CFAL = Lazy FAE

<CFAL> ::= <num>
    | {+ <CFAL> <CFAL>}
    | {- <CFAL> <CFAL>}
    | <id>
    | {fun {<id>} <CFAL>}
    | {<CFAL> <CFAL>}

{{fun {x} 0} {+ 1 {fun {y} 2}}}} ⇒ 0
{{fun {x} x} {+ 1 {fun {y} 2}}} ⇒ error
Implementing CFAL

**Option #1:** Run the FAE interpreter in PLAI Lazy!

```scheme
; interp : CFAL DefrdSub -> CFAL-Value
(define (interp 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))))])

arg-val never used ⇒ interp call never evaluated
```
Implementing CFAL

**Option #2:** Use PLAI Scheme and explicitly delay \textit{arg-exp} interpretation

; interp : CFAL DefrdSub \rightarrow CFAL-Value
(define (interp expr ds)
  ...
  [app (fun-exp expr arg-exp expr)
      (local [(define fun-val
               (interp fun-exp expr ds))
               (define arg-val
               (expr arg-exp expr ds))]
      (interp (closure-body fun-val)
              (aSub (closure-param fun-val)
                   arg-val
                   (closure-ds fun-val))))]

where \textit{exprV} is a new kind of CFAL-Value
(define-type CFAL-Value
  [numV (n number?)]
  [closureV (param symbol?)
    (body CFAL?)
    (ds DefrdSub?)])
  [exprV (expr CFAL?)
    (ds DefrdSub?)])
Forcing Evaluation for Number Operations

\[
\text{(interp \{\{\text{fun} \{x\} \{+ 1 x\}\} 10\} (mtSub))}
\]

\[
\Rightarrow \text{error: expected numV, got exprV}
\]

\[
\text{(define \(\text{num-op} \ \text{op} \ \text{op-name} \ x \ y\))}
\]

\[
\text{(define \(\text{num+} \ x \ y\) \(\text{num-op} + ' + x y\))}
\]

\[
\text{(define \(\text{num-} \ x \ y\) \(\text{num-op} - ' - x y\))}
\]

\[
; \text{strict : CFAL-Value -> CFAL-Value}
\]

\[
\text{(define \(\text{strict} \ v\))}
\]

\[
\text{(type-case CFAL-Value v}
\]

\[
[\text{exprV (expr ds) \(\text{strict (interp expr ds)}\)]}
\]

\[
[\text{else v}])}
\]
Forcing Evaluation for Application

\[
\begin{align*}
\text{(interp} & \quad \{\{\text{fun} \{f\} \{f \, 1\}\} \{\text{fun} \{x\} \{+ \, x \, 1\}\}\} \\
& \quad (\text{mtSub}))
\end{align*}
\]

; interp : CFAL DefrdSub → CFAL-Value
(define (interp expr ds)
  ...
  [app (fun-expr arg-expr)
    (local [(define fun-val
      (strict (interp fun-expr ds)))]
      (define arg-val
        (exprV arg-expr ds)]
      (interp (closureV-body fun-val)
        (aSub (closureV-param fun-val)
          arg-val
          (closureV-ds fun-val))))])}
Redundant Evaluation

\{ \{ \text{fun} \ \{x\} \ \{+ \ \{+ \ x \ x\} \ \{+ \ x \ x\}\} \} \n\{− \ \{+ \ 4 \ 5\} \ \{+ \ 8 \ 9\}\}\} \}

How many times is \{+ \ 8 \ 9\} evaluated?

Since the result is always the same, we’d like to evaluate \{− \ \{+ \ 4 \ 5\} \ \{+ \ 8 \ 9\}\} at most once
Caching Strict Results

(define-type CFAL-Value
  [numV (n number?)]
  [closureV (param symbol?)
    (body CFAL?)
    (ds DefrdSub?)]
  [exprV (expr CFAL?)
    (ds DefrdSub?)
    (value (box/c (or/c false CFAL-Value?)))])

; strict : CFAL-Value -> CFAL-Value
(define (strict v)
  (type-case CFAL-Value v
    [exprV (expr ds value-box)
      (if (not (unbox value-box))
        (local [[(define v (strict (interp expr ds)))]
          (begin
            (set-box! value-box v) v))
        (unbox value-box))]
    [else v])))
(define (interp expr ds)
  ...
  [app ...
    (exprV arg-expr ds (box #f))
    ...]])