CS520, Handout 10, 3 October 2006

References and Side Effects

Reading: [Pierce, Chapter 13]

Allocation, Dereferencing and Assignment

[Pierce,pp 154-155]

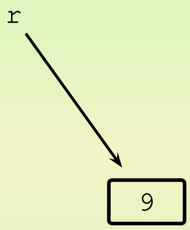
Example

```
  r = ref 5;              (* allocation of a reference *)
> r : Ref Nat
  !r;                     (* dereferencing a reference *)
> 5 : Nat
  r := 7;                 (* assignment of a new value *)
> unit : Unit;
  !r;
> 7 : Nat
  (r := succ(!r) ; !r);
> 8 : Nat
  (lambda _ : Unit. !r) (r := succ(!r));
> 9 : Nat
```

Aliasing and Shared State

[Pierce,pp 155-157]

```
  !r ;
> 9 :  Nat
```

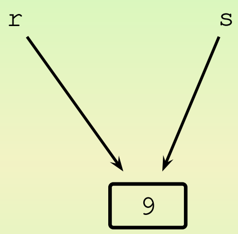


Aliasing and Shared State

[Pierce,pp 155-157]

```
  !r ;
> 9 :  Nat

  s = r ;        (* s  alias of r *)
> s :  Ref Nat
```



Aliasing and Shared State

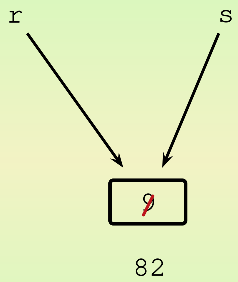
[Pierce,pp 155-157]

```
  !r ;
> 9 :  Nat

  s = r ;        (* s  alias of r *)
> s :  Ref Nat

  s := 82 ;
> unit :  Unit

  !r ;
> 82 :  Nat
```



Aliasing and Shared State

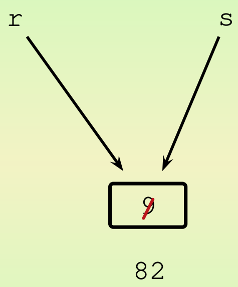
[Pierce,pp 155-157]

```
  !r ;
> 9 :  Nat

  s = r ;        (* s  alias of r *)
> s :  Ref Nat

  s := 82 ;
> unit :  Unit

  !r ;
> 82 :  Nat
```



Quick Exercise: Consider expressions `(a := 1; a := !b)` and `(a := !b)`. State a condition when both have the same effect, and another condition when both do not have the same effect.

Syntax of Terms [Pierce, Figure 13-1, organized differently]

t ::=	...	*terms*	
	`unit`		*constant* `unit`
	`ref` t		*reference creation*
	$!\,t$		*dereference*
	$t := t$		*assignment*
	ℓ		*store location*
v ::=	...	*values*	
	`unit`		*constant* `unit`
	ℓ		*store location*

Evaluation Rules (relative to ***stores*** μ)

Stores

μ ::=		*stores*	
	$\varnothing$		*empty store*
	$\mu, \ell = v$		*location binding*

Evaluation Rules (relative to ***stores*** μ)

Stores

μ ::=		*stores*
	$\varnothing$	*empty store*
	$\mu, \ell = v$	*location binding*

Rules (not all of them, see [Pierce, page 166, right column])

$$(\lambda x : T.\, t)\, v \mid \mu \Rightarrow [x \mapsto v]t \mid \mu \qquad \text{(E-AppAbs)}$$

$$\ell := v \mid \mu \Rightarrow \mathtt{unit} \mid [\ell \mapsto v]\mu \qquad \text{(E-Assign)}$$

Evaluation Rules (relative to ***stores*** μ)

Stores

μ ::=		*stores*
	$\varnothing$	*empty store*
	$\mu, \ell = v$	*location binding*

Rules (not all of them, see [Pierce, page 166, right column])

$$(\lambda x : T.\, t)\, v \mid \mu \Rightarrow [x \mapsto v]t \mid \mu \qquad \text{(E-AppAbs)}$$

$$\ell := v \mid \mu \Rightarrow \mathtt{unit} \mid [\ell \mapsto v]\mu \qquad \text{(E-Assign)}$$

$$\frac{\ell \notin dom(\mu)}{\mathtt{ref}\ v \mid \mu \Rightarrow \ell \mid (\mu, \ell = v)} \qquad \text{(E-RefV)}$$

$$\frac{\mu(\ell) = v}{!\,\ell \mid \mu \Rightarrow v \mid \mu} \qquad \text{(E-DerefLoc)}$$

References and Side Effects – p.5

Typing Rules (relative to ***contexts*** Γ and ***store typings*** Σ)

Store Typings

Σ ::=		*store typings*
	$\varnothing$	*empty store typing*
	$\Sigma, \ell : T$	*location typing*

Typing Rules (relative to ***contexts*** Γ and ***store typings*** Σ)

Store Typings

Σ ::=		*store typings*
	$\varnothing$	*empty store typing*
	$\Sigma, \ell : T$	*location typing*

Rules (not all of them, see [Pierce, page 167, left and right columns])

$$\frac{\Sigma(\ell) = T}{\Gamma \,|\, \Sigma \vdash \ell : \texttt{Ref}\ T} \qquad \text{(T-Loc)}$$

$$\frac{\Gamma \,|\, \Sigma \vdash t : T}{\Gamma \,|\, \Sigma \vdash \texttt{ref}\ t : \texttt{Ref}\ T} \qquad \text{(T-Ref)}$$

$$\frac{\Gamma \,|\, \Sigma \vdash t_1 : \texttt{Ref}\ T \qquad \Gamma \,|\, \Sigma \vdash t_2 : T}{\Gamma \,|\, \Sigma \vdash t_1 := t_2 : \texttt{Unit}} \qquad \text{(T-Assign)}$$

Progress + Preservation [Pierce, Section 13.5]

Preliminary Definition [Pierce, Definition 13.5.1]
A store μ is well-typed w.r.t. to a context Γ and store typing Σ, written $\Gamma \,|\, \Sigma \vdash \mu$, if $dom(\mu) = dom(\Sigma)$ and $\Gamma \,|\, \Sigma \vdash \mu(\ell) : \Sigma(\ell)$ for every $\ell \in dom(\mu)$.

Progress + Preservation [Pierce, Section 13.5]

Preliminary Definition [Pierce, Definition 13.5.1]
A store μ is well-typed w.r.t. to a context Γ and store typing Σ, written $\Gamma \,|\, \Sigma \vdash \mu$, if $dom(\mu) = dom(\Sigma)$ and $\Gamma \,|\, \Sigma \vdash \mu(\ell) : \Sigma(\ell)$ for every $\ell \in dom(\mu)$.

Progress [Pierce, Theorem 13.5.7]
If t is a closed well-typed term, i.e., $\varnothing \,|\, \Sigma \vdash t : T$ for some type T, then either t is a value or else, for any store μ such that $\varnothing \,|\, \Sigma \vdash \mu$, there is a term t' and store μ' such that $t \,|\, \mu \Rightarrow t' \,|\, \mu'$.

Progress + Preservation

[Pierce, Section 13.5]

Preliminary Definition [Pierce, Definition 13.5.1]

A store μ is well-typed w.r.t. to a context Γ and store typing Σ, written $\Gamma \,|\, \Sigma \vdash \mu$, if $dom(\mu) = dom(\Sigma)$ and $\Gamma \,|\, \Sigma \vdash \mu(\ell) : \Sigma(\ell)$ for every $\ell \in dom(\mu)$.

Progress [Pierce, Theorem 13.5.7]

If t is a closed well-typed term, i.e., $\varnothing \,|\, \Sigma \vdash t : T$ for some type T, then either t is a value or else, for any store μ such that $\varnothing \,|\, \Sigma \vdash \mu$, there is a term t' and store μ' such that $t \,|\, \mu \Rightarrow t' \,|\, \mu'$.

Preservation [Pierce, Theorem 13.5.3]

If $\Gamma \,|\, \Sigma \vdash t : T$ and $\Gamma \,|\, \Sigma \vdash \mu$ and $t \,|\, \mu \Rightarrow t' \,|\, \mu'$, then for some $\Sigma' \supseteq \Sigma$ it holds that $\Gamma \,|\, \Sigma' \vdash t' : T$ and $\Gamma \,|\, \Sigma' \vdash \mu'$.

Quick Exercise: Why can we not take $\Sigma = \Sigma'$ in the Preservation Theorem?