Gmail **Shubham thorat <sthorat661@gmail.com>**

(no subject)

Shubham thorat <sthorat661@gmail.com> 14 March 2020 at 03:14
To: shubham.thorat17@vit.edu

```
Commutativity

(define-ruleset commutativity (arithmetic simplify fp-safe)

#:type ([a real] [b real])

a + b = b + a

a*b = b*a



c + a + b = c + b + a

c*a*b = c*b*a

; Associativity



(a + (b + c)) = ((a + b) + c)


(a + (b – c)) = ((a + b) - c)

((a – b) + c) = (a - ( b - c))

(a - ( b + c)) = ( (a - b) - c)

(( a + b) - c) ( a + (b - c))

( (a - b) - c) = (a - (b + c))
( a - (b - c)) ( (a - b) + c)

(a * (b * c)) = ( (a * b) * c)

( ( a * b) * c) = (a * (b * c))

(a * (b / c)) = ( (a * b) / c)]

((a / b) * c) = ((a * c) / b)]

(/ a (* b c)) (/ (/ a b) c)]

(/ (* b c) a) (/ b (/ a c))]

(/ a (/ b c)) (* (/ a b) c)]

(/ (/ b c) a) (/ b (* a c))]

(- a b) (+ a (- b))]

(+ a (- b)) (- a b)])



[associate-+r+.c (+.c a (+.c b c)) (+.c (+.c a b) c)]

[associate-+l+.c (+.c (+.c a b) c) (+.c a (+.c b c))]

[associate-+r-.c (+.c a (-.c b c)) (-.c (+.c a b) c)]

[associate-+l-.c (+.c (-.c a b) c) (-.c a (-.c b c))]

[associate--r+.c (-.c a (+.c b c)) (-.c (-.c a b) c)]

[associate--l+.c (-.c (+.c a b) c) (+.c a (-.c b c))]

[associate--l-.c (-.c (-.c a b) c) (-.c a (+.c b c))]

[associate--r-.c (-.c a (-.c b c)) (+.c (-.c a b) c)]

[associate-*r*.c (*.c a (*.c b c)) (*.c (*.c a b) c)]

[associate-*l*.c (*.c (*.c a b) c) (*.c a (*.c b c))]

[associate-*r/.c (*.c a (/.c b c)) (/.c (*.c a b) c)]

[associate-*l/.c (*.c (/.c a b) c) (/.c (*.c a c) b)]

[associate-/r*.c (/.c a (*.c b c)) (/.c (/.c a b) c)]
```

```
[associate-/l*.c (/.c (*.c b c) a) (/.c b (/.c a c))]
[associate-/r/.c (/.c a (/.c b c)) (*.c (/.c a b) c)]
[associate-/l/.c (/.c (/.c b c) a) (/.c b (*.c a c))]
[sub-neg.c (-.c a b) (+.c a (neg.c b))]
[unsub-neg.c (+.c a (neg.c b)) (-.c a b)])
; Counting
(define-ruleset counting (arithmetic simplify)
#:type ([x real])
[count-2 (+ x x) (* 2 x)])
; Distributivity
(define-ruleset distributivity (arithmetic simplify)
#:type ([a real] [b real] [c real])
[distribute-lft-in (* a (+ b c)) (+ (* a b) (* a c))]
[distribute-rgt-in (* a (+ b c)) (+ (* b a) (* c a))]
[distribute-lft-out (+ (* a b) (* a c)) (* a (+ b c))]
[distribute-lft-out-- (- (* a b) (* a c)) (* a (- b c))]
[distribute-rgt-out (+ (* b a) (* c a)) (* a (+ b c))]
[distribute-rgt-out-- (- (* b a) (* c a)) (* a (- b c))]
[distribute-lft1-in (+ (* b a) a) (* (+ b 1) a)]
[distribute-rgt1-in (+ a (* c a)) (* (+ c 1) a)])
(define-ruleset distributivity.c (arithmetic simplify complex)
#:type ([a complex] [b complex] [c complex])
[distribute-lft-in.c (*.c a (+.c b c)) (+.c (*.c a b) (*.c a c))]
[distribute-rgt-in.c (*.c a (+.c b c)) (+.c (*.c b a) (*.c c a))]
[distribute-lft-out.c (+.c (*.c a b) (*.c a c)) (*.c a (+.c b c))]
[distribute-lft-out--.c (-.c (*.c a b) (*.c a c)) (*.c a (-.c b c))]
```

```
[distribute-rgt-out.c (+.c (*.c b a) (*.c c a)) (*.c a (+.c b c))]

[distribute-rgt-out--.c (-.c (*.c b a) (*.c c a)) (*.c a (-.c b c))]

[distribute-lft1-in.c (+.c (*.c b a) a) (*.c (+.c b (complex 1 0)) a)]

[distribute-rgt1-in.c (+.c a (*.c c a)) (*.c (+.c c (complex 1 0)) a)])

; Safe Distributiviity

(define-ruleset distributivity-fp-safe (arithmetic simplify fp-safe)

#:type ([a real] [b real])

[distribute-lft-neg-in (- (* a b)) (* (- a) b)]

[distribute-rgt-neg-in (- (* a b)) (* a (- b))]

[distribute-lft-neg-out (* (- a) b) (- (* a b))]

[distribute-rgt-neg-out (* a (- b)) (- (* a b))]

[distribute-neg-in (- (+ a b)) (+ (- a) (- b))]

[distribute-neg-out (+ (- a) (- b)) (- (+ a b))]

[distribute-frac-neg (/ (- a) b) (- (/ a b))]

[distribute-neg-frac (- (/ a b)) (/ (- a) b)])

; Difference of squares

(define-ruleset difference-of-squares-canonicalize (polynomials simplify)

#:type ([a real] [b real])

[swap-sqr (* (* a b) (* a b)) (* (* a a) (* b b))]

[unswap-sqr (* (* a a) (* b b)) (* (* a b) (* a b))]

[difference-of-squares (- (* a a) (* b b)) (* (+ a b) (- a b))]

[difference-of-sqr-1 (- (* a a) 1) (* (+ a 1) (- a 1))]

[difference-of-sqr--1 (+ (* a a) -1) (* (+ a 1) (- a 1))]

[sqr-pow (pow a b) (* (pow a (/ b 2)) (pow a (/ b 2)))]

[pow-sqr (* (pow a b) (pow a b)) (pow a (* 2 b))]

)
```

```
(define-ruleset difference-of-squares-flip (polynomials)
#:type ([a real] [b real])
[flip-+ (a + b) = (((a * a) - (b * b)) / (a - b))]
[flip-- (- a b) (/ (- (* a a) (* b b)) (+ a b))])
; Identity
(define-ruleset id-reduce (arithmetic simplify)
#:type ([a real])
[remove-double-div (/ 1 (/ 1 a)) a]
[rgt-mult-inverse (* a (/ 1 a)) 1]
[lft-mult-inverse (* (/ 1 a) a) 1])
(define-ruleset id-reduce-fp-safe-nan (arithmetic simplify fp-safe-nan)
#:type ([a real])
[+-inverses (- a a) 0]
[*-inverses (/ a a) 1]
[div0 (/ 0 a) 0]
[mul0 (* 0 a) 0]
[mul0 (* a 0) 0])
(define-ruleset id-reduce-fp-safe (arithmetic simplify fp-safe)
#:type ([a real])
[+-lft-identity (+ 0 a) a]
[+-rgt-identity (+ a 0) a]
[--rgt-identity (- a 0) a]
[sub0-neg (- 0 a) (- a)]
[remove-double-neg (- (- a)) a]
[*-lft-identity (* 1 a) a]
[*-rgt-identity (* a 1) a]
[/-rgt-identity (/ a 1) a]

[mul-1-neg (* -1 a) (- a)])

(define-ruleset id-transform (arithmetic)

#:type ([a real] [b real])

[div-inv (/ a b) (* a (/ 1 b))]

[un-div-inv (* a (/ 1 b)) (/ a b)]

[clear-num (/ a b) (/ 1 (/ b a))])

(define-ruleset id-transform-fp-safe (arithmetic fp-safe)

#:type ([a real] [b real])

[sub-neg (- a b) (+ a (- b))]

[unsub-neg (+ a (- b)) (- a b)]

[neg-sub0 (- b) (- 0 b)]

[*-un-lft-identity a (* 1 a)]

[neg-mul-1 (- a) (* -1 a)])

; Difference of cubes

(define-ruleset difference-of-cubes (polynomials)

#:type ([a real] [b real])

[sum-cubes (+ (pow a 3) (pow b 3))

(* (+ (* a a) (- (* b b) (* a b))) (+ a b))]

[difference-cubes (- (pow a 3) (pow b 3))

(* (+ (* a a) (+ (* b b) (* a b))) (- a b))]

[flip3-+ (+ a b)

(/ (+ (pow a 3) (pow b 3)) (+ (* a a) (- (* b b) (* a b))))]

[flip3-- (- a b)

(/ (- (pow a 3) (pow b 3)) (+ (* a a) (+ (* b b) (* a b))))])

; Dealing with fractions
```

```
(define-ruleset fractions-distribute (fractions simplify)
#:type ([a real] [b real] [c real] [d real])
[div-sub (/ (- a b) c) (- (/ a c) (/ b c))]
[times-frac (/ (* a b) (* c d)) (* (/ a c) (/ b d))]])
(define-ruleset fractions-distribute.c (fractions simplify complex)
#:type ([a complex] [b complex] [c complex] [d complex])
[div-sub.c (/.c (-.c a b) c) (-.c (/.c a c) (/.c b c))]
[times-frac.c (/.c (*.c a b) (*.c c d)) (*.c (/.c a c) (/.c b d))]])
(define-ruleset fractions-transform (fractions)
#:type ([a real] [b real] [c real] [d real])
[sub-div (- (/ a c) (/ b c)) (/ (- a b) c)]
[frac-add (+ (/ a b) (/ c d)) (/ (+ (* a d) (* b c)) (* b d))]
[frac-sub (- (/ a b) (/ c d)) (/ (- (* a d) (* b c)) (* b d))]
[frac-times (* (/ a b) (/ c d)) (/ (* a c) (* b d))]
[frac-2neg (/ a b) (/ (- a) (- b))]])
(define-ruleset fractions-transform.c (fractions complex)
#:type ([a complex] [b complex] [c complex] [d complex])
[sub-div.c (-.c (/.c a c) (/.c b c)) (/.c (-.c a b) c)]
[frac-add.c (+.c (/.c a b) (/.c c d)) (/.c (+.c (*.c a d) (*.c b c)) (*.c b d))]
[frac-sub.c (-.c (/.c a b) (/.c c d)) (/.c (-.c (*.c a d) (*.c b c)) (*.c b d))]
[frac-times.c (*.c (/.c a b) (/.c c d)) (/.c (*.c a c) (*.c b d))]
[frac-2neg.c (/.c a b) (/.c (neg.c a) (neg.c b))]])
; Square root
(define-ruleset squares-reduce (arithmetic simplify)
#:type ([x real])
[rem-square-sqrt (* (sqrt x) (sqrt x)) x]
```

```
[rem-sqrt-square (sqrt (* x x)) (fabs x)])

(define-ruleset squares-reduce-fp-sound (arithmetic simplify fp-sound)

#:type ([x real])

[sqr-neg (* (- x) (- x)) (* x x)])

(define-ruleset squares-transform (arithmetic)

#:type ([x real] [y real])

[sqrt-prod (sqrt (* x y)) (* (sqrt x) (sqrt y))]

[sqrt-div (sqrt (/ x y)) (/ (sqrt x) (sqrt y))]

[sqrt-pow1 (sqrt (pow x y)) (pow x (/ y 2))]

[sqrt-pow2 (pow (sqrt x) y) (pow x (/ y 2))]

[sqrt-unprod (* (sqrt x) (sqrt y)) (sqrt (* x y))]

[sqrt-undiv (/ (sqrt x) (sqrt y)) (sqrt (/ x y))]

[add-sqr-sqrt x (* (sqrt x) (sqrt x))])

; Cube root

(define-ruleset cubes-reduce (arithmetic simplify)

#:type ([x real])

[rem-cube-cbrt (pow (cbrt x) 3) x]

[rem-cbrt-cube (cbrt (pow x 3)) x]

[cube-neg (pow (- x) 3) (- (pow x 3))])

(define-ruleset cubes-distribute (arithmetic simplify)

#:type ([x real] [y real])

[cube-prod (pow (* x y) 3) (* (pow x 3) (pow y 3))]

[cube-div (pow (/ x y) 3) (/ (pow x 3) (pow y 3))]

[cube-mult (pow x 3) (* x (* x x))])

(define-ruleset cubes-transform (arithmetic)

#:type ([x real] [y real])
```

```
[cbrt-prod (cbrt (* x y)) (* (cbrt x) (cbrt y))]

[cbrt-div (cbrt (/ x y)) (/ (cbrt x) (cbrt y))]

[cbrt-unprod (* (cbrt x) (cbrt y)) (cbrt (* x y))]

[cbrt-undiv (/ (cbrt x) (cbrt y)) (cbrt (/ x y))]

[add-cube-cbrt x (* (* (cbrt x) (cbrt x)) (cbrt x))]

[add-cbrt-cube x (cbrt (* (* x x) x))]])

(define-ruleset cubes-canonicalize (arithmetic simplify)

#:type ([x real])

[cube-unmult (* x (* x x)) (pow x 3)]])

; Exponentials

(define-ruleset exp-expand (exponents)

#:type ([x real])

[add-exp-log x (exp (log x))]

[add-log-exp x (log (exp x))]])

(define-ruleset exp-reduce (exponents simplify)

#:type ([x real])

[rem-exp-log (exp (log x)) x]

[rem-log-exp (log (exp x)) x])

(define-ruleset exp-constants (exponents simplify fp-safe)

[exp-0 (exp 0) 1]

[exp-1-e (exp 1) E]

[1-exp 1 (exp 0)]

[e-exp-1 E (exp 1)]])

(define-ruleset exp-distribute (exponents simplify)

#:type ([a real] [b real])

[exp-sum (exp (+ a b)) (* (exp a) (exp b))]
```

```
[exp-neg (exp (- a)) (/ 1 (exp a))]
[exp-diff (exp (- a b)) (/ (exp a) (exp b))]])
(define-ruleset exp-factor (exponents simplify)
#:type ([a real] [b real])
[prod-exp (* (exp a) (exp b)) (exp (+ a b))]
[rec-exp (/ 1 (exp a)) (exp (- a))]
[div-exp (/ (exp a) (exp b)) (exp (- a b))]
[exp-prod (exp (* a b)) (pow (exp a) b)]
[exp-sqrt (exp (/ a 2)) (sqrt (exp a))]
[exp-cbrt (exp (/ a 3)) (cbrt (exp a))]
[exp-lft-sqr (exp (* a 2)) (* (exp a) (exp a))]
[exp-lft-cube (exp (* a 3)) (pow (exp a) 3)]])
; Powers
(define-ruleset pow-reduce (exponents simplify)
#:type ([a real])
[unpow-1 (pow a -1) (/ 1 a)])
(define-ruleset pow-reduce-fp-safe (exponents simplify fp-safe)
#:type ([a real])
[unpow1 (pow a 1) a])
(define-ruleset pow-reduce-fp-safe-nan (exponents simplify fp-safe-nan)
#:type ([a real])
[unpow0 (pow a 0) 1]
[pow-base-1 (pow 1 a) 1])
(define-ruleset pow-expand-fp-safe (exponents fp-safe)
#:type ([a real])
[pow1 a (pow a 1)])
```

```
(define-ruleset pow-canonicalize (exponents simplify)
#:type ([a real] [b real])
[exp-to-pow (exp (* (log a) b)) (pow a b)]
[pow-plus (* (pow a b) a) (pow a (+ b 1))]
[unpow1/2 (pow a 1/2) (sqrt a)]
[unpow2 (pow a 2) (* a a)]
[unpow3 (pow a 3) (* (* a a) a)]
[unpow1/3 (pow a 1/3) (cbrt a)])
(define-ruleset pow-transform (exponents)
#:type ([a real] [b real] [c real])
[pow-exp (pow (exp a) b) (exp (* a b))]
[pow-to-exp (pow a b) (exp (* (log a) b))]
[pow-prod-up (* (pow a b) (pow a c)) (pow a (+ b c))]
[pow-prod-down (* (pow b a) (pow c a)) (pow (* b c) a)]
[pow-pow (pow (pow a b) c) (pow a (* b c))]
[pow-neg (pow a (- b)) (/ 1 (pow a b))]
[pow-flip (/ 1 (pow a b)) (pow a (- b))]
[pow-div (/ (pow a b) (pow a c)) (pow a (- b c))]
[pow-sub (pow a (- b c)) (/ (pow a b) (pow a c))]
[pow-unpow (pow a (* b c)) (pow (pow a b) c)]
[unpow-prod-up (pow a (+ b c)) (* (pow a b) (pow a c))]
[unpow-prod-down (pow (* b c) a) (* (pow b a) (pow c a))]
[pow1/2 (sqrt a) (pow a 1/2)]
[pow2 (* a a) (pow a 2)]
[pow1/3 (cbrt a) (pow a 1/3)]
[pow3 (* (* a a) a) (pow a 3)])
```

```
(define-ruleset pow-transform-fp-safe-nan (exponents fp-safe-nan)

#:type ([a real])

[pow-base-0 (pow 0 a) 0])

(define-ruleset pow-transform-fp-safe (exponents fp-safe)

#:type ([a real])

[inv-pow (/ 1 a) (pow a -1)])

; Logarithms

(define-ruleset log-distribute (exponents simplify)

#:type ([a real] [b real])

[log-prod (log (* a b)) (+ (log a) (log b))]

[log-div (log (/ a b)) (- (log a) (log b))]

[log-rec (log (/ 1 a)) (- (log a))]

[log-pow (log (pow a b)) (* b (log a))])

(define-ruleset log-distribute-fp-safe (exponents simplify fp-safe)

[log-E (log E) 1])

(define-ruleset log-factor (exponents)

#:type ([a real] [b real])

[sum-log (+ (log a) (log b)) (log (* a b))]

[diff-log (- (log a) (log b)) (log (/ a b))]

[neg-log (- (log a)) (log (/ 1 a))])

; Trigonometry

(define-ruleset trig-reduce (trigonometry simplify)

#:type ([a real] [b real] [x real])

[cos-sin-sum (+ (* (cos a) (cos a)) (* (sin a) (sin a))) 1]

[1-sub-cos (- 1 (* (cos a) (cos a))) (* (sin a) (sin a))]

[1-sub-sin (- 1 (* (sin a) (sin a))) (* (cos a) (cos a))]
```

```
[-1-add-cos (+ (* (cos a) (cos a)) -1) (- (* (sin a) (sin a)))]

[-1-add-sin (+ (* (sin a) (sin a)) -1) (- (* (cos a) (cos a)))]

[sub-1-cos (- (* (cos a) (cos a)) 1) (- (* (sin a) (sin a)))]

[sub-1-sin (- (* (sin a) (sin a)) 1) (- (* (cos a) (cos a)))]

[sin-PI/6 (sin (/ PI 6)) 1/2]

[sin-PI/4 (sin (/ PI 4)) (/ (sqrt 2) 2)]

[sin-PI/3 (sin (/ PI 3)) (/ (sqrt 3) 2)]

[sin-PI/2 (sin (/ PI 2)) 1]

[sin-PI (sin PI) 0]

[sin-+PI (sin (+ x PI)) (- (sin x))]

[sin-+PI/2 (sin (+ x (/ PI 2))) (cos x)]

[cos-PI/6 (cos (/ PI 6)) (/ (sqrt 3) 2)]

[cos-PI/4 (cos (/ PI 4)) (/ (sqrt 2) 2)]

[cos-PI/3 (cos (/ PI 3)) 1/2]

[cos-PI/2 (cos (/ PI 2)) 0]

[cos-PI (cos PI) -1]

[cos-+PI (cos (+ x PI)) (- (cos x))]

[cos-+PI/2 (cos (+ x (/ PI 2))) (- (sin x))]

[tan-PI/6 (tan (/ PI 6)) (/ 1 (sqrt 3))]

[tan-PI/4 (tan (/ PI 4)) 1]

[tan-PI/3 (tan (/ PI 3)) (sqrt 3)]

[tan-PI (tan PI) 0]

[tan-+PI (tan (+ x PI)) (tan x)]

[tan-+PI/2 (tan (+ x (/ PI 2))) (- (/ 1 (tan x)))]

[hang-0p-tan (/ (sin a) (+ 1 (cos a))) (tan (/ a 2))]

[hang-0m-tan (/ (- (sin a)) (+ 1 (cos a))) (tan (/ (- a) 2))]
```

```
[hang-p0-tan (/ (- 1 (cos a)) (sin a)) (tan (/ a 2))]
[hang-m0-tan (/ (- 1 (cos a)) (- (sin a))) (tan (/ (- a) 2))]
[hang-p-tan (/ (+ (sin a) (sin b)) (+ (cos a) (cos b)))
(tan (/ (+ a b) 2))]
[hang-m-tan (/ (- (sin a) (sin b)) (+ (cos a) (cos b)))
(tan (/ (- a b) 2))])
(define-ruleset trig-reduce-fp-sound (trigonometry simplify fp-safe)
[sin-0 (sin 0) 0]
[cos-0 (cos 0) 1]
[tan-0 (tan 0) 0])
(define-ruleset trig-reduce-fp-sound-nan (trigonometry simplify fp-safe-nan)
#:type ([x real])
[sin-neg (sin (- x)) (- (sin x))]
[cos-neg (cos (- x)) (cos x)]
[tan-neg (tan (- x)) (- (tan x))])
(define-ruleset trig-expand (trigonometry)
#:type ([x real] [y real] [a real] [b real])
[sin-sum (sin (+ x y)) (+ (* (sin x) (cos y)) (* (cos x) (sin y)))]
[cos-sum (cos (+ x y)) (- (* (cos x) (cos y)) (* (sin x) (sin y)))]
[tan-sum (tan (+ x y)) (/ (+ (tan x) (tan y)) (- 1 (* (tan x) (tan y))))]
[sin-diff (sin (- x y)) (- (* (sin x) (cos y)) (* (cos x) (sin y)))]
[cos-diff (cos (- x y)) (+ (* (cos x) (cos y)) (* (sin x) (sin y)))]
[sin-2 (sin (* 2 x))
(* 2 (* (sin x) (cos x)))]
[sin-3 (sin (* 3 x))
(- (* 3 (sin x)) (* 4 (pow (sin x) 3)))]
```

```
[2-sin (* 2 (* (sin x) (cos x)))

(sin (* 2 x))]

[3-sin (- (* 3 (sin x)) (* 4 (pow (sin x) 3)))

(sin (* 3 x))]

[cos-2 (cos (* 2 x))

(- (* (cos x) (cos x)) (* (sin x) (sin x)))]

[cos-3 (cos (* 3 x))

(- (* 4 (pow (cos x) 3)) (* 3 (cos x)))]

[2-cos (- (* (cos x) (cos x)) (* (sin x) (sin x)))

(cos (* 2 x))]

[3-cos (- (* 4 (pow (cos x) 3)) (* 3 (cos x)))

(cos (* 3 x))]

[tan-2 (tan (* 2 x)) (/ (* 2 (tan x)) (- 1 (* (tan x) (tan x))))]

[2-tan (/ (* 2 (tan x)) (- 1 (* (tan x) (tan x)))) (tan (* 2 x))]

[sqr-sin (* (sin x) (sin x)) (- 1/2 (* 1/2 (cos (* 2 x))))]

[sqr-cos (* (cos x) (cos x)) (+ 1/2 (* 1/2 (cos (* 2 x))))]

[diff-sin (- (sin x) (sin y)) (* 2 (* (sin (/ (- x y) 2)) (cos (/ (+ x y) 2))))]

[diff-cos (- (cos x) (cos y)) (* -2 (* (sin (/ (- x y) 2)) (sin (/ (+ x y) 2))))]

[sum-sin (+ (sin x) (sin y)) (* 2 (* (sin (/ (+ x y) 2)) (cos (/ (- x y) 2))))]

[sum-cos (+ (cos x) (cos y)) (* 2 (* (cos (/ (+ x y) 2)) (cos (/ (- x y) 2))))]

[cos-mult (* (cos x) (cos y)) (/ (+ (cos (+ x y)) (cos (- x y))) 2)]

[sin-mult (* (sin x) (sin y)) (/ (- (cos (- x y)) (cos (+ x y))) 2)]

[sin-cos-mult (* (sin x) (cos y)) (/ (+ (sin (- x y)) (sin (+ x y))) 2)]

[diff-atan (- (atan x) (atan y)) (atan2 (- x y) (+ 1 (* x y)))]

[sum-atan (+ (atan x) (atan y)) (atan2 (+ x y) (- 1 (* x y)))]

[tan-quot (tan x) (/ (sin x) (cos x))]
```

```
[quot-tan (/ (sin x) (cos x)) (tan x)]
[tan-hang-p (tan (/ (+ a b) 2))
(/ (+ (sin a) (sin b)) (+ (cos a) (cos b)))]
[tan-hang-m (tan (/ (- a b) 2))
(/ (- (sin a) (sin b)) (+ (cos a) (cos b)))])
(define-ruleset trig-expand-fp-safe (trignometry fp-safe)
#:type ([x real])
[sqr-sin (* (sin x) (sin x)) (- 1 (* (cos x) (cos x)))]
[sqr-cos (* (cos x) (cos x)) (- 1 (* (sin x) (sin x)))])
(define-ruleset trig-inverses (trigonometry)
#:type ([x real])
[sin-asin (sin (asin x)) x]
[cos-acos (cos (acos x)) x]
[tan-atan (tan (atan x)) x]
[atan-tan (atan (tan x)) (remainder x PI)]
[asin-sin (asin (sin x)) (- (fabs (remainder (+ x (/ PI 2)) (* 2 PI))) (/ PI 2))]
[acos-cos (acos (cos x)) (fabs (remainder x (* 2 PI)))])
(define-ruleset trig-inverses-simplified (trigonometry)
#:type ([x real])
[atan-tan-s (atan (tan x)) x]
[asin-sin-s (asin (sin x)) x]
[acos-cos-s (acos (cos x)) x])
(define-ruleset atrig-expand (trigonometry)
#:type ([x real])
[cos-asin (cos (asin x)) (sqrt (- 1 (* x x)))]
[tan-asin (tan (asin x)) (/ x (sqrt (- 1 (* x x))))]
```

```
[sin-acos (sin (acos x)) (sqrt (- 1 (* x x)))]

[tan-acos (tan (acos x)) (/ (sqrt (- 1 (* x x))) x)]

[sin-atan (sin (atan x)) (/ x (sqrt (+ 1 (* x x))))]

[cos-atan (cos (atan x)) (/ 1 (sqrt (+ 1 (* x x))))]

[asin-acos (asin x) (- (/ PI 2) (acos x))]

[acos-asin (acos x) (- (/ PI 2) (asin x))]

[asin-neg (asin (- x)) (- (asin x))]

[acos-neg (acos (- x)) (- PI (acos x))]

[atan-neg (atan (- x)) (- (atan x))])

; Hyperbolic trigonometric functions

(define-ruleset htrig-reduce (hyperbolic simplify)

#:type ([x real])

[sinh-def (sinh x) (/ (- (exp x) (exp (- x))) 2)]

[cosh-def (cosh x) (/ (+ (exp x) (exp (- x))) 2)]

[tanh-def (tanh x) (/ (- (exp x) (exp (- x))) (+ (exp x) (exp (- x))))]

[tanh-def (tanh x) (/ (- (exp (* 2 x)) 1) (+ (exp (* 2 x)) 1))]

[tanh-def (tanh x) (/ (- 1 (exp (* -2 x))) (+ 1 (exp (* -2 x))))]

[sinh-cosh (- (* (cosh x) (cosh x)) (* (sinh x) (sinh x))) 1]

[sinh-+-cosh (+ (cosh x) (sinh x)) (exp x)]

[sinh---cosh (- (cosh x) (sinh x)) (exp (- x))])

(define-ruleset htrig-expand (hyperbolic)

#:type ([x real] [y real])

[sinh-undef (/ (- (exp x) (exp (- x))) 2) (sinh x)]

[cosh-undef (/ (+ (exp x) (exp (- x))) 2) (cosh x)]

[tanh-undef (/ (- (exp x) (exp (- x))) (+ (exp x) (exp (- x)))) (tanh x)]

[cosh-sum (cosh (+ x y)) (+ (* (cosh x) (cosh y)) (* (sinh x) (sinh y)))]
```

```
[cosh-diff (cosh (- x y)) (- (* (cosh x) (cosh y)) (* (sinh x) (sinh y)))]

[cosh-2 (cosh (* 2 x)) (+ (* (sinh x) (sinh x)) (* (cosh x) (cosh x)))]

[cosh-1/2 (cosh (/ x 2)) (sqrt (/ (+ (cosh x) 1) 2))]

[sinh-sum (sinh (+ x y)) (+ (* (sinh x) (cosh y)) (* (cosh x) (sinh y)))]

[sinh-diff (sinh (- x y)) (- (* (sinh x) (cosh y)) (* (cosh x) (sinh y)))]

[sinh-2 (sinh (* 2 x)) (* 2 (* (sinh x) (cosh x)))]

[sinh-1/2 (sinh (/ x 2)) (/ (sinh x) (sqrt (* 2 (+ (cosh x) 1))))]

[tanh-sum (tanh (+ x y)) (/ (+ (tanh x) (tanh y)) (+ 1 (* (tanh x) (tanh y))))]

[tanh-2 (tanh (* 2 x)) (/ (* 2 (tanh x)) (+ 1 (* (tanh x) (tanh x))))]

[tanh-1/2 (tanh (/ x 2)) (/ (sinh x) (+ (cosh x) 1))]

[tanh-1/2* (tanh (/ x 2)) (/ (- (cosh x) 1) (sinh x))]

[sum-sinh (+ (sinh x) (sinh y)) (* 2 (* (sinh (/ (+ x y) 2)) (cosh (/ (- x y) 2))))]

[sum-cosh (+ (cosh x) (cosh y)) (* 2 (* (cosh (/ (+ x y) 2)) (cosh (/ (- x y) 2))))]

[diff-sinh (- (sinh x) (sinh y)) (* 2 (* (cosh (/ (+ x y) 2)) (sinh (/ (- x y) 2))))]

[diff-cosh (- (cosh x) (cosh y)) (* 2 (* (sinh (/ (+ x y) 2)) (sinh (/ (- x y) 2)))))])

(define-ruleset htrig-expand-fp-safe (hyperbolic fp-safe)

#:type ([x real])

[sinh-neg (sinh (- x)) (- (sinh x))]

[sinh-0 (sinh 0) 0]

[cosh-neg (cosh (- x)) (cosh x)]

[cosh-0 (cosh 0) 1])

(define-ruleset ahtrig-expand (hyperbolic)

#:type ([x real])

[asinh-def (asinh x) (log (+ x (sqrt (+ (* x x) 1))))]

[acosh-def (acosh x) (log (+ x (sqrt (- (* x x) 1))))]

[atanh-def (atanh x) (/ (log (/ (+ 1 x) (- 1 x))) 2)]
```

```
[acosh-2 (acosh (- (* 2 (* x x)) 1)) (* 2 (acosh x))]

[asinh-2 (acosh (+ (* 2 (* x x)) 1)) (* 2 (asinh x))]

[sinh-asinh (sinh (asinh x)) x]

[sinh-acosh (sinh (acosh x)) (sqrt (- (* x x) 1))]

[sinh-atanh (sinh (atanh x)) (/ x (sqrt (- 1 (* x x))))]

[cosh-asinh (cosh (asinh x)) (sqrt (+ (* x x) 1))]

[cosh-acosh (cosh (acosh x)) x]

[cosh-atanh (cosh (atanh x)) (/ 1 (sqrt (- 1 (* x x))))]

[tanh-asinh (tanh (asinh x)) (/ x (sqrt (+ 1 (* x x))))]

[tanh-acosh (tanh (acosh x)) (/ (sqrt (- (* x x) 1)) x)]

[tanh-atanh (tanh (atanh x)) x])

; Specialized numerical functions

(define-ruleset special-numerical-reduce (numerics simplify)

#:type ([x real] [y real] [z real])

[expm1-def (- (exp x) 1) (expm1 x)]

[log1p-def (log (+ 1 x)) (log1p x)]

[log1p-expm1 (log1p (expm1 x)) x]

[expm1-log1p (expm1 (log1p x)) x]

[hypot-def (sqrt (+ (* x x) (* y y))) (hypot x y)]

[hypot-1-def (sqrt (+ 1 (* y y))) (hypot 1 y)]

[fma-def (+ (* x y) z) (fma x y z)]

[fma-neg (- (* x y) z) (fma x y (- z))]

[fma-udef (fma x y z) (+ (* x y) z)])

(define-ruleset special-numerical-expand (numerics)

#:type ([x real] [y real])

[expm1-udef (expm1 x) (- (exp x) 1)]
```

```
[log1p-udef (log1p x) (log (+ 1 x))]
[log1p-expm1-u x (log1p (expm1 x))]
[expm1-log1p-u x (expm1 (log1p x))]
[hypot-udef (hypot x y) (sqrt (+ (* x x) (* y y)))])
(define-ruleset numerics-papers (numerics)
#:type ([a real] [b real] [c real] [d real])
; "Further Analysis of Kahan's Algorithm for
; the Accurate Computation of 2x2 Determinants"
; Jeannerod et al., Mathematics of Computation, 2013
;
; a * b - c * d ===> fma(a, b, -(d * c)) + fma(-d, c, d * c)
[prod-diff (- (* a b) (* c d))
(+ (fma a b (- (* d c)))
(fma (- d) c (* d c)))])
(define-ruleset bool-reduce (bools simplify fp-safe)
#:type ([a bool] [b bool])
[not-true (not TRUE) FALSE]
[not-false (not FALSE) TRUE]
[not-not (not (not a)) a]
[not-and (not (and a b)) (or (not a) (not b))]
[not-or (not (or a b)) (and (not a) (not b))]
[and-true-l (and TRUE a) a]
[and-true-r (and a TRUE) a]
[and-false-l (and FALSE a) FALSE]
[and-false-r (and a FALSE) FALSE]
[and-same (and a a) a]
```

```
[or-true-l (or TRUE a) TRUE]

[or-true-r (or a TRUE) TRUE]

[or-false-l (or FALSE a) a]

[or-false-r (or a FALSE) a]

[or-same (or a a) a])

(define-ruleset compare-reduce (bools simplify fp-safe-nan)

#:type ([x real] [y real])

[lt-same (< x x) FALSE]

[gt-same (> x x) FALSE]

[lte-same (<= x x) TRUE]

[gte-same (>= x x) TRUE]

[not-lt (not (< x y)) (>= x y)]

[not-gt (not (> x y)) (<= x y)]

[not-lte (not (<= x y)) (> x y)]

[not-gte (not (>= x y)) (< x y)])

(define-ruleset branch-reduce (branches simplify fp-safe)

#:type ([a bool] [b bool] [x real] [y real])

[if-true (if TRUE x y) x]

[if-false (if FALSE x y) y]

[if-same (if a x x) x]

[if-not (if (not a) x y) (if a y x)]

[if-if-or (if a x (if b x y)) (if (or a b) x y)]

[if-if-or-not (if a x (if b y x)) (if (or a (not b)) x y)]

[if-if-and (if a (if b x y) y) (if (and a b) x y)]

[if-if-and-not (if a (if b y x) y) (if (and a (not b)) x y)])

(define-ruleset complex-number-basics (complex simplify)
#: type ([x real] [y real] [a real] [b real] [c real] [d real])

[real-part (re (complex x y)) x]

[imag-part (im (complex x y)) y]

[complex-add-def (+.c (complex a b) (complex c d)) (complex (+ a c) (+ b d))]

[complex-def-add (complex (+ a c) (+ b d)) (+.c (complex a b) (complex c d))]

[complex-sub-def (-.c (complex a b) (complex c d)) (complex (- a c) (- b d))]

[complex-def-sub (complex (- a c) (- b d)) (-.c (complex a b) (complex c d))]

[complex-neg-def (neg.c (complex a b)) (complex (- a) (- b))]

[complex-def-neg (complex (- a) (- b)) (neg.c (complex a b))]

[complex-mul-def (*.c (complex a b) (complex c d)) (complex (- (* a c) (* b d)) (+ (* a
d) (* b c)))]

[complex-div-def (/.c (complex a b) (complex c d)) (complex (/ (+ (* a c) (* b d)) (+ (*
c c) (* d d))) (/ (- (* b c) (* a d)) (+ (* c c) (* d d))))]

[complex-conj-def (conj (complex a b)) (complex a (- b))]
```