AE

\{- \ 20 \ \{+ \ \ 10 \ \ 10\}\} \\
\{- \ 20 \ \{+ \ \ 17 \ \ 17\}\} \\
\{- \ 20 \ \{+ \ \ 3 \ \ 3\}\}
{with {x 10}
  {− 20 {+ x x}}}

{with {x 17}
  {− 20 {+ x x}}}

{with {x 3}
  {− 20 {+ x x}}}
{deffun {f x}
  {- 20 {+ {+ x x}
        {+ x x}}}}

{f 10}

{f 17}

{f 3}
{defun twice y}  {defun twice x}
{+ y y}            {- 20 {twice x}}
{f 3}              {f 10}
{f 17}
FIWAE

{deffun {f x}  {f 10}
  {- 20 {twice
    {twice x} }}  {f 17}

{deffun {twice y}  {f 3}
  {+ y y}}

; interp : FIWAE list-of-FunDef -> num
FIWAE

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

{deffun {twice y} {
    [+ y y]}}

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

{deffun {f x} {f 10}
  {- 20 {twice
    {twice x}}}]]} {f 17}

{deffun {twice y} {f 3}
  {+ y y}}

<FunDef> ::= {deffun {<id> <id>} <FIWAE>}
<FIWAE> ::= ...
  | {<id> <FIWAE>}


F1WAE Grammar

\[<\text{FunDef}> ::= \{\text{deffun} \ [<\text{id}> \ <\text{id}>] \ <\text{F1WAE}>\}\]

\[<\text{F1WAE}> ::= <\text{num}>\]

| \{+ \ <\text{F1WAE}> \ <\text{F1WAE}>\}\]

| \{- \ <\text{F1WAE}> \ <\text{F1WAE}>\}\]

| \{with \ [<\text{id}> \ <\text{F1WAE}>] \ <\text{F1WAE}>\]\]

| <\text{id}>\]

| \{<\text{id}> \ <\text{F1WAE}>\}\]
FIWAE Datatypes

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

(define-type FIWAE
  [num (n number?)])
  [add (lhs FIWAE?)
    (rhs FIWAE?)]
...
  [app (fun-name symbol?)
    (arg FIWAE?)])
(define (interp a-wae fundefs)
  (type-case FlWAE a-wae
      [num (n) n]
      [add (l r) (+ (interp l fundefs)
                    (interp r fundefs))]
...
  [app (name arg-expr)
       ...)])

(define (interp a-wae fundefs)
  (type-case FlWAE 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)
(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)
(define (interp a-wae fundefs)
  (type-case FlWAE 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))
    ...)
(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) ... ]))
(define (interp a-wae fundefs)
  (type-case FlWAE 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
                    (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-fundef : symbol list-of-FunDef -> FunDef
(define (lookup-fundef name fundefs)
  (cond
   [(empty? fundefs)
    ...]
   [else
    ... (first fundefs)
    ... (lookup-fundef name (rest fundefs))
    ...]]))
; 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)))]))
; subst : F1WAE symbol num -> F1WAE
(define (subst a-f1wae sub-id val)
  (type-case F1WAE a-f1wae
    ...
    [app (fun-name arg)
      (app fun-name (subst arg sub-id val))])))