ソフトウェア基礎論配布資料 (13)
参照

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

1 参照と代入

参照(reference) とは，計算機のメモリアドレスを抽象化した概念であり，値を格納するための「箱」，値を指すためのポインタのようなものである．参照は，格納した値，指している値を新しい値に更新することができる．この更新の操作は**代入**(assignment) とも呼ばれている．この意味で参照は，C 言語などの伝統的な言語における変数やポインタの概念に近いが，C 言語のポインタとは違い，アドレスの加算などのいわゆるポインタ演算は行うことができない．また，C 言語では初期化されていない変数，何も指していないポインタ (null ポインタ) などが生じるが，ML では，参照を生成する時に，常に箱に入る初期値を与えることになっており，何も指していない参照は生じない．

ML では，`ref` という特殊な関数で参照を生成する．`ref` 関数は，参照先に格納する初期値を引数とし，それを格納したアドレスを返す．この関数は通常の数学的・集合論的な意味での関数とは違い，呼び出す度に新しい値 (アドレス) を返す．参照に格納された値を取り出す (これを dereference という) ためには，前置オペレータ `!` を使用する．`!`⟨**式**⟩ の値は，⟨**式**⟩を評価して得られた参照値の指す先の値になる．そして，代入は中置演算子 `:=` を用いて行う．

⟨**式**$_1$⟩ `:=` ⟨**式**$_2$⟩

で，⟨**式**$_1$⟩を評価した結果の参照に⟨**式**$_2$⟩の値を代入する．この代入も式の一種であるため値を持つ．OCaml ではユニット値 (“`()`” と表記される) という，(特に使い途のない) 特別な値を返すことになっている．以下は，`1` を指す参照を生成して，その中身を `1` だけ増やす例である．

```
let x = ref 1 in
let z = x := !x + 1 in
!x
```

の値は 2 になる．(let z = は，単に代入をした後に参照の中身を取り出す，という評価の順序付けのために使っており，z は，使用しないダミーの変数である．実際の ML では，「⟨**式**$_1$⟩ を評価して (その値を捨てて)⟨**式**$_2$⟩の値を評価する」という意味の，逐次評価のための

$$\langle \textbf{式}_1 \rangle ; \langle \textbf{式}_2 \rangle$$

という構文が導入されている．)

参照を理解する上では，参照値とその中身を厳格に区別することが大切である．上の式は C 言語で書くならば，

```
int x = 1;
x = x + 1;
return x;
```

に相当するようなコード片である．C 言語では代入 x = x + 1 の左辺は暗黙のうちに変数 x のアドレスを表し，右辺では x の内容を示しているが，ML では，それらを ! を使って明確に区別している．

ML において，参照は関数の実引数として渡すこともでき，また，リストなどデータ構造に格納することもできる第一級の値である．さらに，参照が指すことができる値にも制限がなく，整数・真偽値への参照だけでなく，関数への参照，参照への参照といった参照値も扱うことができる．

特に参照の参照は，C 言語におけるポインタに近く，

```
int *p, *q; ...
*p = *q;
```

のような，ポインタ p の指す先に q を代入するような代入は，

```
!p := !(!q)
```

のような代入式で表現することができる．

2　導出システム EvalRefML3

このような，参照を含む ML の評価の意味論を導出システムとしたのが EvalRefML3 である．ここでは，単純のためリストを含まない ML3 に対して参照に関する機構を加える．また，代入式の値であるユニット値を加えるのは煩雑なため，代入式の値は右辺の値である，とする．

参照の意味を捉えるためには，**式を評価する各時点での**参照が格納している値 (の一覧) を把握する必要がある．この一覧は，いわばメモリの状態を表しているもので，**ストア**(store) と呼ばれる．ストアは，アドレスを表す**ロケーション**(location)—参照値の表現でもある—と値の対応を列挙したものとして表される．変数の値を表す環境と構造は似ているが，環境は変数の有効範囲に応じた局所的な情報を表しているのに対して，ストアは，ロケーション自

体が値，つまり関数を通じてやりとりされるものであるため，プログラム全体で大域的に使われる情報を表しているといえる．

以下に BNF による EvalRefML3 の構文定義を示す．

$\boldsymbol{i} \in \mathbf{int}$
$\boldsymbol{b} \in \mathbf{bool}$
$\boldsymbol{x}, \boldsymbol{y} \in \mathbf{Var}$
$\boldsymbol{l} \in \mathbf{Loc}$
$\boldsymbol{v} \in \mathbf{Value} ::= \boldsymbol{i} \mid \boldsymbol{b} \mid \boldsymbol{l} \mid (\boldsymbol{\mathcal{E}})\texttt{[fun } \boldsymbol{x} \texttt{ -> } \boldsymbol{e}\texttt{]} \mid (\boldsymbol{\mathcal{E}})\texttt{[rec } \boldsymbol{x} \texttt{ = fun } \boldsymbol{y} \texttt{ -> } \boldsymbol{e}\texttt{]}$
$\boldsymbol{\mathcal{E}} \in \mathbf{Env} ::= \bullet \mid \boldsymbol{\mathcal{E}}, \boldsymbol{x} = \boldsymbol{v}$
$\boldsymbol{S} \in \mathbf{Store} ::= \bullet \mid \boldsymbol{S}, \boldsymbol{l} = \boldsymbol{v}$
$\boldsymbol{e} \in \mathbf{Exp} ::= \boldsymbol{i} \mid \boldsymbol{b} \mid \boldsymbol{x} \mid \boldsymbol{e}\ \boldsymbol{op}\ \boldsymbol{e} \mid \texttt{if } \boldsymbol{e} \texttt{ then } \boldsymbol{e} \texttt{ else } \boldsymbol{e} \mid \texttt{let } \boldsymbol{x} \texttt{ = } \boldsymbol{e} \texttt{ in } \boldsymbol{e}$
$\quad \mid \texttt{fun } \boldsymbol{x} \texttt{ -> } \boldsymbol{e} \mid \boldsymbol{e}\ \boldsymbol{e} \mid \texttt{let rec } \boldsymbol{x} \texttt{ = fun } \boldsymbol{y} \texttt{ -> } \boldsymbol{e} \texttt{ in } \boldsymbol{e} \mid \texttt{ref } \boldsymbol{e} \mid \texttt{!}\boldsymbol{e} \mid \boldsymbol{e} \texttt{ := } \boldsymbol{e}$
$\boldsymbol{op} \in \mathbf{Prim} ::= \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{<}$

代入の `:=` は右結合で，最も優先度が低い中置演算子である．また `!` は結合が最も強く，例えば `!f 1` は `!(f 1)` ではなく，`(!f) 1` という意味である．ロケーションは，メタ変数 $\boldsymbol{l}$ で表すが，具体的なロケーションの名前としては，英数字名の前に `@` をつけた，`@l` や `@l1` などを使う．ストア (メタ変数 $\boldsymbol{S}$) は，“$\boldsymbol{l} = \boldsymbol{v}$” という形の表現の列である．環境と同様，(特に演習システムでは，) 先頭の “$\cdot$,” は省略する．

EvalRefML3 の評価の判断は，

$$\boldsymbol{S_1} \ /\ \boldsymbol{\mathcal{E}} \vdash \boldsymbol{e} \Downarrow \boldsymbol{v} \ /\ \boldsymbol{S_2}$$

という形式で表し，「ストア $\boldsymbol{S_1}$，環境 $\boldsymbol{\mathcal{E}}$ のもとで式 $\boldsymbol{e}$ の値は $\boldsymbol{v}$ であり，評価の副作用としてストアが $\boldsymbol{S_2}$ に変化する」という意味である．**副作用**(side effect) ということばは，値を得ることが式を評価することの主たる結果とすると，ストアの変化が，それ以外に発生する事柄であるということを示している．他にも一般のプログラムでは，ディスプレイへの出力やキーボード・マウスからの入力など，様々な「状態変化」が副作用[1] として発生する．演習システムでは，ストアが空の場合，先頭の “$\boldsymbol{S}$ /” と末尾の “/ $\boldsymbol{S}$” は省略可能である．

以下に導出規則 (の一部) を示す．

$$\frac{}{\boldsymbol{S} \ /\ \boldsymbol{\mathcal{E}} \vdash \boldsymbol{i} \Downarrow \boldsymbol{i} \ /\ \boldsymbol{S}} \quad \text{(E-Int)}$$

$$\frac{\boldsymbol{S_1} \ /\ \boldsymbol{\mathcal{E}} \vdash \boldsymbol{e_1} \Downarrow \boldsymbol{i_1} \ /\ \boldsymbol{S_2} \qquad \boldsymbol{S_2} \ /\ \boldsymbol{\mathcal{E}} \vdash \boldsymbol{e_2} \Downarrow \boldsymbol{i_2} \ /\ \boldsymbol{S_3} \qquad \boldsymbol{i_1} \texttt{ plus } \boldsymbol{i_2} \texttt{ is } \boldsymbol{i_3}}{\boldsymbol{S_1} \ /\ \boldsymbol{\mathcal{E}} \vdash \boldsymbol{e_1} \texttt{ + } \boldsymbol{e_2} \Downarrow \boldsymbol{i_3} \ /\ \boldsymbol{S_3}} \quad \text{(E-Plus)}$$

[1]「副作用」という言葉を使うと，日常生活での使われ方から「副作用」の発生は悪いことであるように思えるかもしれないが，特にそういうわけではなく，単に，主たる効果が，値の計算・実行の終了であるとした時には，そう呼べる，といういわば相対的な概念である．もっと中立的な響きのある**計算効果**(computational effect) という用語を使うことも多くなってきている．

$$\frac{\boldsymbol{S_1} \ / \ \boldsymbol{\mathcal{E}} \vdash \boldsymbol{e} \Downarrow \boldsymbol{v} \ / \ \boldsymbol{S_2} \qquad (\boldsymbol{l} \notin \mathit{dom}(\boldsymbol{S_2}))}{\boldsymbol{S_1} \ / \ \boldsymbol{\mathcal{E}} \vdash \texttt{ref}\ \boldsymbol{e} \Downarrow \boldsymbol{l} \ / \ \boldsymbol{S_2}, \boldsymbol{l} = \boldsymbol{v}} \qquad \text{(E-Ref)}$$

$$\frac{\boldsymbol{S_1} \ / \ \boldsymbol{\mathcal{E}} \vdash \boldsymbol{e} \Downarrow \boldsymbol{l} \ / \ \boldsymbol{S_2} \qquad (\boldsymbol{S_2}(\boldsymbol{l}) = \boldsymbol{v})}{\boldsymbol{S_1} \ / \ \boldsymbol{\mathcal{E}} \vdash \texttt{!}\boldsymbol{e} \Downarrow \boldsymbol{v} \ / \ \boldsymbol{S_2}} \qquad \text{(E-Deref)}$$

$$\frac{\boldsymbol{S_1} \ / \ \boldsymbol{\mathcal{E}} \vdash \boldsymbol{e_1} \Downarrow \boldsymbol{l} \ / \ \boldsymbol{S_2} \qquad \boldsymbol{S_2} \ / \ \boldsymbol{\mathcal{E}} \vdash \boldsymbol{e_2} \Downarrow \boldsymbol{v} \ / \ \boldsymbol{S_3} \qquad (\boldsymbol{S_4} = \boldsymbol{S_3}[\boldsymbol{l} = \boldsymbol{v}])}{\boldsymbol{S_1} \ / \ \boldsymbol{\mathcal{E}} \vdash \boldsymbol{e_1} \ \texttt{:=}\ \boldsymbol{e_2} \Downarrow \boldsymbol{v} \ / \ \boldsymbol{S_4}} \qquad \text{(E-Assign)}$$

前提のある規則については，ストアの現れ方によって，部分式の評価を行う順序が明示化されていることが特徴である．(EvalContMLi でも明示化されていたが．) また，参照に関わる規則 E-Ref，E-Deref，E-Assign では，(最後の) 部分式の評価が終わった後のストアと，結論の末尾のストアが異なっており，状態変化という副作用の発生を表している．E-Assign に現れる記法 $\boldsymbol{S}[\boldsymbol{l} = \boldsymbol{v}]$ は，$\boldsymbol{S}$ に現れる $\boldsymbol{l} = \ldots$ を $\boldsymbol{l} = \boldsymbol{v}$ で置き換えたようなストアであり，正確には以下のように定義される．

$$\begin{aligned}
(\boldsymbol{S}, \boldsymbol{l} = \boldsymbol{v})[\boldsymbol{l} = \boldsymbol{v'}] &= \boldsymbol{S}, \boldsymbol{l} = \boldsymbol{v'} \\
(\boldsymbol{S}, \boldsymbol{l} = \boldsymbol{v})[\boldsymbol{l'} = \boldsymbol{v'}] &= (\boldsymbol{S}[\boldsymbol{l'} = \boldsymbol{v'}]), \boldsymbol{l} = \boldsymbol{v} \qquad (\text{if } \boldsymbol{l} \neq \boldsymbol{l'})
\end{aligned}$$

($\boldsymbol{S}$ が $\bullet$ の場合の定義が書かれていないことから，$\boldsymbol{S}$ に $\boldsymbol{l} = \ldots$ が現れない時には，$\boldsymbol{S}[\boldsymbol{l} = \boldsymbol{v}]$ は未定義となる．)