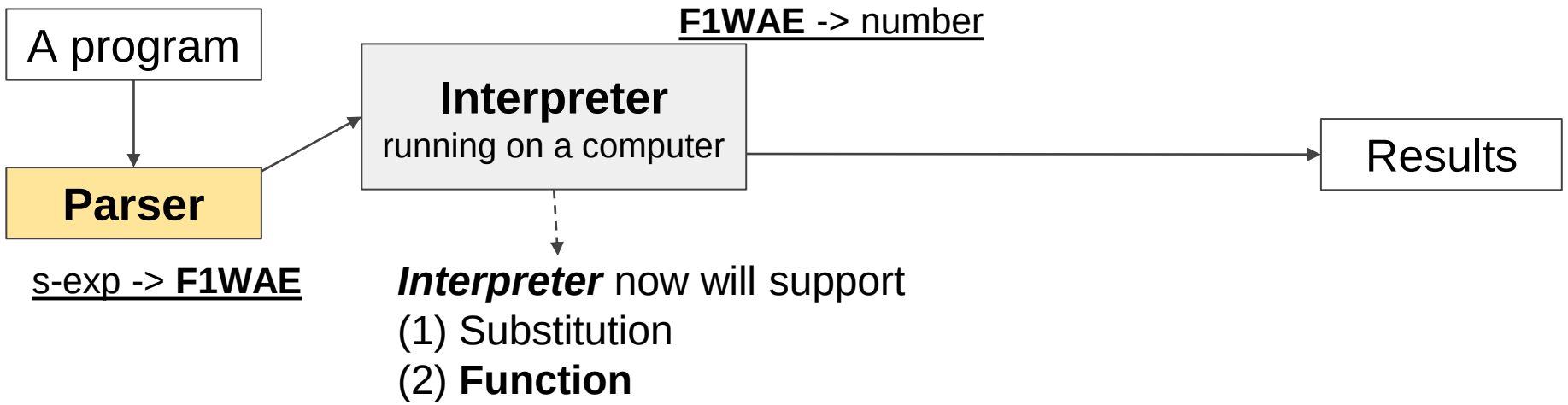
ITP20005 L9

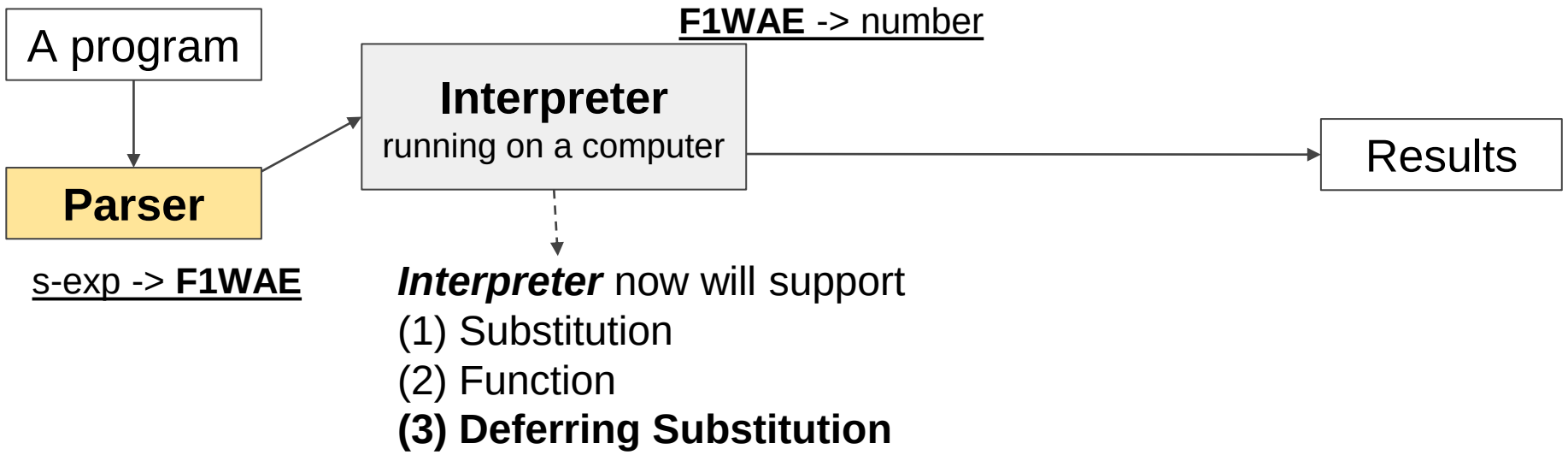
Deferring Substitution

Lecture09
JC

Big Picture (modeling languages: substitution)



Big Picture (modeling languages: substitution)



See this program

```
{with {x 3}
     {with {y 4}
          {with {z 5}
               {+ x {+ y z}}}}}
```

See this program

```
{with {x 3}
      {with {y 4}
            {with {z 5}
                  {+ x {+ y z}}}}}
= {with {y 4}
      {with {z 5}
            {+ 3 {+ y z}}}}
```

See this program

```
{with {x 3}
      {with {y 4}
            {with {z 5}
                  {+ x {+ y z}}}}}
= {with {y 4}
      {with {z 5}
            {+ 3 {+ y z}}}}
= {with z 5}
      {+ 3 {+ 4 z}}}
```

See this program

```
{with {x 3}
      {with {y 4}
            {with {z 5}
                  {+ x {+ y z}}}}}
= {with {y 4}
      {with {z 5}
            {+ 3 {+ y z}}}}
= {with z 5}
      {+ 3 {+ 4 z}}}
= {+ 3 {+ 4 5}}
```

See this program

```
{with {x 3}
      {with {y 4}
            {with {z 5}
                  {+ x {+ y z}}}}}
= {with {y 4}
      {with {z 5}
            {+ 3 {+ y z}}}}
= {with z 5}
      {+ 3 {+ 4 z}}}
= {+ 3 {+ 4 5}}
```

Complexity of the interpreter for this program:

$O(n^2)$

See this program

```
{with {x 3}
      {with {y 4}
            {with {z 5}
                  {+ x {+ y z}}}}}
= {with {y 4}
      {with {z 5}
            {+ 3 {+ y z}}}}
= {with z 5}
      {+ 3 {+ 4 z}}}
= {+ 3 {+ 4 5}}
```

WE CAN do BETTER??

Let's improve Substitution!

⇒ Our new language
AE → WAE → F1WAE with better substitution

Substitution

```
; [contract] subst: WAE symbol number -> WAE
(define (subst wae idtf val)
    (type-case WAE wae
        [num    (n)         wae]
        [add    (l r)       (add (subst l idtf val) (subst r idtf val))]
        [sub    (l r)       (sub (subst l idtf val) (subst r idtf val))]
        [with   (i v e)     (with i (subst v idtf val)
                                    (if (symbol=? i idtf)
                                        e
                                        (subst e idtf val)))]
        [id     (s)         (if (symbol=? s idtf) (num val) wae)]))
```

```
; {with {x 10} {...{with {y 17} x}} ⇒ 10 for x in {with {y 17} x}     ⇒ {with {y 17} 10}
(test (subst (with 'y (num 17) (id 'x)) 'x 10) (with 'y (num 17) (num 10)))
; {with {x 10} {...{with {x x} x}}}  ⇒ 10 for x in {with {x x} x}     ⇒ {with {x 10} x}
(test (subst (with 'x (id 'x) (id 'x)) 'x 10) (with 'x (num 10) (id 'x)))
```

Cost of Substitution

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

Cost of Substitution

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

Cost of Substitution

```
(interp (parse '{with {x 1}
                      {with {y 2}
                            {+ 100 {+ 99 {+ 98 … {+ y x}…}}))
⇒
(interp (parse '{with {y 2}
                      {+ 100 {+ 99 {+ 98 … {+ y 1} …}}))

⇒
(interp (parse '{+ 100 {+ 99 {+ 98 … {+ 2 1} …}))
```

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

* n = the number of abstract syntax tree nodes of a program

How can we do better?

Do you have any idea?

Deferring Substitution

```
(interp (parse '{with {x 1}          [ ]
                  {with {y 2}
                    {+ 100 {+ 99 {+ 98 … {+ y x}...}}))
```

Deferring Substitution

(interp (parse '{with {x 1} [ ]
  {with {y 2}
    {+ 100 {+ 99 {+ 98 … {+ y x}...}}))

⇒

(interp (parse '{with {y 2} [x=1]
  {+ 100 {+ 99 {+ 98 … {+ y x} …}}))

⇒

(interp (parse '{+ 100 {+ 99 {+ 98 … {+ y x} …}))

Deferring Substitution

(interp (parse '{with {x 1} [ ]
  {with {y 2}
    {+ 100 {+ 99 {+ 98 … {+ y x}...}}))

⇒

(interp (parse '{with {y 2} [x=1]
  {+ 100 {+ 99 {+ 98 … {+ y x} …}}))

⇒

(interp (parse '{+ 100 {+ 99 {+ 98 … {+ y x} …} [y=2 x=1]))

Deferring Substitution

(interp (parse '{with {x 1} [ ]
  {with {y 2}
    {+ 100 {+ 99 {+ 98 … {+ y x}…}}))

⇒

(interp (parse '{with {y 2} [x=1]
  {+ 100 {+ 99 {+ 98 … {+ y x} …}}))

⇒

(interp (parse '{+ 100 {+ 99 {+ 98 … {+ y x} …} [y=2 x=1]))

⇒ … ⇒

(interp (parse 'y [y=2 x=1]))

Deferring Substitution

```
(interp (parse '{with {x 1}          [ ]
                  {with {x 2}
                    x}}))
⇒
(interp (parse '{with {x 2}          [x=1]
                  x}))
⇒
(interp (parse 'x          [x=2 x=1] ))
```

Always add to start, then always check from start

Deferring Substitution

```
(interp (parse '{with {x 1}          [ ]
                  {+ {with {x 2} x}
                     x}}))
```

Deferring Substitution

```
(interp (parse '{with {x 1}            [ ]
                  {+ {with {x 2} x}
                     x}}))
```

⇒

```
(interp (parse '{+ {with {x 2} x}      [x=1]
                  x}))
```

Deferring Substitution

```
(interp (parse '{with {x 1}            [ ]
                  {+ {with {x 2} x}
                     x}}))
⇒
(interp (parse '{+ {with {x 2} x}      [x=1]
                  x}))
⇒
(+      (interp (parse '{with {x 2} x}     [x=1] ))
    (interp (parse 'x   [x=1])))
```

Deferring Substitution

```
(interp (parse '{with {x 1}            [ ]
                  {+ {with {x 2} x}
                     x}}))
⇒
(interp (parse '{+ {with {x 2} x}     [x=1]
                   x}))
⇒
(+      (interp (parse '{with {x 2} x}     [x=1] ))
    (interp (parse 'x   [x=1])))
⇒
(+ (interp (parse 'x  [x=2 x=1])) (interp (parse 'x [x=1])))
```

Deferring Substitution

```
(interp (parse '{with {x 1}            [ ]
                  {+ {with {x 2} x}
                     x}}))
⇒
(interp (parse '{+ {with {x 2} x}     [x=1]
                   x}))
⇒
(+      (interp (parse '{with {x 2} x}     [x=1] ))
    (interp (parse 'x   [x=1])))
⇒
(+ (interp (parse 'x  [x=2 x=1])) (interp (parse 'x [x=1])))
⇒ (+ 2 1)
```

Representing Deferred Substitution

Change

; interp : WAE -> number

to

; interp : WAE DefrdSub -> number

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

; example instance

(aSub 'x 1 (aSub 'y 4 (aSub 'x 2 (mtSub))))

* mtSub: mt stands for 'empty' cache (repository)
* aSub: non-empty cache, a pair of an identifier and a value for substitution and the next pair.

interp **with** DefrdSub

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

interp **with** DefrdSub

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

interp **with** DefrdSub

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

interp **with** DefrdSub

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

interp **with** DefrdSub

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

interp **with** DefrdSub

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

WAE **Interpreter with** DefrdSub

```
; interp : WAE DefrdSub -> number
(define (interp wae ds)
    (type-case WAE wae
        [num    (n)         n]
        [add    (l r)       (+ (interp l ds) (interp r ds))]
        [sub    (l r)       (- (interp l ds) (interp r ds))]
        [with   (i v e)     ...]
        [id     (s)         ...]))
```

WAE **Interpreter without** DefrdSub

```
; interp : WAE DefrdSub -> number
(define (interp wae ds)
    (type-case WAE wae
        [num    (n)       n]
        [add    (l r)     (+ (interp l ds) (interp r ds))]
        [sub    (l r)     (- (interp l ds) (interp r ds))]
        [with   (i v e)   (interp (subst e i (interp v)))]
        [id     (s)       (error 'interp "free identifier")])
```

For deferred substitution, we need a helper function to lookup a value of the id, s from ds!!

WAE **Interpreter with** DefrdSub

```
; interp : WAE DefrdSub -> number
(define (interp wae ds)
    (type-case WAE wae
        [num    (n)        n]
        [add    (l r)      (+ (interp l ds) (interp r ds))]
        [sub    (l r)      (- (interp l ds) (interp r ds))]
        [with   (i v e)    ...]
        [id     (s)        (lookup s ds)]))
```

WAE **Interpreter with** DefrdSub

```
; lookup: symbol DefrdSub -> number
(define (lookup name ds)
    (type-case DefrdSub ds
        [mtSub      ()              (error 'lookup "free identifier")]
        [aSub     (i v saved)     (if (symbol=? i name)
                                      v
                                      (lookup name saved))]))
```

```
(test (lookup 'x (aSub 'x 1 (mtSub))) 1)
(test (lookup 'y (aSub 'x 1 (aSub 'y 4 (mtSub)))) 4)
```

WAE **Interpreter with** DefrdSub

```
; interp : WAE DefrdSub -> number
(define (interp wae ds)
    (type-case WAE wae
        [num    (n)        n]
        [add    (l r)      (+ (interp l ds) (interp r ds))]
        [sub    (l r)      (- (interp l ds) (interp r ds))]
        [with   (i v e)    ...]
        [id     (s)        (lookup s ds)]))
```

WAE **Interpreter with** DefrdSub

```
; interp : WAE DefrdSub -> number
(define (interp wae ds)
    (type-case WAE wae
        [num     (n)        n]
        [add     (l r)      (+ (interp l ds) (interp r ds))]
        [sub     (l r)      (- (interp l ds) (interp r ds))]
        [with    (i v e)    … (interp v ds) … ]
        [id      (s)        (lookup s ds)]))
```

WAE **Interpreter with** DefrdSub

```
; interp : WAE DefrdSub -> number
(define (interp wae ds)
    (type-case WAE wae
        [num    (n)        n]
        [add    (l r)      (+ (interp l ds) (interp r ds))]
        [sub    (l r)      (- (interp l ds) (interp r ds))]
        [with   (i v e)    … (aSub i (interp v ds) ds) … ]
        [id     (s)        (lookup s ds)]))
```

WAE **Interpreter with** DefrdSub

```
; interp : WAE DefrdSub -> number
(define (interp wae ds)
    (type-case WAE wae
        [num    (n)        n]
        [add    (l r)      (+ (interp l ds) (interp r ds))]
        [sub    (l r)      (- (interp l ds) (interp r ds))]
        [with   (i v e)    (interp e (aSub i (interp v ds) ds))]
        [id     (s)        (lookup s ds)]))
```

Function Calls

```
{deffun {f x} {+ 1 x}}
(interp (parse '{with {y 2} {f 10}}          [ ]))
```

Function Calls

```
{deffun {f x} {+ 1 x}}
(interp (parse '{with {y 2} {f 10}}          [ ]))
⇒
(interp (parse '{f 10}      [y=2]))
```

Function Calls

{deffun {f x} {+ 1 x}}
(interp (parse '{with {y 2} {f 10}} [ ]))
⇒
(interp (parse '{f 10} [y=2]))
⇒
(interp (parse '{+ 1 x} [...]))

Function Calls

{deffun {f x} {+ 1 x}}
(interp (parse '{with {y 2} {f 10}} [ ]))
⇒
(interp (parse '{f 10} [y=2]))
⇒
(interp (parse '{+ 1 x} [...]))

Interpreting function body starts with only one substitution

Function Calls

What goes wrong if you extend the old substitution?

```
{deffun {f x} {+ y x}}
(interp (parse '{with {y 2} {f 10}}          [ ]))
```

Function Calls

What goes wrong if you extend the old substitution?

{deffun {f x} {+ y x}}
(interp (parse '{with {y 2} {f 10}} [ ]))
⇒
(interp (parse '{f 10} [y=2]))

Function Calls

What goes wrong if you extend the old substitution?

{deffun {f x} {+ y x}}
(interp (parse '{with {y 2} {f 10}} [ ]))
⇒
(interp (parse '{f 10} [y=2]))
⇒
(interp (parse '{+ y x} [x=10 y=2]))
⇒
12 Wrong!

Function Calls

What goes wrong if you extend the old substitution?

{deffun {f x} {+ y x}}
(interp (parse '{with {y 2} {f 10}} [ ]))
⇒
(interp (parse '{f 10} [y=2]))
⇒
(interp (parse '{+ y x} [x=10]))
⇒
"free var: y"

Interpreting function body starts with only one substitution

Scope

- Static scope
  In a language with static scope, the scope of an identifier's binding is a syntactically delimited region.

- Dynamic scope
  In a language with dynamic scope, the scope of an identifier's binding is the entire remainder of the execution during which that binding is in effect.

Scope

```
{deffun {f p} n}
{with {n 5} {f 10}
```

- Static scope
  In a language with static scope, the scope of an identifier's binding is a syntactically delimited region.

- Dynamic scope
  In a language with dynamic scope, the scope of an identifier's binding is the entire remainder of the execution during which that binding is in effect.

Scope

```
{deffun {f p} n}
{with {n 5} {f 10}}
```

- Static scope
  In a language with static scope, the scope of an identifier's binding is a syntactically delimited region.
  The code signals an error.

- Dynamic scope
  In a language with dynamic scope, the scope of an identifier's binding is the entire remainder of the execution during which that binding is in effect.
  The code evaluates to 5.

Static and dynamic scope in Perl

```
$x = 50;

sub fun2 {
   return $x;
}

sub fun1 {
   my $x = 10;
   my $y = fun2();
   return $y;
}

print fun1();
```

Static scope: Output 50

```
$x = 50;

sub fun2 {
   return $x;
}

sub fun1 {
   local $x = 10;
   my $y = fun2();
   return $y;
}

print fun1();
```

dynamic scope: Output 10

Static scope only in C

```
int  x = 50;

int fun2() {
   return x;
}

int fun1() {
   int x = 10;
   int y = fun2();
   return y;
}

int main() {
   printf("%d", fun1());
}
```

Static scope: Output 50

Which one is more reasonable?
Why??

F1WAE Interpreter with Defrdsub

```
; interp : F1WAE list-of-FucDef DefrdSub -> number
(define (interp f1wae fundefs ds
    (type-case F1WAE f1wae
    ...
    [app (ftn arg)
                ...]))
```

F1WAE Interpreter with Defrdsub

```
; interp : F1WAE list-of-FucDef DefrdSub -> number
(define (interp f1wae fundefs ds
    (type-case F1WAE f1wae
    …
    [app (ftn arg)
        (local     [(define a-fundef (lookup-fundef ftn fundefs))]
          (interp     (fundef-body a-fundef)
                    fundefs
                    …
                    (interp arg fundefs ds)
                    … ))]))
```

F1WAE Interpreter with Defrdsub

```
; interp : F1WAE list-of-FucDef DefrdSub -> number
(define (interp f1wae fundefs ds
    (type-case F1WAE f1wae
    ...
    [app (ftn arg)
        (local     [(define a-fundef (lookup-fundef ftn fundefs))]
          (interp     (fundef-body a-fundef)
                  fundefs
                  (aSub (fundef-arg-name a-fundef)
                      (interp arg fundefs ds)
                      (_____)))))]))
```

F1WAE Interpreter with Defrdsub

```
; interp : F1WAE list-of-FucDef DefrdSub -> number
(define (interp f1wae fundefs ds
    (type-case F1WAE f1wae
    ...
    [app (ftn arg)
        (local     [(define a-fundef (lookup-fundef ftn fundefs))]
           (interp     (fundef-body a-fundef)
                       fundefs
                       (aSub (fundef-arg-name a-fundef)
                            (interp arg fundefs ds)
                            (mtSub)))))]))
```

Add substitution info
for the function parameter

```
(test (interp (parse '{f 1}) (list (parse-fd '{deffun (f x) {+ x 3}})) (mtSub)) 4)
```

F1WAE Interpreter with Defrdsub

```
; interp : F1WAE list-of-FucDef DefrdSub -> number
(define (interp f1wae fundefs ds)
 (type-case F1WAE f1wae
  [num  (n)      n]
  [add  (l r)    (+ (interp l fundefs ds) (interp r fundefs ds))]
  [sub  (l r)    (- (interp l fundefs ds) (interp r fundefs ds))]
  [with (i v e)  (interp e fundefs (aSub i (interp v fundefs ds) ds))]
  [id     (s)    (lookup s ds)]
  [app  (f a)    (local
                                [(define a-fundef (lookup-fundef f fundefs))]
                                (interp (fundef-body a-fundef)
                                    fundefs
                                (aSub (fundef-arg-name a-fundef)
                                            (interp a fundefs ds)
                                                (mtSub))
                        ))])))

(test (interp (parse '{f 1}) (list (parse-fd '{deffun (f x) {+ x 3}})) (mtSub)) 4)
```

Compare with F1WAE Interpreter in L08

```
; [contract] subst: F1WAE symbol number -> F1WAE
(define (subst f1wae idtf val)
        (type-case F1WAE f1wae
                [num    (n)             f1wae]
                [add    (l r)           (add (subst l idtf val) (subst r idtf
val))]
                [sub            (l r)           (sub (subst l idtf val) (subst r
idtf val))]
                [with   (i v e)   (with i (subst v idtf val) (if (symbol=? i idtf) e

        (subst e idtf val)))]
                [id             (s)             (if (symbol=? s idtf) (num val)
f1wae)]
                [app    (f a)           (app f    (subst a idtf val))]))
```

Topics we cover and schedule (tentative)

- Racket tutorials (L2,3, HW)
- Modeling languages (L4,5, HW
- Interpreting arithmetic (L5)
- Language principles
  - **Substitution** (L6,7, HW)
  - **Function** (L8)
  - **Deferring Substitution** (L9)
  - First-class Functions (L10,11,HW)
  - Laziness (L11,12)
  - Recursion (L13,14)
  - Representation choices (L15)
  - Mutable data structures (L16, HW)
  - Variables (L17, HW)
  - Continuations (L18,19,20,21, HW)
  - Garbage collection (L22)
  - Semantics (L23,24)
  - Type (L25,26,27, HWs)
- Guest Video Lecture (L28)

No class: October 2 (Fri, Chuseok), October 9 (Fri, Hangul day)
Online only class can be provided.

TODO

Read Chapter 6. First-Class Functions

JC

jcnam@handong.edu

https://lifove.github.io

* Slides are from Prof. Sukyoung Ryu's PL class in 2018 Spring or created by JC based on the main text book.