(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)**
   1. **Assume (and (constant? E1) (constant? E2))**
      1. (mult-simp E1 E2) = (\* E1 E2) by definition of mult-simp
      2. (arith-eval (mult-simp E1 E2) env) = (arith-eval (\* E1 E2) env) by Line i
   2. **Assume (equal? E1 0)**
      1. (mult-simp E1 E2) = 0 by definition of mult-simp
      2. (arith-eval (mult-simp E1 E2) env) = (arith-eval 0 env) = 0 by definition of arith-eval and line i
      3. (arith-eval (\* 0 E2) env) = (\* (arith-eval 0 env) (arith-eval E2 env)) = (\* 0 (arith-eval E2 env)) = 0 by definition of arith-eval
      4. (arith-eval (mult-simp E1 E2) env) = (arith-eval (\* E1 E2) env) by Lines ii and iii
   3. **Assume (equal? E2 0)**
      1. (mult-simp E1 E2) = 0 by definition of mult-simp
      2. (arith-eval (mult-simp E1 E2) env) = (arith-eval 0 env) = 0 by definition of arith-eval and line i
      3. (arith-eval (\* E1 0) env) = (\* (arith-eval E1 env) (arith-eval 0 env)) = (\* (arith-eval E1 env) 0) = 0 by definition of arith-eval
      4. (arith-eval (mult-simp E1 E2) env) = (arith-eval (\* E1 E2) env) by Lines ii and iii
   4. **Assume (equal? E1 1)**
      1. (mult-simp E1 E2) = E2 by definition of mult-simp
      2. (arith-eval (mult-simp E1 E2) env) = (arith-eval E2 env) by definition of arith-eval and Line i
      3. (arith-eval (\* 1 E2) env) = (\* (arith-eval 1 env) (arith-eval E2 env)) = (\* 1 (arith-eval E2 env)) = (arith-eval E2 env)
      4. (arith-eval (mult-simp E1 E2) env) = (arith-eval (\* E1 E2) env) by Lines ii and iii
   5. **Assume (equal? E2 1)**
      1. (mult-simp E1 E2) = E1 by definition of mult-simp
      2. (arith-eval (mult-simp E1 E2) env) = (arith-eval E1 env) by definition of arith-eval and Line i
      3. (arith-eval (\* E1 1) env) = (\* (arith-eval E1 env) (arith-eval 1 env)) = (\* (arith-eval E2 env) 1) = (arith-eval E1 env)
      4. (arith-eval (mult-simp E1 E2) env) = (arith-eval (\* E1 E2) env) by Lines ii and iii
   6. **Assume that Lines 3-7 are not met**
      1. (arith-eval (mult-simp E1 E2) env) = (arith-eval (make-mult E1 E2)) env) = (arith-eval (\* E1 E2) env) by definition of make-mult
   7. (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)**
   1. Base Case: E1,E2 = constant/variable
   2. Inductive Hypothesis:
      1. (arith-eval (arith-simp E1) env) = (arith-eval E1 env)
      2. (arith-eval (arith-simp E2) env) = (arith-eval E2 env)
   3. Inductive Proof: Prove (arith-eval (arith-simp (\* E1 E2)) env) = (arith-eval (\* E1 E2) env)
      1. RHS:
         1. (arith-eval (\* E1 E2) env) = (\* (arith-eval E1 env) (arith-eval E2 env)) by definition of arith-eval
      2. 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
      3. (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))

)

)

1. **(is-simplified? (arith-simp expr)) = #t**
2. (is-simplified? (arith-simp expr)) = (and (noconstant-arith? (arith-simp expr)) (nozeros? (arith-simp expr)) (nomult1? (arith-simp expr)))
3. **(noconstant-arith? (arith-simp expr))**
   1. Base Case: expr = constant/variable
      1. (arith-simp expr) = expr
      2. (noconstant-arith? expr) = #t by definition of nonconstant-arith?
   2. Base Case: expr = (+ E1 E2)
      1. (arith-simp expr) = (plus-simp E1 E2)
      2. (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?
      3. (and (constant? E1) (constant? E2)) = #f
         1. (noconstant-arith? (+ E1 E2)) = (and (noconstant-arith? E1) (no-constant-arith? E2)) = (and #t #t) = #t
   3. Base Case: expr = (\* E1 E2)
      1. (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?
      2. (and (constant? E1) (constant? E2)) = #f
         1. (noconstant-arith? (\* E1 E2)) = (and (noconstant-arith? E1) (no-constant-arith? E2)) = (and #t #t) = #t
   4. Inductive Hypothesis:
      1. (and (noconstant-arith? (arith-simp E1)) (noconstant-arith? (arith-simp E2))) = #t
   5. Inductive Proof: (noconstant-arith? (arith-simp (+ E1 E2)))
      1. (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
   6. Inductive Proof: (noconstant-arith? (arith-simp (\* E1 E2)))
      1. (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
   7. (noconstant-arith? expr) = #t has been proved by Induction
4. **(nozeros? (arith-simp expr))**
   1. Base Case: (variable? expr)
      1. (arith-simp expr) = expr
      2. (nozeros? expr) = #t by definition of nozeros?
   2. Base Case: (and (constant? expr) (not (equal? expr 0)))
      1. (arith-simp expr) = expr
      2. (nozeros? expr) = #t by definition of nozeros?
   3. Inductive Hypothesis
      1. (and (nozeros? (arith-simp E1)) (nozeros? (arith-simp E2)))
   4. Inductive Proof (nozeros? (arith-simp (+ E1 E2))) = #t
      1. (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
   5. Inductive Proof (nozeros? (arith-simp (\* E1 E2))) = #t
      1. (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
   6. (nozeros? (arith-simp expr)) = #t has been proven by induction
5. **(no-mult1? (arith-simp expr))**
   1. Base Case: (or (constant? expr) (variable? expr))
      1. (arith-simp expr) = expr
      2. (no-mult1? expr) = #t by definition of nomult1?
   2. Inductive Hypothesis:
      1. (and (no-mult1? E1) (no-mult1? E2)) = #t
   3. Inductive Proof: (no-mult1? (arith-simp (+ E1 E2))) = #t
      1. (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
   4. Inductive Proof: (no-mult1? (arith-simp (\* E1 E2))) = #t
      1. (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
   5. (no-mult1? (arith-simp expr)) = #t has been proven by induction
6. (is-simplified? (arith-simp expr)) = (and (noconstant-arith? (arith-simp expr)) (nozeros? (arith-simp expr)) (nomult1? (arith-simp expr))) = (and #t #t #t) = #t