Why Functions as Values

- Abstraction is easier with functions as values

  ○ abstract over `add` and `sub` cases

  ○ `filter`, `map`, etc.

Why Functions as Values

- Abstraction is easier with functions as values

  ○ abstract over **add** and **sub** cases

  ○ **filter**, **map**, etc.

- What are objects? Callbacks?

Why Functions as Values

- Abstraction is easier with functions as values

    ○ abstract over **add** and **sub** cases

    ○ **filter**, **map**, etc.

- What are objects? Callbacks?

- Separate **deffun** form becomes unnecessary

    ○
    ```
    {deffun {f x} {+ 1 x}}
    {f 10}
    ⟹
    {with {f {fun {x} {+ 1 x}}}
      {f 10}}
    ```

FWAE Grammar, Almost

```
<FWAE> ::=  <num>
         |  {+ <FWAE> <FWAE>}
         |  {- <FWAE> <FWAE>}
         |  {with {<id> <FWAE>} <FWAE>}
         |  <id>
         |  {<id> <FWAE>}              ?
         |  {fun {<id>} <FWAE>}       NEW
```

FWAE Evaluation

```
10 ⟹ 10

{+ 1 2} ⟹ 3

{- 1 2} ⟹ -1

{with {x 7} {+ x 2}} ⟹ {+ 7 2} ⟹ 9
```

y ⟹ *free variable*

FWAE Evaluation

```
10  ⇒  10

{+ 1 2}  ⇒  3

{- 1 2}  ⇒  -1

{with {x 7} {+ x 2}}  ⇒  {+ 7 2}  ⇒  9

y  ⇒  free variable

{fun {x} {+ 1 x}}  ⇒
```

FWAE Evaluation

```
10  ⟹  10

{+ 1 2}  ⟹  3

{- 1 2}  ⟹  -1

{with {x 7} {+ x 2}}  ⟹  {+ 7 2}  ⟹  9
```

y ⟹ *free variable*

```
{fun {x} {+ 1 x}}  ⟹
{fun {x} {+ 1 x}}
```

FWAE Evaluation

```
10  ⇒  10

{+ 1 2}  ⇒  3

{- 1 2}  ⇒  -1

{with {x 7} {+ x 2}}  ⇒  {+ 7 2}  ⇒  9

y  ⇒  free variable

{fun {x} {+ 1 x}}  ⇒
{fun {x} {+ 1 x}}
```

Result is not always a number!

```
; interp FWAE ... -> FWAE-Value
```

FWAE Evaluation

`{with {y 10} {fun {x} {+ y x}}}`

$\Rightarrow$

FWAE Evaluation

```
{with {y 10} {fun {x} {+ y x}}}

⟹  {fun {x} {+ 10 x}}
```

FWAE Evaluation

```
    {with {y 10} {fun {x} {+ y x}}}

 ⟹  {fun {x} {+ 10 x}}

{with {f {fun {x} {+ 1 x}}}
  {f 3}}
 ⟹
```

FWAE Evaluation

```
{with {y 10} {fun {x} {+ y x}}}

⇒  {fun {x} {+ 10 x}}

{with {f {fun {x} {+ 1 x}}}
  {f 3}}
  ⇒  {{fun {x} {+ 1 x}} 3}
```

FWAE Evaluation

```
{with {y 10} {fun {x} {+ y x}}}

⟹  {fun {x} {+ 10 x}}

{with {f {fun {x} {+ 1 x}}}
  {f 3}}
⟹  {{fun {x} {+ 1 x}} 3}
```

Doesn't match the grammar for **<FWAE>**

FWAE Grammar

```
<FWAE> ::= <num>
         | {+ <FWAE> <FWAE>}
         | {- <FWAE> <FWAE>}
         | {with {<id> <FWAE>} <FWAE>}
         | <id>
         | {<id> <FWAE>}
         | {fun {<id>} <FWAE>}        NEW
         | {<FWAE> <FWAE>}            NEW
```

FWAE Evaluation

```
{with {f {fun {x} {+ 1 x}}} {f 3}}
  ⟹  {{fun {x} {+ 1 x}} 3}
  ⟹  {+ 1 3}  ⟹  4
```

FWAE Evaluation

```
{with {f {fun {x} {+ 1 x}}} {f 3}}
 ⇒  {{fun {x} {+ 1 x}} 3}
 ⇒  {+ 1 3}  ⇒  4

{{fun {x} {+ 1 x}} 3}  ⇒  {+ 1 3}  ⇒  4
```

FWAE Evaluation

```
{with {f {fun {x} {+ 1 x}}} {f 3}}
 ⟹  {{fun {x} {+ 1 x}} 3}
 ⟹  {+ 1 3}  ⟹  4

{{fun {x} {+ 1 x}} 3}  ⟹  {+ 1 3}  ⟹  4

{1 2}  ⟹
```

FWAE Evaluation

```
{with {f {fun {x} {+ 1 x}}} {f 3}}
 ⟹  {{fun {x} {+ 1 x}} 3}
 ⟹  {+ 1 3}  ⟹  4

{{fun {x} {+ 1 x}} 3}  ⟹  {+ 1 3}  ⟹  4

{1 2}  ⟹  not a function
```

FWAE Evaluation

```
{with {f {fun {x} {+ 1 x}}} {f 3}}
 ⟹  {{fun {x} {+ 1 x}} 3}
 ⟹  {+ 1 3}  ⟹  4

{{fun {x} {+ 1 x}} 3}  ⟹  {+ 1 3}  ⟹  4

{1 2}  ⟹  not a function

{+ 1 {fun {x} 10}}  ⟹
```

FWAE Evaluation

```
{with {f {fun {x} {+ 1 x}}} {f 3}}
 ⟹ {{fun {x} {+ 1 x}} 3}
 ⟹ {+ 1 3}  ⟹  4

{{fun {x} {+ 1 x}} 3}  ⟹  {+ 1 3}  ⟹  4

{1 2}  ⟹  not a function

{+ 1 {fun {x} 10}}  ⟹  not a number
```

FWAE Datatype

```
(define-type FWAE
  [num (n number?)]
  [add (lhs FWAE?)
       (rhs FWAE?)]
  [sub (lhs FWAE?)
       (rhs FWAE?)]
  [with (name symbol?)
        (named-expr FWAE?)
        (body FWAE?)]
  [id (name symbol?)]
  [fun (param symbol?)
       (body FWAE?)]
  [app (fun-expr FWAE?)
       (arg-expr FWAE?)])
```

FWAE Datatype

```
(define-type FWAE
  [num (n number?)]
  [add (lhs FWAE?)
       (rhs FWAE?)]
  [sub (lhs FWAE?)
       (rhs FWAE?)]
  [with (name symbol?)
        (named-expr FWAE?)
        (body FWAE?)]
  [id (name symbol?)]
  [fun (param symbol?)
       (body FWAE?)]
  [app (fun-expr FWAE?)
       (arg-expr FWAE?)])


(test (parse '{fun {x} {+ x 1}})
      (fun 'x (add (id 'x) (num 1))))
```

FWAE Datatype

```
(define-type FWAE
  [num (n number?)]
  [add (lhs FWAE?)
       (rhs FWAE?)]
  [sub (lhs FWAE?)
       (rhs FWAE?)]
  [with (name symbol?)
        (named-expr FWAE?)
        (body FWAE?)]
  [id (name symbol?)]
  [fun (param symbol?)
       (body FWAE?)]
  [app (fun-expr FWAE?)
       (arg-expr FWAE?)])


(test (parse '{{fun {x} {+ x 1}} 10})
      (app (fun 'x (add (id 'x) (num 1))) (num 10)))
```

FWAE-Value

```
(define-type FWAE-Value
  [numV (n number?)]
  [funV (param symbol?)
        (body FWAE?)])
```

FWAE Interpreter

```
; interp : FWAE -> FWAE-Value
(define (interp a-wae)
  (type-case FWAE a-wae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l) (interp r))]
    [sub (l r) (num- (interp l) (interp r))]
    [with (bound-id named-expr body-expr)
      (interp (subst body-expr
                     bound-id
                     (interp named-expr)))]
    [id (name) (error 'interp "free variable")]
    [fun (param body-expr)
         ...]
    [app (fun-expr arg-expr)
         ...])))
```

FWAE Interpreter

```
; interp : FWAE -> FWAE-Value
(define (interp a-wae)
  (type-case FWAE a-wae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l) (interp r))]
    [sub (l r) (num- (interp l) (interp r))]
    [with (bound-id named-expr body-expr)
      (interp (subst body-expr
                     bound-id
                     (interp named-expr)))]
    [id (name) (error 'interp "free variable")]
    [fun (param body-expr)
         (funV param body-expr)]
    [app (fun-expr arg-expr)
         ...])))
```

FWAE Interpreter

```
; interp : FWAE -> FWAE-Value
(define (interp a-wae)
  (type-case FWAE a-wae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l) (interp r))]
    [sub (l r) (num- (interp l) (interp r))]
    [with (bound-id named-expr body-expr)
      (interp (subst body-expr
                     bound-id
                     (interp named-expr)))]
    [id (name) (error 'interp "free variable")]
    [fun (param body-expr)
         (funV param body-expr)]
    [app (fun-expr arg-expr)
         ... (interp fun-expr)
         ... (interp arg-expr) ...]))
```

FWAE Interpreter

```
; interp : FWAE -> FWAE-Value
(define (interp a-wae)
  (type-case FWAE a-wae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l) (interp r))]
    [sub (l r) (num- (interp l) (interp r))]
    [with (bound-id named-expr body-expr)
      (interp (subst body-expr
                      bound-id
                      (interp named-expr)))]
    [id (name) (error 'interp "free variable")]
    [fun (param body-expr)
         (funV param body-expr)]
    [app (fun-expr arg-expr)
         (local [(define fun-val (interp fun-expr))]
           ... (funV-body fun-val) ...
           ... (funV-param fun-val) ...
           ... (interp arg-expr) ...)]))
```

FWAE Interpreter

```
; interp : FWAE -> FWAE-Value
(define (interp a-wae)
  (type-case FWAE a-wae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l) (interp r))]
    [sub (l r) (num- (interp l) (interp r))]
    [with (bound-id named-expr body-expr)
      (interp (subst body-expr
                     bound-id
                     (interp named-expr)))]
    [id (name) (error 'interp "free variable")]
    [fun (param body-expr)
         (funV param body-expr)]
    [app (fun-expr arg-expr)
         (local [(define fun-val (interp fun-expr))]
           (interp (subst (funV-body fun-val)
                          (funV-param fun-val)
                          (interp arg-expr))))]))
```

Add and Subtract

```
; num+ : FWAE-Value FWAE-Value -> FWAE-Value
(define (num+ x y)
  (numV (+ (numV-n x) (numV-n y))))

; num- : FWAE-Value FWAE-Value -> FWAE-Value
(define (num- x y)
  (numV (- (numV-n x) (numV-n y))))
```

Add and Subtract

```
; num+ : FWAE-Value FWAE-Value -> FWAE-Value
(define (num+ x y)
  (numV (+ (numV-n x) (numV-n y))))

; num- : FWAE-Value FWAE-Value -> FWAE-Value
(define (num- x y)
  (numV (- (numV-n x) (numV-n y))))
```

Better:

```
; num-op :
;  (num num -> num) ->
;  (FWAE-Value FWAE-Value -> FWAE-Value)
(define (num-op op)
  (lambda (x y)
    (numV (op (numV-n x) (numV-n y)))))

(define num+ (num-op +))
(define num- (num-op -))
```

FWAE Subst

```
; subst : FWAE symbol FWAE-Value -> FWAE
(define (subst exp sub-id val)
  (type-case FWAE exp
    ...
    [id (name)
      (cond
        [(equal? name sub-id)
         (type-case FWAE-Value val
           [numV (n) (num n)]
           [funV (param body) (fun param body)])
        [else exp]])]
    ...))
```

FWAE Subst

```
; subst : FWAE symbol FWAE-Value -> FWAE
(define (subst exp sub-id val)
  (type-case FWAE exp
    ...
    [app (f arg)
        (app (subst f sub-id val)
             (subst arg sub-id val))]
    [fun (id body)
        (if (equal? sub-id id)
            exp
            (fun id (subst body sub-id val)))]))
```

FWAE Subst

Beware: with the implementation on the previous slide,

```
(subst {with {y 10} z}
       'z
       {fun {x} {+ x y}} )
⇒  {with {y 10} {fun {x} {+ x y}}}
```

which is wrong, but we ignore this problem

- Only happens when the original program has free variables

- The problem disappears with deferred substitution, anyway

No More With

Compare the **with** and **app** implementations:

```
(define (interp a-wae)
  (type-case FWAE a-wae
    ...
    [with (bound-id named-expr body-expr)
      (interp (subst body-expr
                     bound-id
                     (interp named-expr)))]
    ...
    [app (fun-expr arg-expr)
         (local [(define fun-val (interp fun-expr))]
            (interp (subst (funV-body fun-val)
                           (funV-param fun-val)
                           (interp arg-expr))))]))
```

The **app** case does everything that **with** does

No More With

```
{with {x 10} x}
```

is the same as

```
{{fun {x} x} 10}
```

No More With

$$\texttt{\{with \{x 10\} x\}}$$

is the same as

$$\texttt{\{\{fun \{x\} x\} 10\}}$$

In general,

$$\texttt{\{with \{<id> <FWAE>}_1\texttt{\} <FWAE>}_2\texttt{\}}$$

is the same as

$$\texttt{\{\{fun \{<id>\} <FWAE>}_2\texttt{\} <FWAE>}_1\texttt{\}}$$

No More With

```
{with {x 10} x}
```

is the same as

```
{{fun {x} x} 10}
```

In general,

```
{with {<id> <FWAE>₁} <FWAE>₂}
```

is the same as

```
{{fun {<id>} <FWAE>₂} <FWAE>₁}
```

Let's assume

```
(test {with {<id> <FWAE>₁} <FWAE>₂}
      (app (fun '<id> <FWAE>₂) <FWAE>₁))
```

FAE Grammar

```
<FAE> ::= <num>
        | {+ <FAE> <FAE>}
        | {- <FAE> <FAE>}
        | {with {<id> <FAE>} <FAE>}
        | <id>
        | {fun {<id>} <FAE>}
        | {<FAE> <FAE>}
```

FAE Grammar

```
<FAE>  ::=   <num>
        |   {+ <FAE> <FAE>}
        |   {- <FAE> <FAE>}
        |   {with {<id> <FAE>} <FAE>}
        |   <id>
        |   {fun {<id>} <FAE>}
        |   {<FAE> <FAE>}
```

- We'll still use **with** in boxes

- No more case lines in **interp**, etc. for **with**

- No more test cases for **interp**, etc. using **with**

FAE Interpreter

```
; interp : FAE -> FAE
(define (interp a-fae)
  (type-case FAE a-fae
    [num (n) a-fae]
    [add (l r) (num+ (interp l) (interp r))]
    [sub (l r) (num- (interp l) (interp r))]
    [id (name) (error 'interp "free variable")]
    [fun (param body-expr) a-fae]
    [app (fun-expr arg-expr)
         (local [(define fun-val (interp fun-expr))]
           (interp (subst (funV-body fun-val)
                          (funV-param fun-val)
                          (interp arg-expr))))]))
```

FAE with Deferred Substitution

`(interp {with {y 10} {fun {x} {+ y x}}} )`

FAE with Deferred Substitution

(interp {with {y 10} {fun {x} {+ y x}}})

$\Rightarrow$

y = 10

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

FAE with Deferred Substitution

(interp `{with {y 10} {fun {x} {+ y x}}}`)

$\Rightarrow$

(interp `{fun {x} {+ y x}}`)   y = 10

(interp `{{fun {y} {fun {x} {+ y x}}} 10}`)

44

FAE with Deferred Substitution

```
(interp  {with {y 10} {fun {x} {+ y x}}} )

⇒

(interp  {fun {x} {+ y x}} )          y = 10

(interp  {{fun {y} {fun {x} {+ y x}}} 10} )

⇒

(interp  {fun {x} {+ y x}} )          y = 10
```

FAE with Deferred Substitution

```
(interp  {{with {y 10} {fun {x} {+ y x}}}
          {with {y 7} y}}                      )
```

FAE with Deferred Substitution

(interp `{{with {y 10} {fun {x} {+ y x}}}`
`         {with {y 7} y}}`)

Argument expression:

(interp `{with {y 7} y}`)

⇒

(interp `y`)   ⇒   7

`y = 7`

FAE with Deferred Substitution

`(interp {{with {y 10} {fun {x} {+ y x}}}`
`       {with {y 7} y}} )`

Argument expression:

`(interp {with {y 7} y} )`

$\Rightarrow$

y = 7

`(interp y )` $\Rightarrow$ 7

Function expression:

`(interp {with {y 10} {fun {x} {+ y x}}} )`

$\Rightarrow$

y = 10

`(interp {fun {x} {+ y x}} )` $\Rightarrow$ **?**

FAE Values

A function value needs to keep its substitution cache

```
(define-type FWAE-Value
  [numV (n number?)]
  [closureV (param symbol?)
            (body FAE?)
            (ds DefrdSub?)])


(define-type DefrdSub
  [mtSub]
  [aSub (name symbol?)
        (value FWAE-Value?)
        (ds DefrdSub?)])
```

FAE Values

A function value needs to keep its substitution cache

```
(define-type FWAE-Value
  [numV (n number?)]
  [closureV (param symbol?)
            (body FAE?)
            (ds DefrdSub?)])


(define-type DefrdSub
  [mtSub]
  [aSub (name symbol?)
        (value FWAE-Value?)
        (ds DefrdSub?)])
```

```
(test (interp {with {y 10} {fun {x} {+ y x}}} )
      ...)
```

FAE Values

A function value needs to keep its substitution cache

```
(define-type FWAE-Value
  [numV (n number?)]
  [closureV (param symbol?)
            (body FAE?)
            (ds DefrdSub?)])


(define-type DefrdSub
  [mtSub]
  [aSub (name symbol?)
        (value FWAE-Value?)
        (ds DefrdSub?)])
```

```
(test (interp {with {y 10} {fun {x} {+ y x}}} )
      (closureV ... ... ...))
```

FAE Values

A function value needs to keep its substitution cache

```
(define-type FWAE-Value
  [numV (n number?)]
  [closureV (param symbol?)
            (body FAE?)
            (ds DefrdSub?)])

(define-type DefrdSub
  [mtSub]
  [aSub (name symbol?)
        (value FWAE-Value?)
        (ds DefrdSub?)])
```

```
(test (interp {with {y 10} {fun {x} {+ y x}}} )
      (closureV 'x {+ y x}
                (aSub 'y (num 10) (mtSub)))))
```
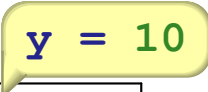
Continuing Evaluation

y = 10

Function: `{fun {x} {+ y x}}`

Argument: 7

Continuing Evaluation

Function: {fun {x} {+ y x}}   y = 10

Argument: 7

To apply, interpret the function body with the given argument:

```
(interp ...)
```

Continuing Evaluation

Function: `{fun {x} {+ y x}}`   ⟵ y = 10

Argument: 7

To apply, interpret the function body with the given argument:

`(interp {+ y x})`   ⟵ ...

Continuing Evaluation

Function: {fun {x} {+ y x}}   y = 10

Argument: 7

To apply, interpret the function body with the given argument:

(interp {+ y x})   x = 7   y = 10

FAE Interpreter with Deferred Substitution

```
; interp : FAE DefrdSub -> FWAE-Value
(define (interp a-fae ds)
  (type-case FAE a-fae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l ds) (interp r ds))]
    [sub (l r) (num- (interp l ds) (interp r ds))]
    [id (name) (lookup name ds)]
    [fun (param body-expr)
         ...]
    [app (fun-expr arg-expr)
         ...]))
```

FAE Interpreter with Deferred Substitution

```
; interp : FAE DefrdSub -> FWAE-Value
(define (interp a-fae ds)
  (type-case FAE a-fae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l ds) (interp r ds))]
    [sub (l r) (num- (interp l ds) (interp r ds))]
    [id (name) (lookup name ds)]
    [fun (param body-expr)
         (closureV param body-expr ds)]
    [app (fun-expr arg-expr)
         ...]))
```

FAE Interpreter with Deferred Substitution

```
; interp : FAE DefrdSub -> FWAE-Value
(define (interp a-fae ds)
  (type-case FAE a-fae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l ds) (interp r ds))]
    [sub (l r) (num- (interp l ds) (interp r ds))]
    [id (name) (lookup name ds)]
    [fun (param body-expr)
         (closureV param body-expr ds)]
    [app (fun-expr arg-expr)
         ... (interp fun-expr ds)
         ... (interp arg-expr ds) ...]))
```

FAE Interpreter with Deferred Substitution

```
; interp : FAE DefrdSub -> FWAE-Value
(define (interp a-fae ds)
  (type-case FAE a-fae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l ds) (interp r ds))]
    [sub (l r) (num- (interp l ds) (interp r ds))]
    [id (name) (lookup name ds)]
    [fun (param body-expr)
         (closureV param body-expr ds)]
    [app (fun-expr arg-expr)
         (local [(define fun-val
                    (interp fun-expr ds))
                 (define arg-val
                    (interp arg-expr ds))]
            ...)]))
```

FAE Interpreter with Deferred Substitution

```
; interp : FAE DefrdSub -> FWAE-Value
(define (interp a-fae ds)
  (type-case FAE a-fae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l ds) (interp r ds))]
    [sub (l r) (num- (interp l ds) (interp r ds))]
    [id (name) (lookup name ds)]
    [fun (param body-expr)
         (closureV param body-expr ds)]
    [app (fun-expr arg-expr)
         (local [(define fun-val
                   (interp fun-expr ds))
                 (define arg-val
                   (interp arg-expr ds))]
           (interp (closureV-body fun-val)
                   ...))]))
```

FAE Interpreter with Deferred Substitution

```
; interp : FAE DefrdSub -> FWAE-Value
(define (interp a-fae ds)
  (type-case FAE a-fae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l ds) (interp r ds))]
    [sub (l r) (num- (interp l ds) (interp r ds))]
    [id (name) (lookup name ds)]
    [fun (param body-expr)
         (closureV param body-expr ds)]
    [app (fun-expr arg-expr)
         (local [(define fun-val
                    (interp fun-expr ds))
                 (define arg-val
                    (interp arg-expr ds))]
           (interp (closureV-body fun-val)
                   (aSub (closureV-param fun-val)
                         arg-val
                         ...)))]))
```

FAE Interpreter with Deferred Substitution

```
; interp : FAE DefrdSub -> FWAE-Value
(define (interp a-fae ds)
  (type-case FAE a-fae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l ds) (interp r ds))]
    [sub (l r) (num- (interp l ds) (interp r ds))]
    [id (name) (lookup name ds)]
    [fun (param body-expr)
         (closureV param body-expr ds)]
    [app (fun-expr arg-expr)
         (local [(define fun-val
                    (interp fun-expr ds))
                 (define arg-val
                    (interp arg-expr ds))]
            (interp (closureV-body fun-val)
                    (aSub (closureV-param fun-val)
                          arg-val
                          (closureV-ds fun-val)))))]))
```