13 章 参照

水野 雅之

2015 年 6 月 16 日

アウトライン

動機

これまで純粋な言語機能について議論してきたが、実用的な言語に含まれるような純粋でない機能についても議論したい

ここでは副作用（より一般的に計算的作用と言われる）の一種である書き換え可能な参照を扱う

アウトライン

導入

ML 系言語は変数束縛と参照は分けて扱う

- `let x = 5 and y = ref 5`

C 系の言語では特に指定しなければ変数は書き換え可能なセルを持つ

- `int x = 5; x = 84;`

ここでは前者のアプローチを採用する

参照の基本

参照に対しての基本的な操作

- 書き換え可能なセルの確保 `ref`
- dereference（参照セルから値の取り出し） `!`
- 代入 `:=`

```
r = ref 5;
- r : Ref Nat
!r;
- 5 : Nat
r := 7;
- unit : Unit
!r;
- 7 : Nat
```

だいたい ML と同じ挙動

副作用と逐次実行

代入が unit に評価されることは逐次実行と相性がよい

```
(r := succ (!r); !r);
- 8 : Nat
```

一つ目の式が Unit 型であることを要請したおかげでデバッグで幸せになれる
二つ目の式も代入の場合全体の型は Unit となるため、逐次実行の列を作ることもできる

```
(r:=succ(!r);r:=succ(!r);r:=succ(!r);r:=succ(!r);!r);
- 13 : Nat
```

参照とエイリアス

r に束縛された参照とその参照が指すセルの違いに注意
r をコピーしても、中身まではコピーされない

```
s = r;
- s : Ref Nat
```

従って、s に代入すると r も書きかわる

```
s := 82;
- unit : Unit
!r;
- 82 : Nat
```

p.155 と p.156 の図も参照
このとき、s と r は同じセルのエイリアスという

共有状態

エイリアスがあるとプログラムの性質が証明しづらい

- (r:=1;r:=!s) と r:=!s は同じように振る舞うか？

とはいえ便利で、プログラムの異なる部分に「暗黙の通信チャンネル」を作れたりする
例えばこういう風に参照セルと関数を作ると

```
c = ref 0;
- c : Ref Nat
incc = \x:Unit. (c := succ (!c); !c);
- incc : Unit -> Nat
decc = \x:Unit. (c := pred (!c); !c);
- decc : Unit -> Nat
```

共有状態

```
incc unit;
- 1 : Nat
decc unit;
- 0 : Nat
```

incc を呼び出すと c を変更でき、decc を呼び出すとそれが分かる

incc と decc をレコードにまとめると他の所に渡しやすい

```
o = {i = incc, d = decc};
- o : {i : Unit -> Nat, d : Unit -> Nat}
```

これは 18 章で詳しく見るオブジェクトを作ったともいえる

複合的な型の参照

参照が取り扱えるのは数値に限らない
例えば関数への参照を使うと、配列を実装できる

```
NatArray = Ref (Nat -> Nat);
newarray = \_ : Unit. ref (\n : Nat. 0);
- newarray : Unit -> NatArray
lookup = \a : NatArray. \n : Nat. (!a) n;
- lookup : NatArray -> Nat -> Nat
update = \a : NatArray. \m : Nat. \v : Nat.
            let oldf = !a in
            a := (\n : Nat. if equal m n then v
                            else oldf n);
- update : NatArray -> Nat -> Nat -> Unit
```

ガベージコレクション

必要が無くなったセルを解放するプリミティブがないが、現代的な言語では GC により解決している
設計者の趣味ではなく、明示的な解放操作があると型安全の達成が非常に難しいため

ぶら下がり参照問題

- 数値を保持した参照セルを明示的に解放
- 真偽値を保持した参照セルを確保
- 同じ位置のメモリが使用されるとどうなるか?

アウトライン

型付け

`ref`、 `:=`と`!`の型付けは自明

$$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \texttt{ref}\ t_1 : \texttt{Ref}\ T_1}\ \text{T-Ref}$$

$$\frac{\Gamma \vdash t_1 : \texttt{ref}\ T_1}{\Gamma \vdash\ !t_1 : T_1}\ \text{T-Deref}$$

$$\frac{\Gamma \vdash t_1 : \texttt{ref}\ T_1 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1 := t_2 : \texttt{Unit}}\ \text{T-Assign}$$

アウトライン

評価関係の拡張

ストア μ はポインタを表すロケーション l を、値に写す部分関数として表す

項の値はストアに依存し、評価の前後でストアは変化しうるため、評価関係を拡張する

$$t|\mu \to t'|\mu'$$

既存の規則はごく自然に拡張できる

$$(\lambda x : T_{11}.t_{12})\ v_2|\mu \to [x \mapsto v_2]t_{12}|\mu\ \text{E-AppAbs}$$

$$\frac{t_1|\mu \to t_1'|\mu'}{t_1\ t_2|\mu \to t_1'\ t_2|\mu'}\ \text{E-App1}$$

$$\frac{t_1|\mu \to t_1'|\mu'}{t_1\ t_2|\mu \to t_1'\ t_2|\mu'}\ \text{E-App2}$$

値

`ref` t の値は新しく確保したセルのロケーションなので、ロケーションも値とする

$v ::=$		値:
	$\lambda x : T.\ t$	関数抽象
	`unit`	unit
	l	ロケーション

項

small-step semantics を採用するため、計算の途中結果も項

$t ::=$		項:
	x	変数
	$\lambda x : T.\ t$	関数抽象
	$t\ t$	関数適用
	$\texttt{unit}$	unit
	$\texttt{ref}\ t$	参照セルの作成
	$!t$	値の取り出し
	$t := t$	代入
	l	ロケーション

dereference の評価

項がロケーションになるまで評価する

$$\frac{t_1|\mu \to t_1'|\mu'}{!t_1|\mu \to !t_1'|\mu'}\ \text{E-Deref}$$

dereference の結果はロケーションが指す位置のストアの値

$$\frac{\mu(l) = v}{!l|\mu \to v|\mu}\ \text{E-DerefLoc}$$

代入の評価

代入先がロケーションになるまで評価する

$$\frac{t_1|\mu \to t_1'|\mu'}{t_1 := t_2|\mu \to t_1' := t_2|\mu'}\ \text{E-Assign1}$$

代入する値を求める

$$\frac{t_2|\mu \to t_2'|\mu'}{v_1 := t_2|\mu \to v_1 := t_2'|\mu'}\ \text{E-Assign2}$$

ロケーションが指すセルの値を書き換える

$$l := v_2|\mu \to \texttt{unit}|[l \mapsto v_2]\mu\ \text{E-Assign}$$

ref の評価

初期値が値になるまで評価する

$$\frac{t_1|\mu \to t_1'|\mu'}{\texttt{ref}\ t_1|\mu \to \texttt{ref}\ t_1'|\mu'}\ \text{E-Ref}$$

ref は fresh なロケーションを返す必要がある

$$\frac{l \notin \texttt{dom}(\mu)}{\texttt{ref}\ v_1|\mu \to l|(\mu, l \mapsto v_1)}\ \text{E-RefV}$$

GC は形式的には違いが無いので扱いません

アウトライン

ロケーションの型付け

ロケーションの型は？
ストアを見ないと分からない

$$\frac{\Gamma \vdash \mu(l) : T_1}{\Gamma \vdash l : \texttt{Ref}\ T_1}$$

項の型はストアにも依存するので、型付け関係を拡張

$$\frac{\Gamma | \mu \vdash \mu(l) : T_1}{\Gamma | \mu \vdash l : \texttt{Ref}\ T_1}$$

しかし、この型付け規則にはいくつか問題がある

問題点

いちいち $\mu(l)$ の型を求めるので非効率

$$
\begin{aligned}
(l_1 &\mapsto \lambda x : \mathtt{Nat}.\ 999, \\
l_2 &\mapsto \lambda x : \mathtt{Nat}.\ (!l_1)\ x, \\
l_3 &\mapsto \lambda x : \mathtt{Nat}.\ (!l_2)\ x, \\
l_4 &\mapsto \lambda x : \mathtt{Nat}.\ (!l_3)\ x, \\
l_5 &\mapsto \lambda x : \mathtt{Nat}.\ (!l_4)\ x)
\end{aligned}
$$

循環した構造に型を付けられない

$$
\begin{aligned}
(l_1 &\mapsto \lambda x : \mathtt{Nat}.\ (!l_2)\ x, \\
l_2 &\mapsto \lambda x : \mathtt{Nat}.\ (!l_1)\ x),
\end{aligned}
$$

ストア型付け

ロケーションの型は ref で作られた時に決まっている
ロケーションの型の情報を集めるストア型付け μ を導入

$$\frac{\Sigma(l) = T_1}{\Gamma|\Sigma \vdash l : \texttt{Ref}\ T_1}\ \text{T-Loc}$$

$$\frac{\Gamma|\Sigma \vdash t_1 : T_1}{\Gamma|\Sigma \vdash \texttt{ref}\ t_1 : \texttt{Ref}\ T_1}\ \text{T-Ref}$$

ロケーションは評価された時にしか増えない

$$\frac{\Gamma|\Sigma \vdash t_1 : \texttt{ref}\ T_1}{\Gamma\Sigma \vdash\ !t_1 : T_1}\ \text{T-Deref}$$

$$\frac{\Gamma|\Sigma \vdash t_1 : \texttt{ref}\ T_1 \qquad \Gamma|\Sigma \vdash t_2 : T_1}{\Gamma|\Sigma \vdash t_1 := t_2 : \texttt{Unit}}\ \text{T-Assign}$$

アウトライン

安全性

well-typed な項はスタックしないことを証明したい
型付け関係と評価関係を拡張したため、それに応じた修正が必要

定義 13.5.1

$\mathtt{dom}(\mu) = \mathtt{dom}(\Sigma)$ かつ全ての $l \in \mathtt{dom}(\mu)$ が成り立つような l に対して $\Gamma|\Sigma \vdash \mu(l) : \Sigma(l)$ が成り立つとき、ストア μ は型環境 Γ とストア型付け Σ の下で well-typed であるといい、$\Gamma|\Sigma \vdash \mu$ と書く。

代入補題

補題 13.5.4 (代入補題)

$\Gamma, x : S | \Sigma \vdash t : T$ かつ $\Gamma | \Sigma \vdash s : S$ ならば
$\Gamma | \Sigma \vdash [x \mapsto s]t : T$

Proof.

補題 9.3.8 と同様に証明できる。
substitution は 13 章で追加された項に対しては特別な操作は行わないため、今回追加された型付け規則の場合も T-App などの場合と同様に証明できる。 □

ストア型付けに対する代入補題

補題 13.5.5 (ストア型付けに対する代入補題)

$$(\Gamma|\Sigma \vdash \mu) \wedge (\Sigma(l) = T) \wedge (\Gamma|\Sigma \vdash v : T) \Rightarrow (\Gamma|\Sigma \vdash [l \mapsto v]\mu)$$

Proof.

$\Gamma|\Sigma \vdash v : T$ の定義を振り返ると、結論は $\mathtt{dom}(\mu) = \mathtt{dom}(\Sigma)$ と $\forall l' \in \mathtt{dom}(\Gamma).\ [l \mapsto v]\mu(l') : \Sigma(l')$ が成り立てば満たされることが分かる。
このうち、前者は仮定 $\Gamma|\Sigma \vdash \mu$ より明らか。
後者も $l = l'$ が成り立つ場合とそうでない場合に分けることで示せる。 □

ストア型付けの弱化

補題 13.5.6 (ストア型付けの弱化)

$\Gamma|\Sigma \vdash t : T$ かつ $\Sigma \subseteq \Sigma'$ ならば $\Gamma|\Sigma' \vdash t : T$

Proof.

型付けの導出に関する帰納法で証明できる。
Γ について言及している規則が殆ど無いので自明。
今回の定義した型付け規則に関する反転補題も用いるので、適宜証明する。 □

preservation

定理 13.5.3 (preservation)

$$(\Gamma|\Sigma \vdash t : T) \land (\Gamma|\Sigma \vdash \mu) \land (t|\mu \to t'|\mu')$$
$$\Rightarrow \exists \Sigma'.\ (\Sigma \subseteq \Sigma') \land (\Gamma|\Sigma' \vdash t' : T) \land (\Gamma|\Sigma' \vdash \mu')$$

Proof.

$t|\mu \to t'|\mu'$ の導出に関する帰納法で証明する。

E-AppAbs の場合

$t = (\lambda x : T_{11}.\ t_{12})\ v_2 \qquad t' = [x \mapsto v_2]t_{12} \qquad \mu = \mu'$

仮定 $\Gamma|\Sigma \vdash (\lambda x : T_{11}.\ t_{12})\ v_2 : T$ に反転補題を適用する

$\Gamma, x : T_{11}|\Sigma \vdash t_{12} : T_{11} \to T \qquad \Gamma|\Sigma \vdash v_2 : T_{11}$

代入補題より $\Gamma|\Sigma \vdash [x \mapsto v_2]t_{12} : T$ が成り立つので、

$\Sigma' = \Sigma$ とすればよい。 □

preservation

Proof.

E-App1 の場合

$t = t_1\ t_2 \qquad t' = t_1'\ t_2 \qquad t_1|\mu \to t_1'|\mu'$

仮定 $\Gamma|\Sigma \vdash t_1\ t_2 : T$ に反転補題を適用する。

$\Gamma|\Sigma \vdash t_1 : R \to T \qquad \Gamma|\Sigma \vdash t_2 : R$

帰納法の仮定より

$\Sigma \subseteq \Sigma' \qquad \Gamma|\Sigma' \vdash t_1' : R \to T \qquad \Gamma|\Sigma' \vdash \mu'$

T-App より $\Gamma|\Sigma' \vdash t_1'\ t_2 : T$ が成り立つ。 □

preservation

Proof.

E-RefV の場合

$t = \texttt{ref}\ v_1 \qquad t' = l \qquad \mu' = (\mu, l \mapsto v_1) \qquad l \notin \texttt{dom}(\mu)$

仮定 $\Gamma|\Sigma \vdash \texttt{ref}\ v_1 : T$ に反転補題を適用する。

$\Gamma|\Sigma \vdash v_1 : T_1 \qquad T = \texttt{Ref}\ T_1$

$\Sigma' = (\Sigma, l \mapsto T_1)$ とする。

$\Sigma \subseteq (\Sigma, l \mapsto T_1)$ は自明。

T-Loc より $\Gamma|(\Sigma, l \mapsto T_1) \vdash l : \texttt{Ref}\ T_1$

定義より $\Gamma|(\Sigma, l \mapsto T_1) \vdash (\mu, l \mapsto v)$ □

preservation

Proof.

E-DerefLoc の場合

$t = !l \qquad t' = v \qquad \mu' = \mu \qquad \mu(l) = v$

仮定 $\Gamma|\Sigma \vdash !l : T$ に反転補題をくりかえし適用する。

$\Gamma|\Sigma \vdash l : \texttt{Ref}\ T \qquad \Sigma(l) = T_1$

$\Sigma' = \Sigma$ とする。

仮定 $\Gamma|\Sigma \vdash \mu$ の定義より $\Gamma|\Sigma \vdash \mu(l) : \Sigma(l)$

よって $\Gamma|\Sigma \vdash v : T$ □

Proof.

E-Assign の場合

$t = (l := v_2) \qquad t' = \texttt{unit} \qquad \mu' = [l \mapsto v_2]\mu$

仮定 $\Gamma|\Sigma \vdash (l := v_2) : T$ に反転補題を適用する。

$\Gamma|\Sigma \vdash l : \texttt{Ref}\ T_{11} \qquad \Gamma|\Sigma \vdash v_2 : T_{11} \qquad T = \texttt{Unit}$

$\Sigma' = [l \mapsto T_{11}]\Sigma$ とする。

T-Unit より $\Gamma|[l \mapsto T_{11}]\Sigma \vdash \texttt{unit} : \texttt{Unit}$

定義より $\Gamma|[l \mapsto T_{11}] \vdash [l \mapsto v_2]\mu$

残りの場合は E-App1 と同様に示せる。 □

progress

定理 13.5.7 (progress)

$\emptyset|\Sigma \vdash t : T$ ならば、t は値、または全ての $\emptyset|\Sigma \vdash \mu$ が成り立つような μ に対して $t|\mu \to t'|\mu'$ となるような t' と μ' が存在する。

Proof.

$\emptyset|\Sigma \vdash t : T$ の導出に関する帰納法で証明する。

$\mathrm{T\text{-}Loc}$ の場合

ロケーションは値なので自明 □

progress

Proof.

T-Ref の場合

$t = \texttt{ref}\ t_1 \qquad T = \texttt{Ref}\ T_1 \qquad \emptyset|\Sigma \vdash t_1 : T_1$

帰納法の仮定より、t_1 は値、または

$t_1|\mu \to t_1'|\mu' \qquad \emptyset|\Sigma \vdash \mu$

t_1 が値 v_1 の場合、E-RefV より $\texttt{ref}\ v_1|\mu \to l|(\mu, l \mapsto v_1)$

t_2 が値でない場合、E-Ref より $\texttt{ref}\ t_1|\mu \to \texttt{ref}\ t_1'$ □

progress

Proof.

T-DEREF の場合

$t = !t_1 \qquad \emptyset|\Sigma \vdash t_1 : \texttt{Ref}\ T$

帰納法の仮定より、t_1 は値、または

$t_1|\mu \to t_1'|\mu' \qquad \emptyset|\Sigma \vdash \mu$

t_1 が値の場合、標準形の補題により t_1 はロケーション l_1

反転補題より $\Sigma(l_1) = T_1$

$\emptyset|\Sigma \vdash \mu$ の定義より $\mu(l_1) = v \qquad \emptyset|\Sigma \vdash v : T_1$

従って、E-DEREFLOC より $!l|\mu \to v|\mu$

t_1 が値でない場合、E-DEREF より $!t_1|\mu \to !t_1'|\mu'$ □

progress

Proof.

T-Assign の場合

$t = (t_1 := t_2) \quad \emptyset|\Sigma \vdash t_1 : \mathtt{Ref}\ T_{11} \quad \emptyset|\Sigma \vdash t_2 : T_{11}$

帰納法の仮定より、t_1 は値、または

$t_1|\mu \to t_1'|\mu' \qquad \emptyset|\Sigma \vdash \mu$

t_1 が値の場合、標準形の補題により t_1 はロケーション l_1

帰納法の仮定より、t_2 は値、または $t_2|\mu \to t_2'|\mu'$

t_2 が値 v_2 の場合、E-Assign より

$l_1 := v_2|\mu \to \mathtt{unit}|[l \mapsto v_2]\mu$

t_2 が値でない場合、E-Assign2 より

$l_1 := t_2|\mu \to l_1 := t_2'|\mu'$

t_1 が値でない場合、E-Deref より $!t_1|\mu \to !t_1'|\mu'$ □

Type safety が証明できました

NEW GAME! 1 巻より