Eric Wan – [ezw23@drexel.edu](mailto:ezw23@drexel.edu)

(define (is-simplified? expr)

(if (constant? expr)

#t

(and (noconstant-arith? expr) (nozeros? expr) (nomult1? expr))

)

)

(plus-simp 2 5) = (+ 2 5) = 7

(eval (plus-simp a b) env) = (eval (+ a b) env)

case 1:

let c1 be a constant int

let c2 be a constant int

let (+ c1 c2) = c3

(eval (plus-simp c1 c2) env)

by def:

= (eval (+ c1 c2) env)

= c3

(eval (+ c1 c2) env)

= c3

let c2 be any value

then (eval c2 env) = c2

and (+ c2 0) = c2

(eval (plus-simp 0 c2) env)

= (eval c2 env)

= c2

(eval (+ 0 c2) env)

= c2

Prove that (is-simplified? (arith-simp expr)) = #t

1. Prove k=0, 1 case is true

assume expr is constant:

(arith-simp expr)

= (arith-simp constant)

= (constant? constant)

= constant

(is-simplified? constant)

= (constant? constant) = #t

assume expr is variable:

(arith-simp expr)

= (arith-simp variable)

= (variable? variable)

= variable

(is-simplified? variable)

= (constant? variable) = #f

= (and (noconstant-arith? expr) (nozeros? expr) (nomult1? expr))))

= (and (variable? expr) (variable? expr) (variable? expr))

= (and #t #t #t) = #t

2. Assume k=n case is true

expr is (+ E1 E2):

= (is-simplified? expr)

= (constant? expr) = #f

= (and (noconstant-arith? expr) (nozeros? expr) (nomult1? expr))

= (and (plus? expr) (nozeros? expr) (nomult1? expr))

= (and (and (constant? (op1 expr)) (constant? (op2 expr))) … )

= (and (and (constant? E1) (constant? E2)) … )  
= (and (and #t #t) … )

= (and (#t) (nozeros? expr) (nomult1? expr))

= (and #f (plus? expr) (plus? expr))

= (and #f (and (nozeros? (op1 expr)) (nozeros? (op2 expr))…)

= (and #f (and (nozeros? E1) (nozeros? E2)) (plus? expr))

= (and #f (and #t #t) (and (nomult1? (op1 expr)) (nomult1? (op2 expr))))

= (and #f #t (and (nonmult1? E1) (nonmult1? E2)))

= (and #f #t (and #t #t))

= (and #f #t #t) = #f

expr is (\* E1 E2):

= (is-simplified? expr)

= (constant? expr) = #f

= (and (noconstant-arith? expr) (nozeros? expr) (nomult1? expr))

= (and (plus? expr) (nozeros? expr) (nomult1? expr))

= (and (and (constant? (op1 expr)) (constant? (op2 expr))) … )

= (and (and (constant? E1) (constant? E2)) … )  
= (and (and #t #t) … )

= (and (#t) (nozeros? expr) (nomult1? expr))

= (and #f (plus? expr) (plus? expr))

= (and #f (and (nozeros? (op1 expr)) (nozeros? (op2 expr))…)

= (and #f (and (nozeros? E1) (nozeros? E2)) (plus? expr))

= (and #f (and #t #t) (and (nomult1? (op1 expr)) (nomult1? (op2 expr))))

= (and #f #t (and (nonmult1? E1) (nonmult1? E2)))

= (and #f #t (and #t #t))

= (and #f #t #t) = #f

3. Prove k=n+1 case is true

expr is (+ (+ E1 E2) E3):

= (is-simplified? expr)

= (constant? expr) = #f

= (and (noconstant-arith? expr) (nozeros? expr) (nomult1? expr))

= (and (plus? expr) (nozeros? expr) (nomult1? expr))

= (and (and (constant? (op1 expr)) (constant? (op2 expr))) … )

= (constant? (+ E1 E2) => #f

= (and (and (and (noconstant-arith? (op1 expr)) (noconstant-arith? (op2 expr))…)

= (and (noconstant-arith? E1) (noconstant-arith? E2)…) => #f

= (and #f (nozeros? expr) (nomult1? expr))

= (plus? expr)

= (and (nozeros? (op1 expr)) (nozernos? (op2 expr)))

= (and (and (nozeros? E1) (nozeros? E2)) (nozeros? E3)) => #t

= (and #f #t (nomult1? expr))

= (nomult1? expr)

= (plus? expr)

= (and (nomult1? (op1 expr)) (nomult1? (op2 expr)))

= (and (plus? (+ E1 E2)) (nomult1? (op2 expr)))

= (and ((nomult1? E1) (nomult1? E2)) (nomult1? E3)) => #t

= (and #f #f #t) => #f

expr is (\* (+ E1 E2):

= (is-simplified? expr)

= (constant? expr) = #f

= (and (noconstant-arith? expr) (nozeros? expr) (nomult1? expr))

= (and (mult? expr) …)

= (and (and (constant? (op1 expr)) (constant? (op2 expr))) …)

= (and (and (constant? (+ E1 E2)) (constant? E3)))

following similar steps from earlier

= (and #f (nozeros? expr) (nomult1? expr))

= (and #f (mult? expr) … )

= (and #f (and (nozeros? (op1 expr)) (nozeros? (op2 expr))) … )

= (and #f (and (plus? (+ E1 E2)) (nozeros? E3)) … )

= (and #f (and (and (nozeros? E1) (nozeros E2)) (nozeros? E3)) … )

= (and #f #t (nomult1? expr))

= (and #f #t (mult? expr))

= (mult? expr)

= (or (equal? (op1 expr) 1) (equal? (op2 expr) 1))

= (or (equal? (+ E1 E2) 1) (equal? E3))

= (or #f #f) => #f

= (and (nomult1? (op1 expr)) (nomult1? (op2 expr))

= (and (plus? (+ E1 E2)) (nomult1? E3))

= (and (and (nomult1? E1) (nomult1? E2)) (nomult1? E3))

= (and (and #t #t) #t) => #t

= (and #f #t #t) => #f