

Unhygienic letregion and the region phase change

Ben Lippmeier

University of New South Wales

Fp-Syd 2011/08/18

Programming with References

```
letref x = "foo" in  
do putStr (readRef x)  
    writeRef x "bar"  
    putStr (readRef x)
```

```
readRef    :: Ref a -> a
```

```
writeRef   :: Ref a -> a -> ()
```

Mutable References and Locations

```

--> H ; letref x = val in t
--> H, l ~ val ; t[l/x] l fresh

H, l ~ val ; readRef l
--> H, l ~ val ; val

H, l ~ val ; writeRef l val2
--> H, l ~ val2 ; ()
```

Evaluation

```
H ; letref x = "foo" in  
  do putStr (readRef x)  
      writeRef x "bar"  
      putStr (readRef x)
```

Evaluation

```
H, l ~ "foo" ; do putStr (readRef l)
                writeRef l "bar"
                putStr (readRef l)
```

Evaluation

```
H, l ~ "foo" ; do putStr "foo"  
                writeRef l "bar"  
                putStr (readRef l)
```

Evaluation

```
H, l ~ "foo" ; do writeRef l "bar"  
                putStr (readRef l)
```

"foo"

Evaluation

```
H, l ~ "bar" ; do putStr (readRef l)
```

"foo"

Evaluation

H, l ~ "bar" ; do putStr "bar"

"foo"

Evaluation

H, l ~ "bar" ; ()

"foobar"

Location Phase Change

```
H ; letref x = "foo" in
  do putStr (readRef x)
     writeRef x "bar"
     putStr (readRef x)
```

```
--> H, l ~ "foo" ; do putStr (readRef l)
     writeRef l "bar"
     putStr (readRef l)
```

Type Environments and Store Typings

$TE \quad ; \quad ST \quad | - \quad v \quad :: \quad T1$

$TE, \quad x \quad : \quad Ref \quad T1 \quad ; \quad ST \quad | - \quad t \quad :: \quad T2$

$TE \quad ; \quad ST \quad | - \quad \mathbf{letref} \quad x = v \quad \mathbf{in} \quad t \quad :: \quad T2$

```
letref x = "foo" in  
do putStr (readRef x)  
    writeRef x "bar"  
    putStr (readRef x)
```

Type Environments and Store Typings

$$\begin{array}{l} \text{TE} \quad ; \text{ST} \quad | - \quad v :: T1 \\ \text{TE}, x : \text{Ref } T1 \quad ; \text{ST} \quad | - \quad t :: T2 \end{array}$$

$$\text{TE} \quad ; \text{ST} \quad | - \quad \mathbf{\text{letref } x = v \text{ in } t} :: T2$$
$$x : T \quad \text{`elem`} \quad \text{TE}$$

$$\text{TE} \quad ; \text{ST} \quad | - \quad x :: T$$

```
letref x = "foo" in
do putStr (readRef x)
    writeRef x "bar"
    putStr (readRef x)
```

Type Environments and Store Typings

$$\begin{array}{l} \text{TE} \quad ; \text{ST} \quad | - \quad v \quad :: \quad T1 \\ \text{TE}, x : \text{Ref } T1 \quad ; \text{ST} \quad | - \quad t \quad :: \quad T2 \end{array}$$

$$\text{TE} \quad ; \quad \text{ST} \quad | - \quad \mathbf{\text{letref}} \quad x = v \quad \mathbf{\text{in}} \quad t \quad :: \quad T2$$
$$l : T \quad \text{`elem`} \quad \text{ST}$$

$$\text{TE} \quad ; \quad \text{ST} \quad | - \quad l \quad :: \quad T$$

```
letref x = "foo" in
do putStr (readRef x)
    writeRef x "bar"
    putStr (readRef x)
```

Programming with Regions

```
letregion r in  
letref    x = "foo" at r in  
do putStr (readRef x)  
    writeRef "bar"  
    putStr (readRef x)
```

Programming with Regions

```
r :: %  
x :: Ref r String
```

```
letregion r in  
letref x = "foo" at r in  
do putStr (readRef x)  
    writeRef "bar"  
    putStr (readRef x)
```

```
readRef :: forall (r:%) (a:*)  
        . Ref r a -> a
```

```
writeRef :: forall (r:%) (a:*)  
         . Ref r a -> a -> ()
```


Region allocation

$$\begin{array}{l} \mathbf{H} \quad ; \text{ letregion } \mathbf{r} \text{ in } t \\ \text{-->} \mathbf{H}, \mathbf{p} \quad ; t[\mathbf{p}/\mathbf{r}] \quad \mathbf{p} \text{ fresh} \end{array}$$

$$\begin{array}{l} \mathbf{H} \quad ; \text{ letref } x = \text{val at } \mathbf{p} \text{ in } t \\ \text{-->} \mathbf{H}, \mathbf{p}, \mathbf{l}(\mathbf{p}) \sim \text{val} \quad ; t[\mathbf{l}/x] \quad \mathbf{l} \text{ fresh} \end{array}$$

Evaluation

```
H ; letregion r in
    letref x = "foo" at r in
    do putStr (readRef x)
       writeRef x "bar"
       putStr (readRef x)
```

Evaluation

```
H, p ; letref x = "foo" at p in
      do putStr (readRef x)
         writeRef x "bar"
         putStr (readRef x)
```

Evaluation

```
H, p, l(p) ~ "foo" ; do putStr (readRef l)
                        writeRef l "bar"
                        putStr (readRef l)
```

Evaluation

```
H, p, l(p) ~ "foo" ; do putStr "foo"  
                      writeRef l "bar"  
                      putStr (readRef l)
```

New typing rules for letregion and letref

$$\text{TE}, r : \% ; \text{ST} \mid - t :: T$$

$$\text{TE} ; \text{ST} \mid - \text{letregion } r \text{ in } t :: T$$
$$r : \% \text{ `elem` TE}$$
$$\text{TE} ; \text{ST} \mid - v :: T1$$
$$\text{TE}, x : \text{Ref } r \text{ T1} ; \text{ST} \mid - t :: T2$$

$$\text{TE} ; \text{ST} \mid - \text{letref } x = v \text{ at } r \text{ in } t :: T2$$

Type change during evaluation


```
r :: %  
x :: Ref r String
```

```
letregion r in  
letref      x = "foo" at r in  
do putStr (readRef x)  
    writeRef "bar"  
    putStr (readRef x)
```

Type change during evaluation

```
r :: %  
x :: Ref r String
```

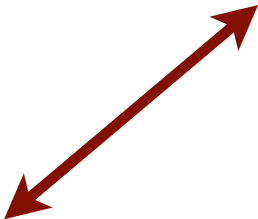
```
letregion r in  
letref      x = "foo" at r in  
do putStr (readRef x)  
    writeRef "bar"  
    putStr (readRef x)
```



Type change during evaluation

`r :: %`
`x :: Ref p String`

--> `letref x = "foo" at p in`
`do putStr (readRef x)`
`writeRef "bar"`
`putStr (readRef x)`



Region encapsulation

```
letregion r in
letref x = "foo" at r
in x :: Ref r String
```

```
letregion r in
letref x = "foo" at r
in putStr (readRef x) :: ()
```

Imposing hygiene?

$r \notin \text{freeVars}(T)$
 $TE, r : \text{rgn} ; ST \vdash t :: T$

$TE ; ST \vdash \text{letregion } r \text{ in } t :: T$

Imposing hygiene?

```
letregion r in  
letref x = "foo" at r  
in \() -> readRef x      :: () -> String
```