_}\} \gg \boldsymbol{k} \Downarrow \boldsymbol{v_2}} \quad \text{(C-EvalFun)}$$

$$\frac{\mathcal{E}, x = (\mathcal{E})[\texttt{rec } x \texttt{ = fun } y \to e], y = v_1 \vdash e \gg k \Downarrow v_2}{v_1 \Rightarrow \{(\mathcal{E})[\texttt{rec } x \texttt{ = fun } y \to e]\ \_\} \gg k \Downarrow v_2} \quad \text{(C-EvalFunR)}$$

$$\frac{\mathcal{E} \vdash e \gg \{v_1 :: \_\} \gg k \Downarrow v_2}{v_1 \Rightarrow \{\mathcal{E} \vdash \_ :: e\} \gg k \Downarrow v_2} \quad \text{(C-EvalConsR)}$$

$$\frac{v_1 :: v_2 \Rightarrow k \Downarrow v_3}{v_2 \Rightarrow \{v_1 :: \_\} \gg k \Downarrow v_3} \quad \text{(C-Cons)}$$

$$\frac{\mathcal{E} \vdash e_1 \gg k \Downarrow v}{[] \Rightarrow \{\mathcal{E} \vdash \texttt{match \_ with } [] \to e_1 \texttt{ | } x :: y \to e_2\} \gg k \Downarrow v} \quad \text{(C-MatchNil)}$$

$$\frac{\mathcal{E}, x = v_1, y = v_2 \vdash e_2 \gg k \Downarrow v}{v_1 :: v_2 \Rightarrow \{\mathcal{E} \vdash \texttt{match \_ with } [] \to e_1 \texttt{ | } x :: y \to e_2\} \gg k \Downarrow v} \quad \text{(C-MatchCons)}$$

(規則 B-Plus など，二項演算に関する 4 つの規則は省略している．)

2 継続の捕捉

継続を捕捉するための構文は `letcc` と呼ばれるものである．この `cc` は current continuation，すなわち「現在の継続」の略で，他の言語では call/cc, call-with-current-continuation とも呼ばれる機構 (の変形版) である．`letcc` は，

$$\texttt{letcc } x \texttt{ in } e$$

という形で用いる．この意味は，直感的には，変数 x を，式 e の値が渡されるであろう継続 (これが current continuation である) を関数値化したものに束縛した上で，e を評価する，という意味である．x の有効範囲は式 e である．ここで捕捉される継続の内容は e には依存せずに，`letcc` の**外側にある式の形で決まる**．例えば，

```
(1 + 2) + ((letcc f in ...) + 4)
```

という式で `f` が束縛される継続関数は，`fun x → 3 + (x + 4)` 相当のものである．(評価が `letcc` に到達した時点で `1 + 2` は既に計算されて値になっているため，`1 + 2` は `3` に置き換えられている．)

この，`letcc` で捕捉された関数を呼び出すと，あたかも，`letcc f in ...` の評価が済んだ直後から計算を再開したような効果を得ることができる．例えば，

```
(1 + 2) + ((letcc f in f (1 + 2) + 4 + 5) + 6)
```

の値は，fun x → 3 + (x + 6) を 1 + 2 の値である 3 に適用した，12 になる．(関数適用と足し算の優先度に注意．) これを，letcc と継続の呼び出しを消去した式

```
(1 + 2) + ((1 + 2) + 4 + 5) + 6)
```

と比べてみると，あたかも + 4 + 5 という計算が「飛ばされて」いるかのようである．

letcc を使うことで，計算途中で計算を続けられない・続ける必要がないような例外的な状況 (例えば 0 での除算が発生しそうになった，などの状況) で，残りの計算を飛ばして，適当な時点から計算を再開する，といったプログラムを書くことができる．これを，整数のリストを受け取って，その整数の積を計算するプログラムを例に取って説明する．

最初は再帰的関数を使った素直な定義である．

```
let rec prod = fun n ->
  match n with [] -> 1 | x :: y -> x * prod y
in prod (3 :: 4 :: 5 :: [])
```

この式の値は，結局，リストの長さ (:: の数) に比例した回数の乗算，すなわち 3 * 4 * 5 * 1, を行って 60 となる．

これは途中に 0 が挟まっても同じで，

```
let rec prod = fun n ->
  match n with [] -> 1 | x :: y -> x * prod y
in prod (3 :: 0 :: 5 :: [])
```

は，3 * 0 * 5 * 1 を計算して 0 になる．

次は，リストの要素に 0 を見つけた場合の処理を追加した定義である．

```
let eq = fun x -> fun y -> if x < y + 1 then y < x + 1 else false in
let rec prod = fun n ->
  match n with
     [] -> 1
   | x :: y -> if eq 0 x then 0 else x * prod y
in prod (3 :: 0 :: 5 :: [])
```

この場合，確かに prod (5 :: [])，すなわち 5 * 1 の計算をせずに 0 を返すので，無駄な計算は除かれているのだが，相変わらず 3 * 0 という無駄な計算が残っている．

最後に，letcc を使って，0 を発見したら，一番外側の prod の呼び出しが終わった時点にジャンプするようなプログラムにしたのが以下の例である．

```
let eq = fun x -> fun y -> if x < y + 1 then y < x + 1 else false in
let prod = fun l ->
  letcc k in
    let rec prodaux = fun l ->
      match l with
        [] -> 1
      | n :: l' -> if eq 0 n then k 0 else n * prod l'
    in prodaux l
in
prod (3 :: 0 :: 5 :: [])
```

この関数定義は二段構えになっている．まず prod (これは再帰的には定義されていない) が呼びだされると，その時点での継続 (つまり prod の返り値を受け取ったあとの計算) が k に捕捉される．内側の prodaux が，上の prod 相当の再帰的関数の定義である．但し，リスト中の先頭要素が 0 だった場合には 0 を返すのではなく，letcc で捕捉された継続を呼び出している．これにより，0 以前に現れるリストの要素に関するかけ算を飛ばして，いきなり prod は 0 を返すことになる．

3 導出システム EvalContML4

導出システム EvalContML4 は，letcc を導入した ML4 の操作的意味論を表現したシステムである．値として，継続を関数化した一級継続が加わる．一級継続は，単に継続そのもの (を区切りのための [] で囲って) $[\boldsymbol{k}]$ で表す．

$$\begin{aligned}\boldsymbol{v} \in \mathbf{Value} &::= \cdots \mid [\boldsymbol{k}] \\ \boldsymbol{e} \in \mathbf{Exp} &::= \cdots \mid \texttt{letcc}\ \boldsymbol{x}\ \texttt{in}\ \boldsymbol{e}\end{aligned}$$

以下が第一級継続に関連する推論規則である．

$$\frac{\boldsymbol{\mathcal{E}}, \boldsymbol{x} = [\boldsymbol{k}] \vdash \boldsymbol{e} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}}{\boldsymbol{\mathcal{E}} \vdash \texttt{letcc}\ \boldsymbol{x}\ \texttt{in}\ \boldsymbol{e} \gg \boldsymbol{k} \Downarrow \boldsymbol{v}} \qquad \text{(E-LetCc)}$$

$$\frac{\boldsymbol{v_1} \Rightarrow \boldsymbol{k_1} \Downarrow \boldsymbol{v_2}}{\boldsymbol{v_1} \Rightarrow \{[\boldsymbol{k_1}]\ \_\} \gg \boldsymbol{k_2} \Downarrow \boldsymbol{v_2}} \qquad \text{(C-EvalFunC)}$$

規則 E-LetCc は，letcc 式の値は $\boldsymbol{x}$ を現在の継続 $\boldsymbol{k}$ に束縛して $\boldsymbol{e}$ を評価した値と同じであることを示している．捕捉した継続が関数適用の形で呼び出された場合の状況は C-EvalFunC で表現されている．この規則の結論では，関数呼び出しの後の (本来の) 継続は $\boldsymbol{k_2}$ であることを示しているが，それを「捨てて」，$\boldsymbol{k_1}$ を新しい継続として計算を行った時の値が，継続関数を呼び出した結果の値であることを示している．