The Environmental Model of Execution

Amitabha Sanyal

Department of Computer Science and Engineering
IIT Bombay
Mumbai - 400076

as@cse.iitb.ac.in

March 2018

Environment model of execution

```
(define x 2)
(define y 3)
```

At this point only x and y are visible.

```
(define (f z x)
    (set! z 6)
    (set! x 1)
    ...
```

At this point y, z and x are visible.

```
)

(define (g p q)
    (set! p 12)
    (set! q 1)
    ...
```

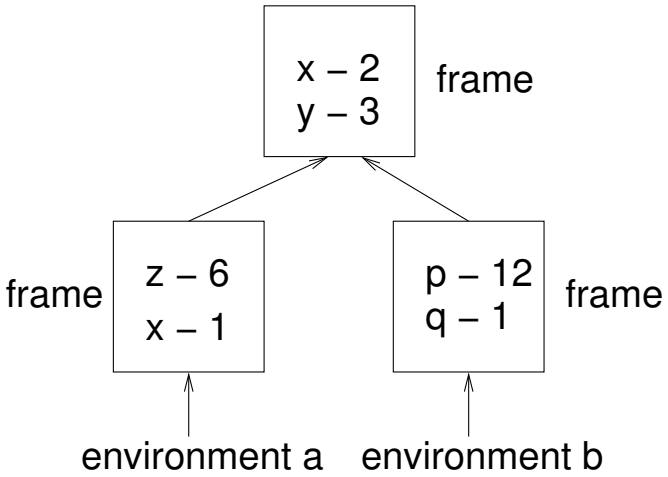
At this point p, q, x and y are visible.

```
)
```

Environment model of execution

- Question: How does a running program access the variables in scope (visible variables) during execution?
- Answer: Through an arrangement called an environment.

What does the environment look like?



- A function call or a let creates a new frame.
- A `define` extends a frame.
- An environment is a chain of frames.

Environment model of execution

- In a side-effect free language, we can think of a value being bound to a variable directly. We denote it as x — 5.
- For language with side-effects:
  - A variable is bound to the address of a location. This location contains the value. In pictures: x — 1024 — 5. Here 1024 is the address of the location.
  - If the exact address is not important, this is denoted as x — $\boxed{5}$.
- In the same scope, the binding of variable to its location does not change.
  - x always remains bound to address 1024.

  However, the contents of this location can change
  - The contents of 1024 can change to 6.
- We shall continue to describe x — $\boxed{5}$ as "x is bound to 5", However we know now the underlying picture is more elaborate.

Building the environment

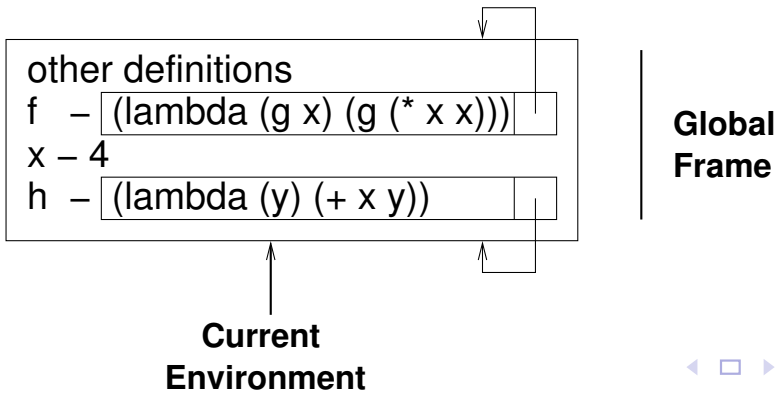
Rules for environment creation:

Rule 1: A `define` extends the current frame

Rule 2: The value of a function is a closure. The environment E of the closure is the same environment in which the function was evaluated.

```
(define (f g x) (g (* x x)))
(define x 4)
(define (h y) (+ x y))
(define w (f h 5))
```

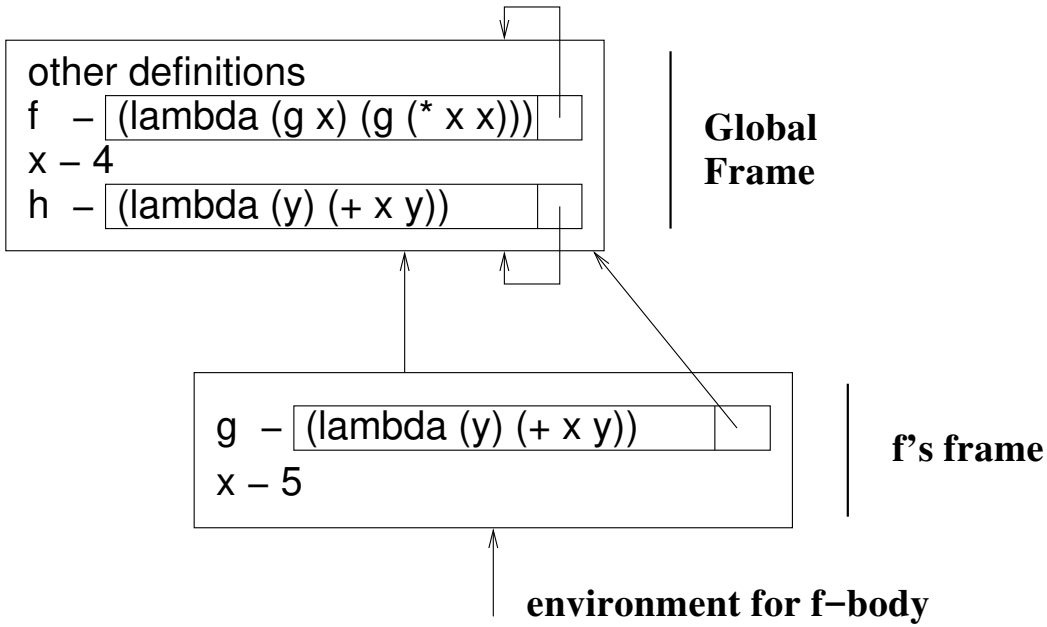
Start with a global frame. The first three `defines` extend the global frame.



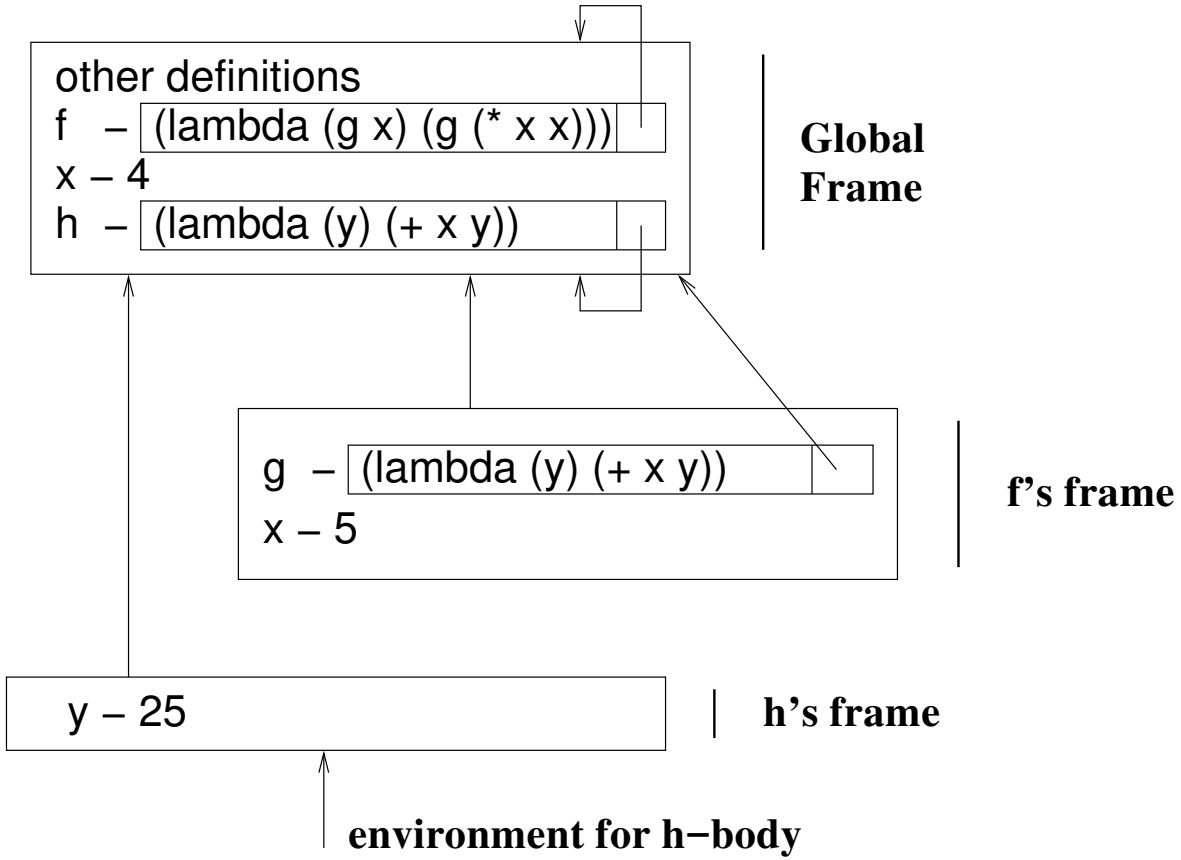
Building the environment

Rule 3: During a function call, a new frame containing the parameters is created.

Rule 4: The global pointer of the function frame is made to point to the environment carried by the function.



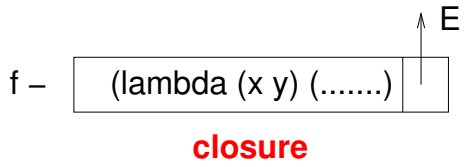
First example



Environment model of execution

Key Ideas:

- A function is created by evaluating a `define` or a `lambda`. This results in a pair consisting of:
  - The text of the lambda.
  - A pointer to the environment in which the function was created.
  - The pair (lambda, E) taken together is called a closure.



Environment model of execution

Key Ideas:

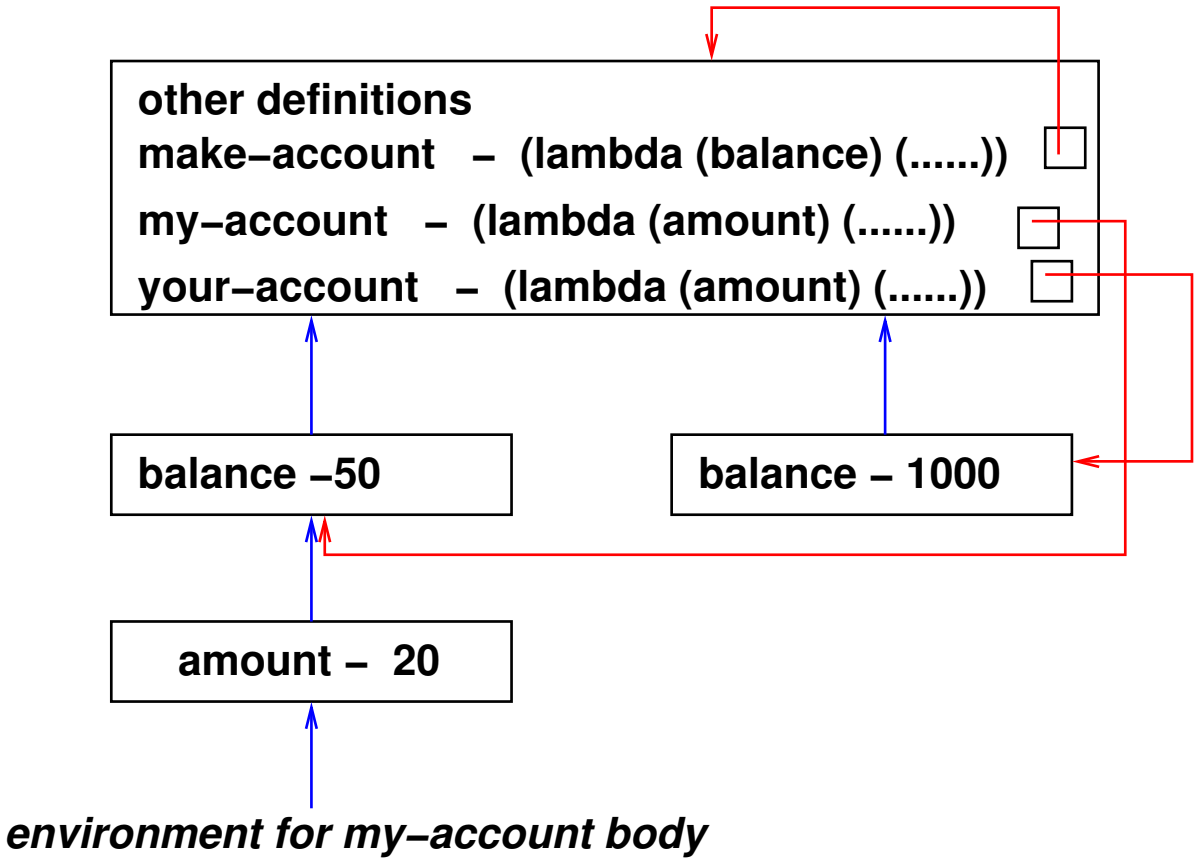
- A function is applied to a set of arguments by:
  - Constructing a frame binding the formal parameters of the function to the arguments of the call.
  - Evaluating the body of the function in:
    - An environment E' that starts with the new frame created, and
    - has as its enclosing (global) environment the environment E of the function being applied.

A second example

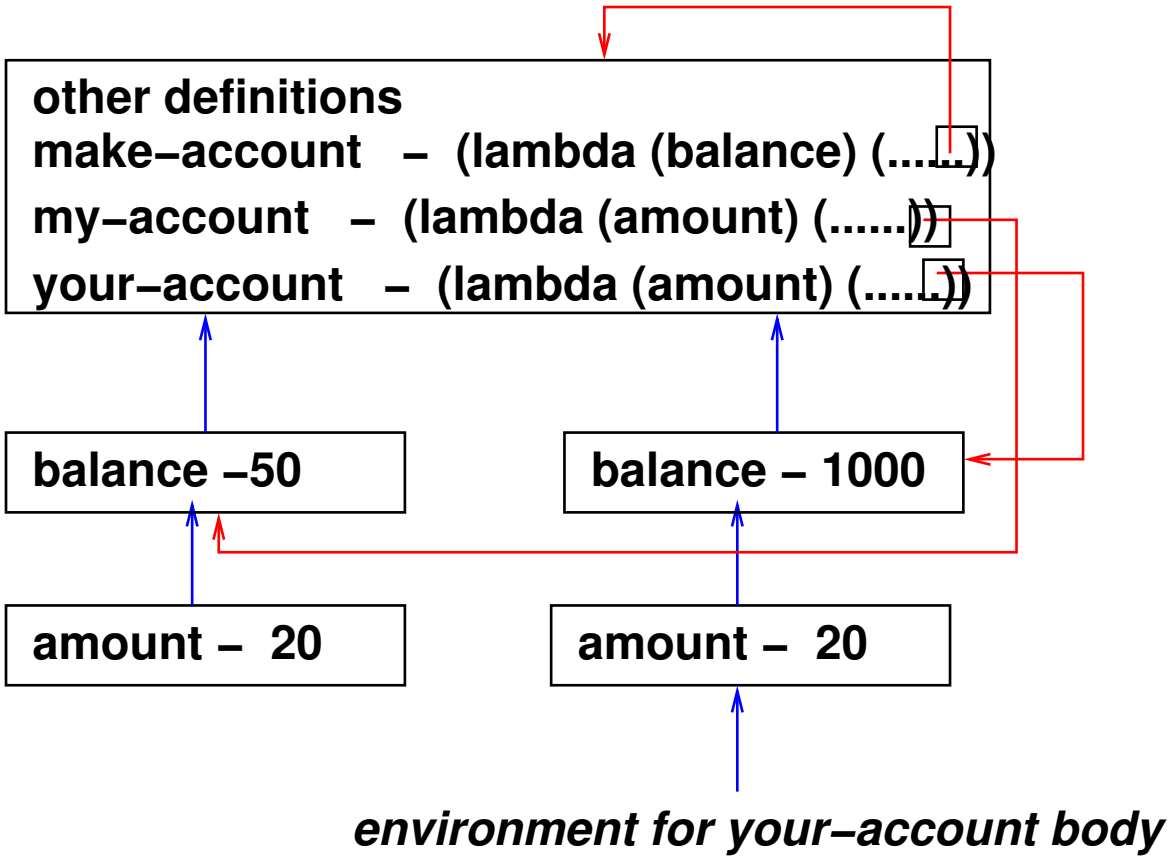
```
(define (make-account balance)
   (lambda  (amount)
      (if (>= balance amount)
          (begin
             (set! balance (- balance amount))
             balance)
          "Insufficient funds")))

(define my-account (make-account 50))
(define your-account (make-account 1000))
> (my-account 20)
 30
> (your-account 20)
 980
> (my-account 50)
 "Insufficient funds"
```

Second example

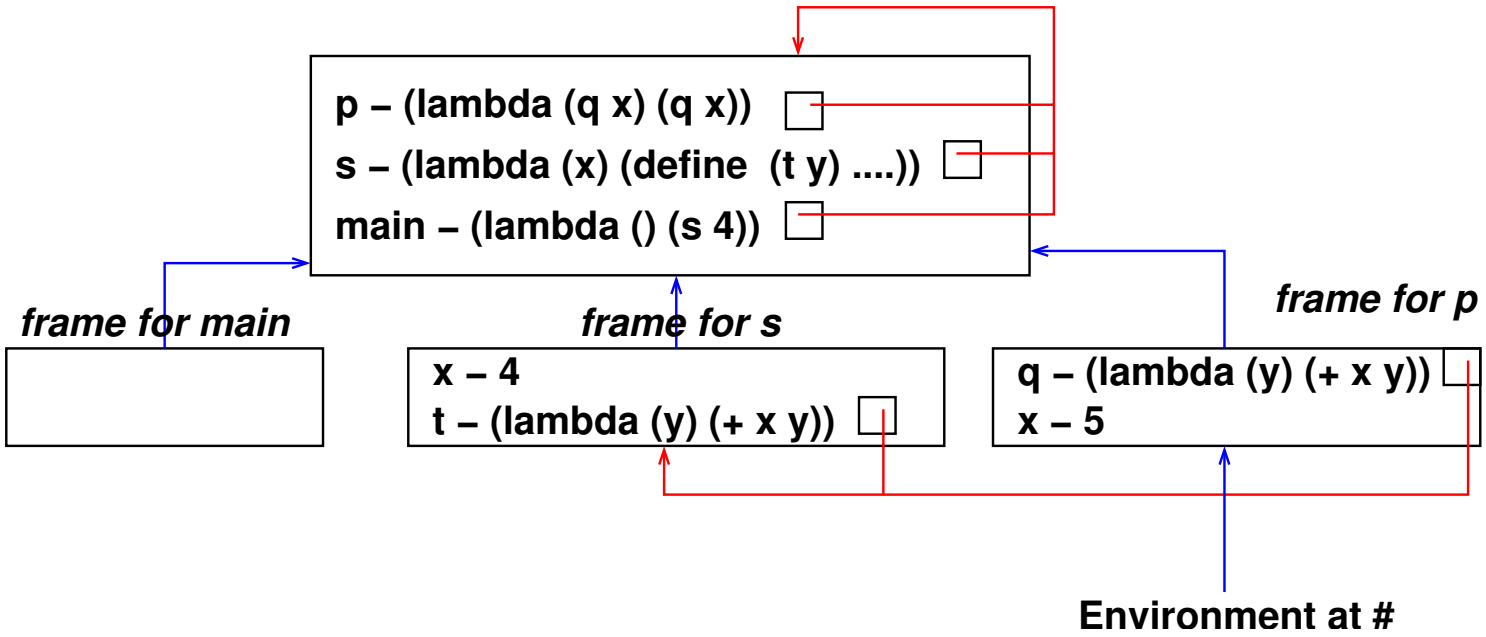


Second example



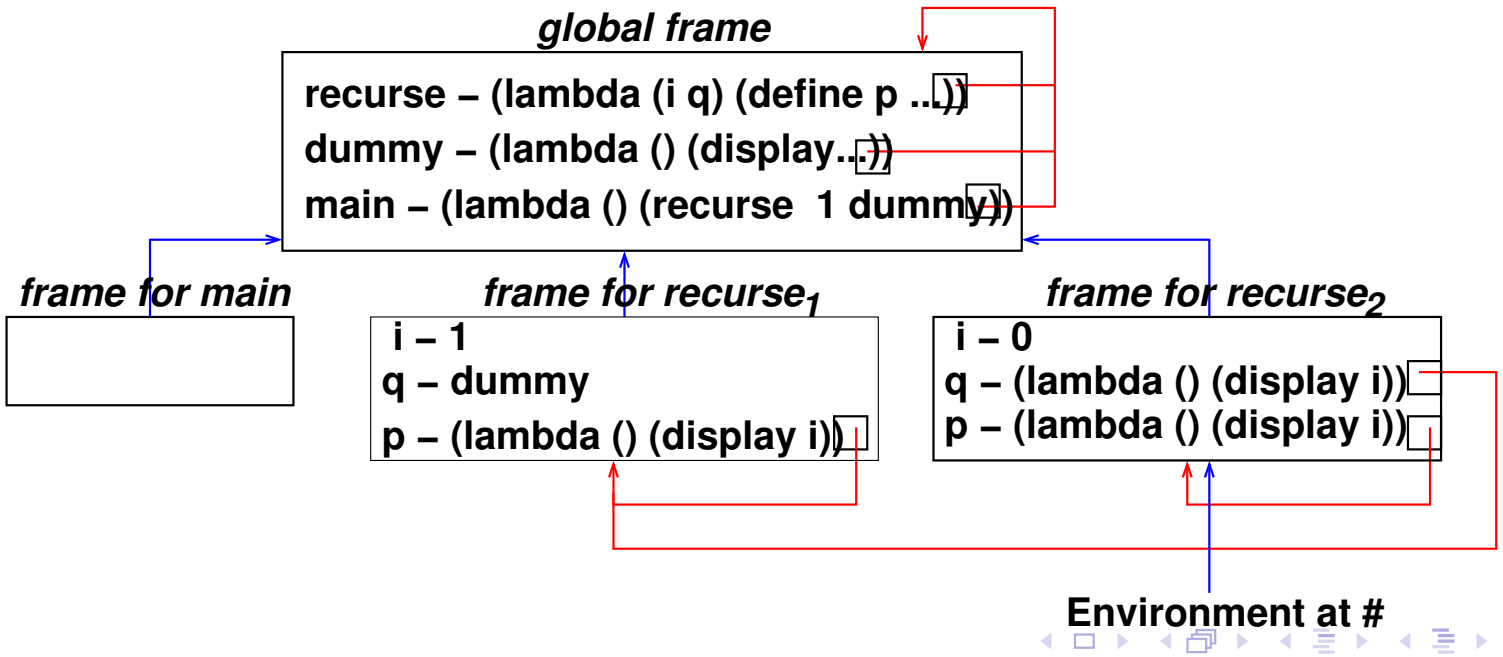
Third example

```
(define (p q x) #(q x))
(define (s x)
  (define (t y) (+ x y))
  (p t 5))
(define (main) (s 4))
(main)
```



Fourth example

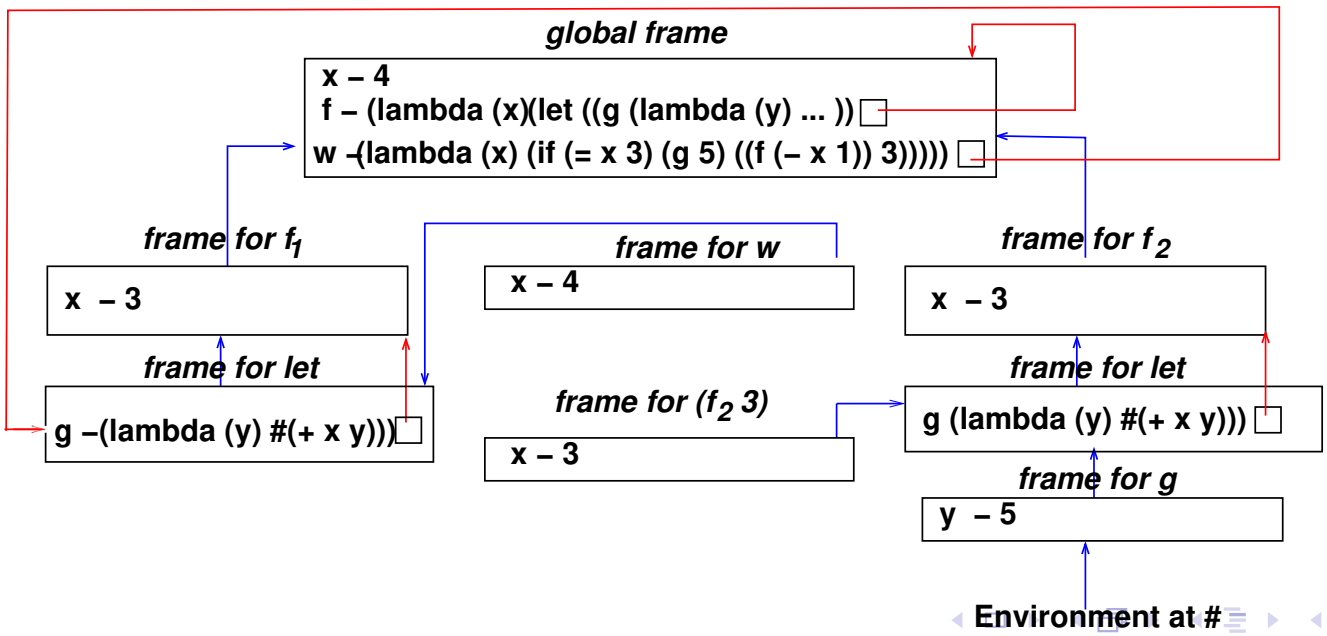
```
(define (recurse i q)
  (define (p) (display i))
  (if (> i 0) (recurse (- i 1) p)
      (begin # (p) (q))))
(define (dummy) (display ""))
(define (main) (recurse 1 dummy))
(main)
```



Fifth example

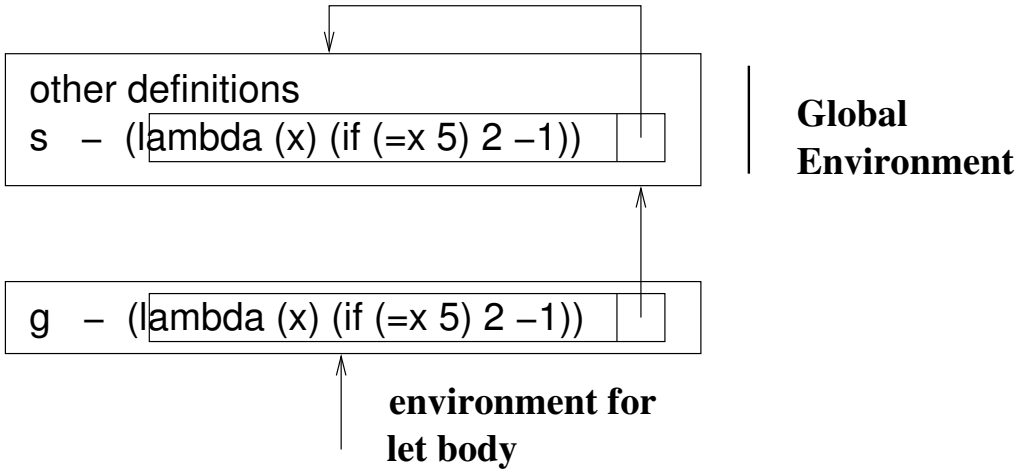
Rule 5: A `let` creates its own frame.

```
(define x 4)
(define (f x)
  (let ((g (lambda (y) #(+ x y))))
    (lambda (x) (if (= x 3) (g 5) ((f (- x 1)) 3)))))
(define w (f 3))
(define result (w 4))
```



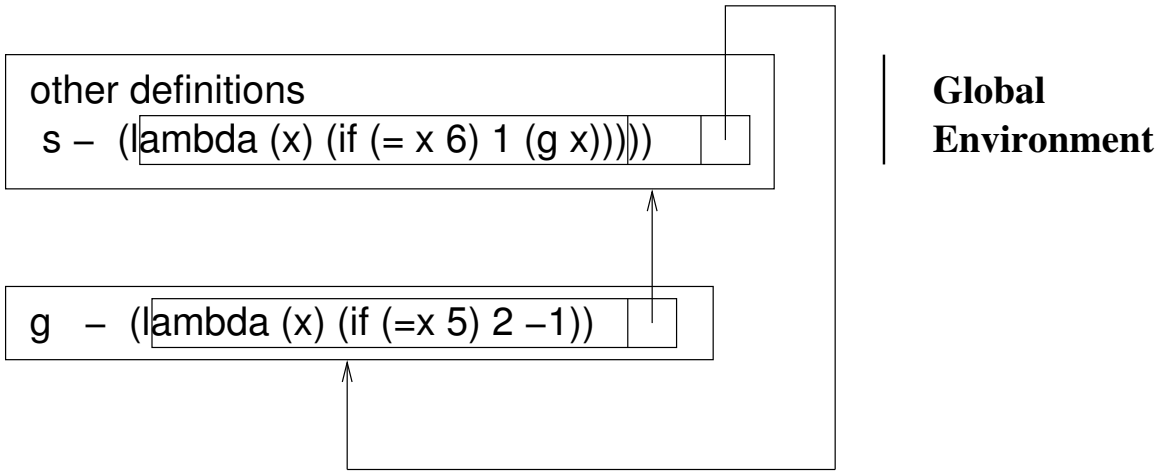
Sixth example

```
(define (s x) (if (= x 5) 2 -1))
(set! s (let ((g s)) (lambda (x) (if (= x 6) 1 (g x)))))
```



Sixth example

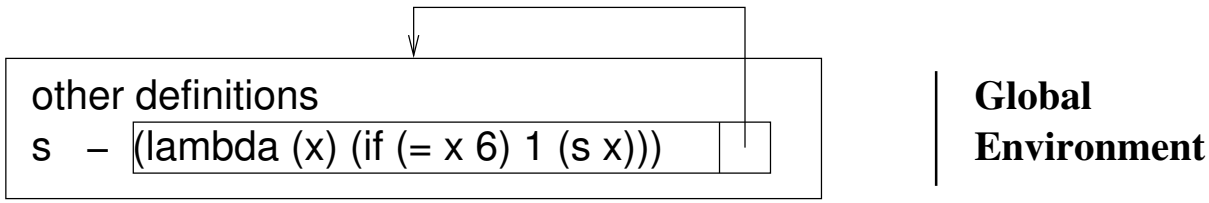
After the `set!`



Seventh example

```
(define (s x) (if (= x 5) 2 -1))
(set! s (lambda (x) (if (= x 6) 1 (s x))))
```

After the `set!`



Conclusion

The environmental model presented here is a simplification of the actual execution in the following ways:

1. The stack is not shown. In fact as the current environment keeps changing, the earlier environment is pushed into the stack.
2. The model presented seems to suggest that variables are accessed at run-time by starting from the current environment and searching in each frame. This would be unacceptably slow in practice.
3. The exact details of variable access will be described in CS 302 (Implementation of Programming Languages).