

Cost of Substitution

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

Cost of Substitution

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

⇒

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

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}}}} )
```

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!

Deferring Substitution



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

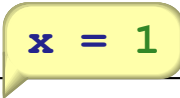
Deferring Substitution

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



⇒

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



Deferring 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}}}} )
```

Deferring 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 )
```



Deferring Substitution with the Same Identifier

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



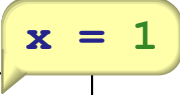
Deferring Substitution with the Same Identifier

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



⇒

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



Deferring Substitution with the Same Identifier

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

⇒

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

⇒

`(interp x)`

Deferring 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

Deferring Substitution with the Same Identifier

(interp

{with {x 1}

{+ {with {x 2}

x}

x}})

)

Deferring Substitution with the Same Identifier

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

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

x = 1

Deferring Substitution with the Same Identifier

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

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

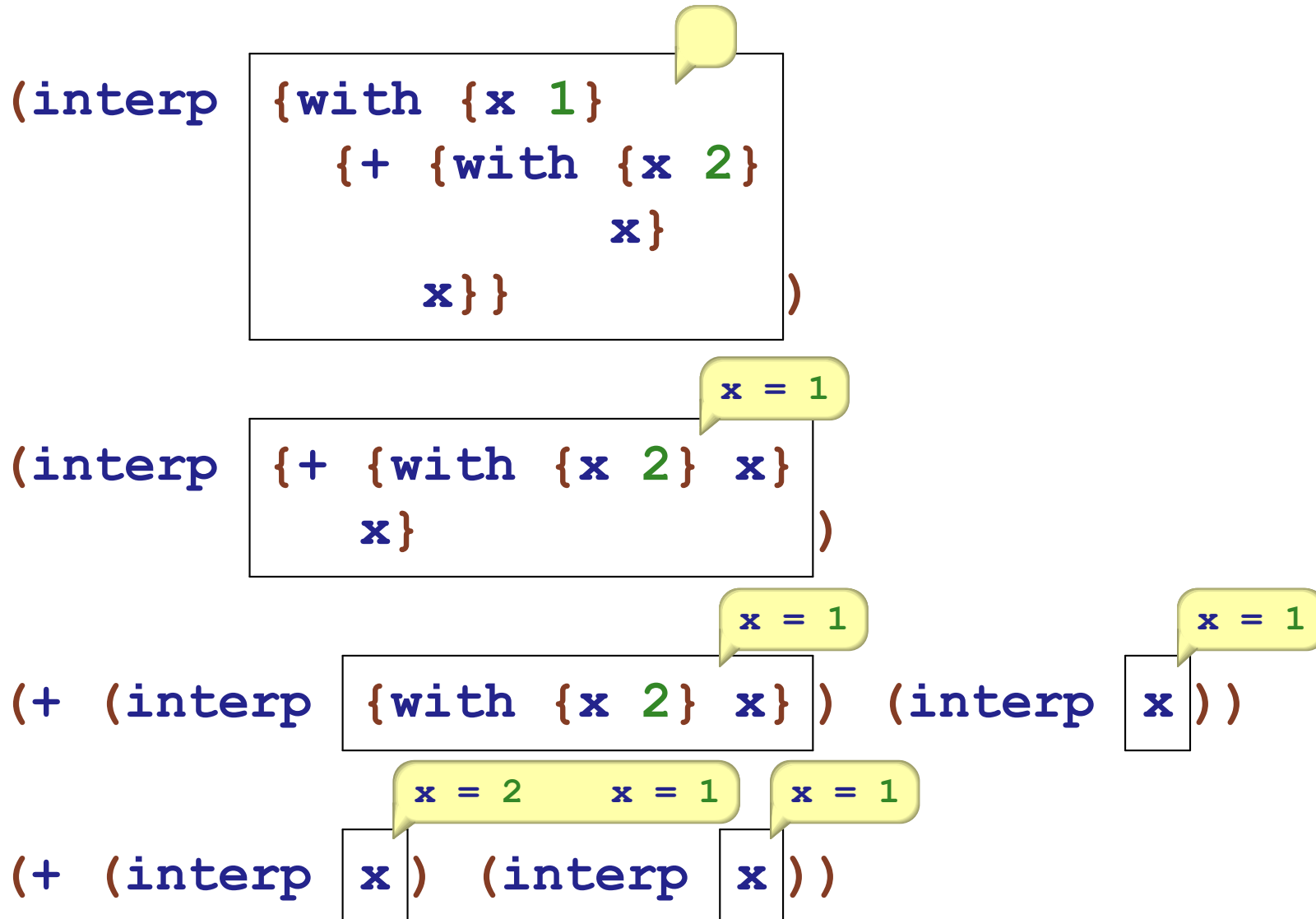
x = 1

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

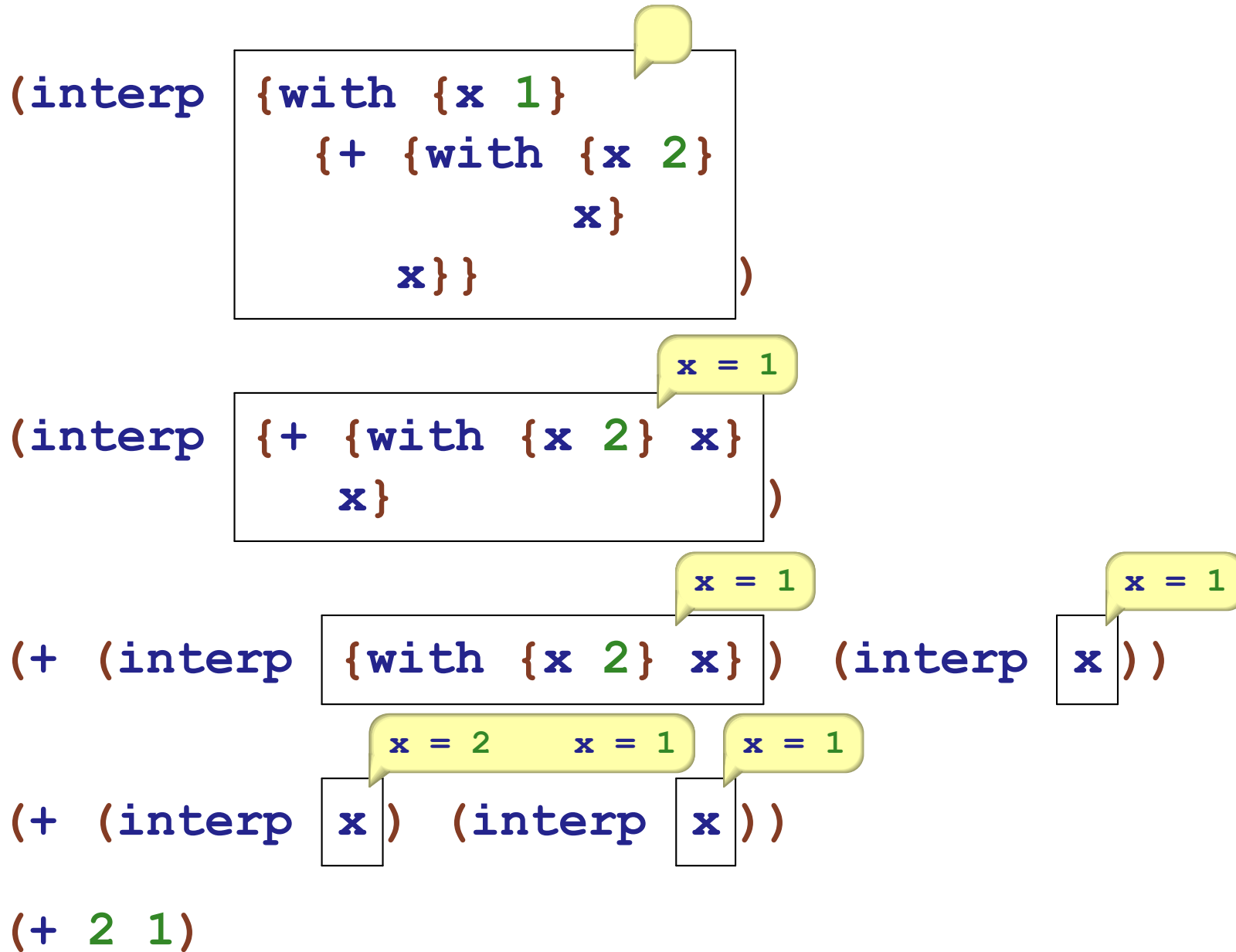
x = 1

x = 1

Deferring Substitution with the Same Identifier



Deferring Substitution with the Same Identifier



Representing Deferred Substitution

Change

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

to

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

Representing Deferred Substitution

Change

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

to

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

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

Interp with DefrdSub

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

Interp with DefrdSub

```
(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 with DefrdSub

```
(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 with DefrdSub

```
(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 Deferred Substitutions

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


WAE Interpreter with Deferred Substitutions

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

WAE Interpreter with Deferred Substitutions

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

WAE Interpreter with Deferred Substitutions

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

WAE Interpreter with Deferred Substitutions

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

WAE Interpreter with Deferred Substitutions

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

WAE Interpreter with Deferred Substitutions

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

Function Calls

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

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



Function Calls

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

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



⇒

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



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

Function Calls

What goes wrong if you extend the old substitution?

```
{defun {f x} {+ y x}}
```

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



Function Calls

What goes wrong if you extend the old substitution?

```
{defun {f x} {+ y x}}
```

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

⇒

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

Function Calls

What goes wrong if you extend the old substitution?

```
{defun {f x} {+ y x}}
```

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

⇒

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

⇒

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

⇒ 12 wrong!

Function Calls

What goes wrong if you extend the old substitution?

```
{defun {f x} {+ y x}}
```

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

⇒

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

⇒

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

⇒ "free var: y"

FIWAE Interpreter with Deferred Substitutions

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

FIWAE Interpreter with Deferred Substitutions

```
; interp : F1WAE list-of-FunDef DefrdSub -> num
(define (interp a-flwae fundefs ds)
  (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 ds)
                         ...)))]))
```


FIWAE Interpreter with Deferred Substitutions

```
; interp : FlWAE list-of-FunDef DefrdSub -> num
(define (interp a-flwae fundefs ds)
  (type-case FlWAE 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 ds)
                              (mtSub)))))]))
```

Timing tests

```
(define (mk-sums n)
  (cond
    [(zero? n) 1]
    [else
     (let ([varn (string->symbol (format "x~a" n))])
       `{+ ,varn , (mk-sums (- n 1))}))])

(define (mk-withs n body)
  (cond
    [(zero? n) body]
    [else
     (let ([varn (string->symbol (format "x~a" n))])
       `{with {,varn 1}
           , (mk-withs (- n 1) body)}))])
```

Timing tests, 2

```
(define (mk-exp n) (mk-withs n (mk-sums n)))

(test (mk-exp 2)
      `{with {x2 1}
        {with {x1 1}
          {+ x2 {+ x1 1}}}})

(define (run n)
  (let ([expr (parse (mk-exp n))])
    (time (interp-expr expr ' ())))))
```

Timing tests, 3

With the substitution-based interpreter, expect the difference between adjacent timings to be growing linearly. With the environment-based one, you will also see linear growth, but if you make the environment use a more efficient data structure, that'll go away

(you may need to make the numbers bigger or smaller to see what is going on here)

```
(collect-garbage) (collect-garbage)
(collect-garbage) (collect-garbage)
(run 100) (run 110) (run 120)
(run 130) (run 140) (run 160)
```

Evenness

```
; even? : num[pos] -> num[bool]
{def fun {even? n}
  {if0 n
    0
    {if0 {- n 1}
      1
      {even? {- n 2}}}}}}
; div2 : even? -> num[bool]
{def fun {div2 n}
  {if0 n
    0
    {+ 1 {div2 {- n 2}}}}}}
```

Evenness tests

```
(test (interp (parse `{even? 0})) 0)
(test (interp (parse `{even? 1})) 1)
(test (interp (parse `{even? 2})) 0)
(test (interp (parse `{div2 0})) 0)
(test (interp (parse `{div2 2})) 1)
(test (interp (parse `{div2 12})) 6)
```

Collatz numbers

$n \rightarrow n/2$, if n is even

$n \rightarrow 3n+1$, if n is odd

terminate when n is 1

```
(test (interp (parse `{orbit 1})) 0)
(test (interp (parse `{orbit 2})) 1)
(test (interp (parse `{orbit 3})) 7)
(test (interp (parse `{orbit 30})) 18)
(test (interp (parse `{orbit 31})) 106)
(test (interp (parse `{orbit 32})) 5)
```

Collatz numbers

```
; orbit : number[>0] -> number
; return the length of the collatz orbit
{def fun {orbit n}
  {if0 {- n 1}
    0
    {+ 1 {if0 {even? n}
      {orbit {div2 n}}
      {orbit {+ {+ n n}
        {+ n 1}}}}}}}}}}
```