

Cost of Substitution

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}} )
```

⇒

```
(interp {with {y 2}
          {+ 100 {+ 99 {+ 98 ... {+ y 1}}}} } )
```

⇒

```
(interp {+ 100 {+ 99 {+ 98 ... {+ 2 1}}}} )
```

With n variables, evaluation will take $O(n^2)$ time!

Delaying Substitution

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}} )
```

⇒

```
(interp {with {y 2}
          {+ 100 {+ 99 {+ 98 ... {+ y x}}}} )
```

⇒

```
(interp {+ 100 {+ 99 {+ 98 ... {+ y x}}}} )
```

⇒ ... ⇒

```
(interp y)
```

Delaying Substitution with the Same Identifier

`(interp {with {x 1}
 {with {x 2}
 x}})`

⇒

`(interp {with {x 2}
 x})`

⇒

`(interp x)`

Always add to start, then always check from start

Representing Delayed Substitution

Change

```
; interp : WAE -> num
```

to

```
; interp : WAE SubCache -> num
```

```
(define-type SubCache  
  [mtSub]  
  [aSub (name symbol?)  
        (value number?)  
        (rest SubCache?)])
```

Interp with SubCache

```
(interp {with {x 1}
         {with {y 2}
          {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}})
(mtSub))
```

```
⇒ (interp {with {y 2}
          {+ 100 {+ 99 {+ 98 ... {+ y x}}}}})
(aSub 'x 1 (mtSub)))
```

```
⇒ (interp {+ 100 {+ 99 {+ 98 ... {+ y x}}}})
(aSub 'y 2 (aSub 'x 1 (mtSub))))
```

⇒ ...

```
⇒ (interp y (aSub 'y 2 (aSub 'x 1 (mtSub))))
```

WAE Interpreter with Delayed Substitutions

```
; interp : WAE SubCache -> num
(define (interp a-wae sc)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l sc) (interp r sc))]
    [sub (l r) (- (interp l sc) (interp r sc))]
    [with (bound-id named-expr body-expr)
      ...]
    [id (name) ...]))
```

WAE Interpreter with Delayed Substitutions

```
; interp : WAE SubCache -> num
(define (interp a-wae sc)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l sc) (interp r sc))]
    [sub (l r) (- (interp l sc) (interp r sc))]
    [with (bound-id named-expr body-expr)
      ...]
    [id (name) (lookup name sc)]))
```

WAE Interpreter with Delayed Substitutions

```
; lookup : symbol SubCache -> num
(define (lookup name sc)
  (type-case SubCache sc
    [mtSub () (error 'lookup "free variable")]
    [aSub (sub-name num rest-sc)
     (if (symbol=? sub-name name)
         num
         (lookup name rest-sc))]))
```


WAE Interpreter with Delayed Substitutions

```
; interp : WAE SubCache -> num
(define (interp a-wae sc)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l sc) (interp r sc))]
    [sub (l r) (- (interp l sc) (interp r sc))]
    [with (bound-id named-expr body-expr)
      ...]
    [id (name) (lookup name sc)]))
```

WAE Interpreter with Delayed Substitutions

```
; interp : WAE SubCache -> num
(define (interp a-wae sc)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l sc) (interp r sc))]
    [sub (l r) (- (interp l sc) (interp r sc))]
    [with (bound-id named-expr body-expr)
      ... (interp named-expr sc) ...]
    [id (name) (lookup name sc)]))
```

WAE Interpreter with Delayed Substitutions

```
; interp : WAE SubCache -> num
(define (interp a-wae sc)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l sc) (interp r sc))]
    [sub (l r) (- (interp l sc) (interp r sc))]
    [with (bound-id named-expr body-expr)
      ...
      (aSub bound-id (interp named-expr sc) sc)
      ...
    ]
    [id (name) (lookup name sc)]))
```


WAE Interpreter with Delayed Substitutions

```
; interp : WAE SubCache -> num
(define (interp a-wae sc)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l sc) (interp r sc))]
    [sub (l r) (- (interp l sc) (interp r sc))]
    [with (bound-id named-expr body-expr)
      (interp
        body-expr
        (aSub bound-id (interp named-expr sc) sc))]
    [id (name) (lookup name sc)]))
```

Function Calls

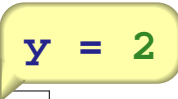
```
{defun {f x} {+ 1 x}}
```

```
(interp {with {y 2}  
        {f 10}})
```



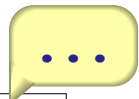
⇒

```
(interp {f 10})
```



⇒

```
(interp {+ 1 x})
```



Function Calls

```
{defun {f x} {+ 1 x}}
```

```
(interp {with {y 2}  
        {f 10}} )
```

⇒

```
(interp {f 10} )
```

⇒

```
(interp {+ 1 x} )
```

Interpreting function body starts with only one substitution

F1WAE Interpreter with Delayed Substitutions

```
; interp : F1WAE list-of-FunDef SubCache -> num
(define (interp a-flwae fundefs sc)
  (type-case F1WAE a-flwae
    ...
    [app (name arg-expr)
         ...]))
```

F1WAE Interpreter with Delayed Substitutions

```
; interp : F1WAE list-of-FunDef SubCache -> num
(define (interp a-flwae fundefs sc)
  (type-case F1WAE a-flwae
    ...
    [app (name arg-expr)
         (local [(define a-fundef
                    (lookup-fundef name fundefs))]
                 (interp (fundef-body a-fundef)
                         fundefs
                         ...
                         (interp arg-expr fundefs sc)
                         ...))
                ]))
```


F1WAE Interpreter with Delayed Substitutions

```
; interp : F1WAE list-of-FunDef SubCache -> num
(define (interp a-flwae fundefs sc)
  (type-case F1WAE a-flwae
    ...
    [app (name arg-expr)
         (local [(define a-fundef
                   (lookup-fundef name fundefs))]
                 (interp (fundef-body a-fundef)
                         fundefs
                         (aSub (fundef-arg-name a-fundef)
                              (interp arg-expr fundefs sc)
                              (mtSub)))))) ]))
```

HW 3: The PLAI Void language

Install `handin+plai-v4.plt` to get the **PLAI Void** language:

- Defined functions take zero arguments and return void
- No `lambda` or `local`

HW 3 is to translate the WAE+SubCache `interp` into this language

From PLAI Advanced to PLAI Void

```
; f : num -> num
(define (f x)
  (+ x 1))
```

```
(test (f 10) 11)
```

```
(define x 0)
(define fresult 0)
; f : -> void
(define (f)
  (set! fresult (+ x 1)))
```

```
(test (begin (set! x 10) (f) fresult)
      11)
```

Tree in PLAI Advanced

```
(define-type Tree
  [leaf (n number?)]
  [fork (l Tree?)
        (r Tree?)])

; sum : Tree -> num
(define (sum t)
  (type-case Tree t
    [leaf (n) n]
    [fork (l r) (+ (sum l) (sum r))]))

(test (sum (fork (leaf 10)
                 (fork (leaf 5)
                       (leaf 3))))
      18)
```

Tree in PLAI Void

```
(define-type Tree
  [leaf (n number?)]
  [fork (l Tree?)
        (r Tree?)])

(define t (leaf 0))
(define result 0)
; sum : -> void
(define (sum)
  (type-case Tree t
    [leaf (n) (set! result n)]
    [fork (l r)
          (begin
             (set! t l) (sum)
             (set! t r) (sum)
             ...)])))

(test (begin (set! t (fork (leaf 10) (fork (leaf 5) (leaf 3))))
      (sum) result)
      18)
```

Tree in PLAI Void

```
(define-type Tree
  [leaf (n number?)]
  [fork (l Tree?)
        (r Tree?)])

(define t (leaf 0))
(define result 0)
; sum : -> void
(define (sum)
  (type-case Tree t
    [leaf (n) (set! result n)]
    [fork (l r)
          (begin
            (set! t l) (sum)
            (set! t r) (sum)
            ... (+ result result) ; No...
            ...)]))

(test (begin (set! t (fork (leaf 10) (fork (leaf 5) (leaf 3))))
      (sum) result)
      18)
```

Tree in PLAI Void

```
(define-type Tree
  [leaf (n number?)]
  [fork (l Tree?)
        (r Tree?)])

(define t (leaf 0))
(define result 0)
; sum : -> void
(define (sum)
  (type-case Tree t
    [leaf (n) (set! result n)]
    [fork (l r)
          (begin
            (set! t l) (sum)
            ... (+ result
                (begin (set! t r) (sum) result))
            ...))
          ]))

(test (begin (set! t (fork (leaf 10) (fork (leaf 5) (leaf 3))))
      (sum) result)
      18)
```

Tree in PLAI Void

```
(define-type Tree
  [leaf (n number?)]
  [fork (l Tree?)
        (r Tree?)])

(define t (leaf 0))
(define result 0)
; sum : -> void
(define (sum)
  (type-case Tree t
    [leaf (n) (set! result n)]
    [fork (l r)
      (begin
        (set! t l) (sum)
        (set! result
          (+ result
            (begin (set! t r) (sum) result)))))]))

(test (begin (set! t (fork (leaf 10) (fork (leaf 5) (leaf 3))))
      (sum) result)
      18)
```