```
(define (arith-eval expr env)
     (cond
     [ (constant? expr) expr ]
     [ (variable? expr) (lookup expr env) ]
     [ (plus? expr) (+ (arith-eval (op1 expr) env)
     (arith-eval (op2 expr) env)) ]
     [ (mult? expr) (* (arith-eval (op1 expr) env)
     (arith-eval (op2 expr) env)) ]
))
```

#2

1. **(arith-eval (multi-simp E1 E2) env) = (arith-eval (* E1 E2)) env)**
   a. **Assume (and (constant? E1) (constant? E2))**
      i. (mult-simp E1 E2) = (* E1 E2) by definition of mult-simp
      ii. (arith-eval (mult-simp E1 E2) env) = (arith-eval (* E1 E2) env) by Line i
   b. **Assume (equal? E1 0)**
      i. (mult-simp E1 E2) = 0 by definition of mult-simp
      ii. (arith-eval (mult-simp E1 E2) env) = (arith-eval 0 env) = 0 by definition of arith-eval and line i
      iii. (arith-eval (* 0 E2) env) = (* (arith-eval 0 env) (arith-eval E2 env)) = (* 0 (arith-eval E2 env)) = 0 by definition of arith-eval
      iv. (arith-eval (mult-simp E1 E2) env) = (arith-eval (* E1 E2) env) by Lines ii and iii
   c. **Assume (equal? E2 0)**
      i. (mult-simp E1 E2) = 0 by definition of mult-simp
      ii. (arith-eval (mult-simp E1 E2) env) = (arith-eval 0 env) = 0 by definition of arith-eval and line i
      iii. (arith-eval (* E1 0) env) = (* (arith-eval E1 env) (arith-eval 0 env)) = (* (arith-eval E1 env) 0) = 0 by definition of arith-eval
      iv. (arith-eval (mult-simp E1 E2) env) = (arith-eval (* E1 E2) env) by Lines ii and iii
   d. **Assume (equal? E1 1)**
      i. (mult-simp E1 E2) = E2 by definition of mult-simp
      ii. (arith-eval (mult-simp E1 E2) env) = (arith-eval E2 env) by definition of arith-eval and Line i
      iii. (arith-eval (* 1 E2) env) = (* (arith-eval 1 env) (arith-eval E2 env)) = (* 1 (arith-eval E2 env)) = (arith-eval E2 env)
      iv. (arith-eval (mult-simp E1 E2) env) = (arith-eval (* E1 E2) env) by Lines ii and iii
   e. **Assume (equal? E2 1)**
      i. (mult-simp E1 E2) = E1 by definition of mult-simp
      ii. (arith-eval (mult-simp E1 E2) env) = (arith-eval E1 env) by definition of arith-eval and Line i
      iii. (arith-eval (* E1 1) env) = (* (arith-eval E1 env) (arith-eval 1 env)) = (* (arith-eval E2 env) 1) = (arith-eval E1 env)
      iv. (arith-eval (mult-simp E1 E2) env) = (arith-eval (* E1 E2) env) by Lines ii and iii

f. **Assume that Lines 3-7 are not met**
    i. (arith-eval (mult-simp E1 E2) env) = (arith-eval (make-mult E1 E2)) env) = (arith-eval (* E1 E2) env) by definition of make-mult

g. (arith-eval (mult-simp E1 E2) env) = (arith-eval (* E1 E2)) env) has been proven by considering all the different cases of mult-simp

2. (arith-eval (arith-simp expr env) = (arith-eval expr env); expr = (* E1 E2)

a. <u>Base Case: E1,E2 = constant/variable</u>

b. <u>Inductive Hypothesis:</u>
    i. (arith-eval (arith-simp E1) env) = (arith-eval E1 env)
    ii. (arith-eval (arith-simp E2) env) = (arith-eval E2 env)

c. <u>Inductive Proof: Prove (arith-eval (arith-simp (* E1 E2)) env) = (arith-eval (* E1 E2) env)</u>
    i. RHS:
        1. (arith-eval (* E1 E2) env) = (* (arith-eval E1 env) (arith-eval E2 env)) by definition of arith-eval
    ii. LHS:
        1. (arith-eval (arith-simp (* E1 E2)) env) = (arith-eval (mult-simp (arith-simp E1) (arith-simp E2)) env) by definition of arith-simp
        2. (arith-eval (mult-simp (arith-simp E1) (arith-simp E2)) env) = (arith-eval (* (arith-simp E1) (arith-simp E2)) env) by proof for mult-simp in Line 1
        3. (arith-eval (* (arith-simp E1) (arith-simp E2)) env) = (* (arith-eval (arith-simp E1) env) (arith-eval (arith-simp E2) env)) by definition of arith-eval
        4. (* (arith-eval (arith-simp E1) env) (arith-eval (arith-simp E2) env)) = (* (arith-eval E1 env) (arith-eval E2 env)) by Inductive Hypothesis
    iii. (arith-eval (arith-simp (* E1 E2)) env) = (arith-eval (* E1 E2) env) has been proven by Lines i and ii

#3

```
(define (is-simplified? expr)
      (if (constant? expr)
            #t
            (and (noconstant-arith? expr) (nozeros? expr) (nomult1?
            expr))
      )
)
```

3. (is-simplified? (arith-simp expr)) = #t

4. (is-simplified? (arith-simp expr)) = (and (noconstant-arith? (arith-simp expr)) (nozeros? (arith-simp expr)) (nomult1? (arith-simp expr)))

5. (noconstant-arith? (arith-simp expr))

   a. Base Case: expr = constant/variable
      i. (arith-simp expr) = expr
      ii. (noconstant-arith? expr) = #t by definition of nonconstant-arith?
   b. Base Case: expr = (+ E1 E2)
      i. (arith-simp expr) = (plus-simp E1 E2)
      ii. (and (constant? E1) (constant? E2)) = #t
         1. (plus-simp E1 E2) = (+ E1 E2)
         2. (noconstant-arith? (arith-simp expr)) = (noconstant-arith? (+ E1 E2)) = #t by definition of noconstant-arith?
      iii. (and (constant? E1) (constant? E2)) = #f
         1. (noconstant-arith? (+ E1 E2)) = (and (noconstant-arith? E1) (no-constant-arith? E2)) = (and #t #t) = #t
   c. Base Case: expr = (* E1 E2); (and (constant? E1) (constant? E2)) = #f
      i. (and (constant? E1) (constant? E2)) = #t
         1. (plus-simp E1 E2) = (* E1 E2)
         2. (noconstant-arith? (arith-simp expr)) = (noconstant-arith? (* E1 E2)) = #t by definition of noconstant-arith?
      ii. (and (constant? E1) (constant? E2)) = #f
         1. (noconstant-arith? (* E1 E2)) = (and (noconstant-arith? E1) (no-constant-arith? E2)) = (and #t #t) = #t
   d. Inductive Hypothesis:
      i. (and (noconstant-arith? (arith-simp E1)) (noconstant-arith? (arith-simp E2))) = #t
   e. Inductive Proof: (noconstant-arith? (arith-simp (+ E1 E2)))
      i. (noconstant-arith? (arith-simp (+ E1 E2))) = (no-constant-arith? (+ (arith-simp E1) (arith-simp E2))) = (and (no-constant-arith? (arith-simp E1)) (no-constant-arith? (arith-simp E2))) = (and #t #t) = #t by Inductive Hypothesis
   f. Inductive Proof: (noconstant-arith? (arith-simp (* E1 E2)))
      i. (noconstant-arith? (arith-simp (* E1 E2))) = (no-constant-arith? (* (arith-simp E1) (arith-simp E2))) = (and (no-constant-arith? (arith-simp E1)) (no-constant-arith? (arith-simp E2))) = (and #t #t) = #t by Inductive Hypothesis
   g. (noconstant-arith? expr) = #t has been proved by Induction

6. (nozeros? (arith-simp expr))

   a. Base Case: (variable? expr)

i. (arith-simp expr) = expr
ii. (nozeros? expr) = #t by definition of nozeros?

b. Base Case: (and (constant? expr) (not (equal? expr 0)))
   i. (arith-simp expr) = expr
   ii. (nozeros? expr) = #t by definition of nozeros?
c. Inductive Hypothesis
   i. (and (nozeros? (arith-simp E1)) (nozeros? (arith-simp E2)))
d. Inductive Proof (nozeros? (arith-simp (+ E1 E2))) = #t
   i. (no-zeros? (arith-simp (+ E1 E2))) = (no-zeros? (+ (arith-simp E1) (arith-simp E2))) = (and (no-zeros? (arith-simp E1)) (no-zeros? (arith-simp E2))) = (and #t #t) = #t by Inductive Hypothesis
e. Inductive Proof (nozeros? (arith-simp (* E1 E2))) = #t
   i. (no-zeros? (arith-simp (* E1 E2))) = (no-zeros? (* (arith-simp E1) (arith-simp E2))) = (and (no-zeros? (arith-simp E1)) (no-zeros? (arith-simp E2))) = (and #t #t) = #t by Inductive Hypothesis
f. (nozeros? (arith-simp expr)) = #t has been proven by induction

7. **(no-mult1? (arith-simp expr))**
   a. Base Case: (or (constant? expr) (variable? expr))
      i. (arith-simp expr) = expr
      ii. (no-mult1? expr) = #t by definition of nomult1?
   b. Inductive Hypothesis:
      i. (and (no-mult1? E1) (no-mult1? E2)) = #t
   c. Inductive Proof: (no-mult1? (arith-simp (+ E1 E2))) = #t
      i. (no-mult1? (arith-simp (+ E1 E2))) = (no-mult1? (+ (arith-simp E1) (arith-simp E2)) = (and (no-mult1? (arith-simp E1) (arith-simp E2))) = (and #t #t) = #t by Inductive Hypothesis
   d. Inductive Proof: (no-mult1? (arith-simp (* E1 E2))) = #t
      i. (no-mult1? (arith-simp (* E1 E2))) = (no-mult1? (* (arith-simp E1) (arith-simp E2)) = (and (no-mult1? (arith-simp E1) (arith-simp E2))) = (and #t #t) = #t by Inductive Hypothesis
   e. (no-mult1? (arith-simp expr)) = #t has been proven by induction
8. (is-simplified? (arith-simp expr)) = (and (noconstant-arith? (arith-simp expr)) (nozeros? (arith-simp expr)) (nomult1? (arith-simp expr))) = (and #t #t #t) = #t