AE

```
{- 20 {+ 10 10}}

{- 20 {+ 17 17}}

{- 20 {+ 3 3}}
```

WAE

```
{with {x 10}
  {- 20 {+ x x}}}

{with {x 17}
  {- 20 {+ x x}}}

{with {x 3}
  {- 20 {+ x x}}}
```

F1WAE

```
{deffun {f x}
  {- 20 {+ x x}}}
```

```
{f 10}

{f 17}

{f 3}
```

F1WAE

```
{deffun {f x}
  {- 20 {twice x}}}

{deffun {twice y}
  {+ y y}}
```

```
{f 10}

{f 17}

{f 3}
```

```
; interp : F1WAE list-of-FunDef -> num
```

F1WAE

```
{deffun {f x}
  {- 20 {twice x}}}

{deffun {twice y}
  {+ y y}}
```

```
{f 10}

{f 17}

{f 3}
```

<FunDef> ::= **{deffun {<id> <id>} <F1WAE>}**

F1WAE

```
{deffun {f x}
  {- 20 {twice x}}}

{deffun {twice y}
  {+ y y}}
```

```
{f 10}

{f 17}

{f 3}
```

```
<FunDef>  ::=  {deffun {<id> <id>} <F1WAE>}
<F1WAE>   ::=  …
            |  {<id> <F1WAE>}
```

F1WAE Grammar

```
<FunDef>  ::=  {deffun {<id> <id>} <F1WAE>}

<F1WAE>   ::=  <num>
            |  {+ <F1WAE> <F1WAE>}
            |  {- <F1WAE> <F1WAE>}
            |  {with {<id> <F1WAE>} <WAE>}
            |  <id>
            |  {<id> <F1WAE>}
```

NEW

NEW

F1WAE Datatypes

```
(define-type FunDef
  [fundef (fun-name symbol?)
          (arg-name symbol?)
          (body F1WAE?)])

(define-type F1WAE
  [num (n number?)]
  [add (lhs F1WAE?)
       (rhs F1WAE?)]
  ...
  [app (fun-name symbol?)
       (arg F1WAE?)])
```

F1WAE Interp

```
(define (interp a-wae fundefs)
  (type-case F1WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (name arg-expr)
         ...]))


 (test (interp (add (num 1) (num 1))
               empty)
       2)
```

F1WAE Interp

```
(define (interp a-wae fundefs)
  (type-case F1WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (name arg-expr)
         ...]))


(test (interp (add (num 1) (num 1))
              (list
               (fundef 'f 'x
                       (add (id 'x) (num 3)))))
      2)
```

F1WAE Interp

```
(define (interp a-wae fundefs)
  (type-case F1WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (name arg-expr)
         ...]))


(test (interp (app 'f (num 1))
              (list
               (fundef 'f 'x
                       (add (id 'x) (num 3)))))
      4)
```

F1WAE Interp

```
(define (interp a-wae fundefs)
  (type-case F1WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (name arg-expr)
         ...]))

(test (interp (app 'f (num 10))
              (list
               (fundef 'f 'x
                       (sub (num 20)
                            (app 'twice (id 'x))))
               (fundef 'twice 'y
                       (add (id 'y) (id 'y))))
              0))
```

F1WAE Interp

```
(define (interp a-wae fundefs)
  (type-case F1WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (name arg-expr)
         ... (interp arg-expr fundefs) ...]))
```

F1WAE Interp

```
(define (interp a-wae fundefs)
  (type-case F1WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (name arg-expr)
         ... (lookup-fundef name fundefs)
         ... (interp arg-expr fundefs) ...]))

; lookup-fundef : symbol list-of-FunDef -> FunDef
```

F1WAE Interp

```
(define (interp a-wae fundefs)
  (type-case F1WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (name arg-expr)
         (local [(define a-fundef
                   (lookup-fundef name fundefs))]
           (interp (subst (fundef-body a-fundef)
                          (fundef-arg-name a-fundef)
                          (interp arg-expr fundefs))
                   fundefs))]))
```

Lookup

```
; lookup-fundef : symbol list-of-FunDef -> FunDef
(define (lookup-fundef name fundefs)
  ...)
```

Lookup

```
; lookup-fundef : symbol list-of-FunDef -> FunDef
(define (lookup-fundef name fundefs)
  (cond
    [(empty? fundefs)
     ...]
    [else
     ... (first fundefs)
     ... (lookup-fundef name (rest fundefs))
     ...]))
```

Lookup

```
; lookup-fundef : symbol list-of-FunDef -> FunDef
(define (lookup-fundef name fundefs)
  (cond
    [(empty? fundefs)
     (error 'interp "unknown function")]
    [else
     (if (symbol=? name (fundef-fun-name
                         (first fundefs)))
         (first fundefs)
         (lookup-fundef name (rest fundefs)))]))
```