```
;;; generated by SMT-LIB2 driver
;;; SMT-LIB2 driver: bit-vectors, common part
(set-option :produce-models true)
;;; SMT-LIB2: real arithmetic
;;; SMT-LIB2: integer arithmetic
(declare-sort t 0)

(declare-fun tzero () t)

(declare-fun tone () t)

(declare-fun prefix_mndt (t) t)

(declare-fun infix_pldt (t t) t)

(declare-fun infix_asdt (t t) t)

(declare-fun im () t)

(declare-fun r_to_t (Real) t)

(declare-fun real_part (t) Real)

(declare-fun im_part (t) Real)

(declare-fun t_real_part (t) t)

(declare-fun t_im_part (t) t)

(declare-sort d_frac 0)

(declare-fun value (d_frac) t)

(declare-fun add (d_frac d_frac) d_frac)

(declare-fun dyadic (Int Int) d_frac)

(declare-fun div_two (d_frac) d_frac)

(declare-fun d_cos (d_frac) t)

(declare-fun d_sin (d_frac) t)

(assert
;; h
  (not false))
(push)
(check-sat)
(get-model)
;; omega_d10
  (assert (= (value (dyadic 1 0)) tone))

;; omega_d12
  (assert (= (value (dyadic 1 2)) im))

;; value_zero
  (assert (= (value (dyadic 0 0)) tone))

;; im_Def
  (assert (= (infix_asdt im im) (prefix_mndt tone)))

;; NonTrivialRing
  (assert (not (= tzero tone)))

(push)
(check-sat)
(get-model)
;; h
  (assert
  (forall ((d d_frac))
  (= (value d) (infix_pldt (d_cos d) (infix_asdt im (d_sin d))))))

;; d_sin_def
  (assert (forall ((o d_frac)) (= (d_sin o) (t_im_part (value o)))))

;; d_cos_def
  (assert (forall ((o d_frac)) (= (d_cos o) (t_real_part (value o)))))

;; Div_two_dyadic
  (assert
  (forall ((k Int) (n Int))
  (=> (<= 0 n) (= (div_two (dyadic k n)) (dyadic k (+ n 1))))))

;; Dyadic_add
  (assert
  (forall ((k Int) (kqt Int) (n Int))
  (=> (<= 0 n) (= (add (dyadic k n) (dyadic kqt n)) (dyadic (+ k kqt) n)))))

;; Dyadic_inv
  (assert
  (forall ((k Int) (n Int))
  (=> (<= 0 n) (= (add (dyadic k n) (dyadic (- k) n)) (dyadic 0 0)))))

;; equal_d_frac
  (assert
  (forall ((d d_frac) (dqt d_frac)) (= (= d dqt) (= (value d) (value dqt)))))

;; t_im_part_def
  (assert (forall ((x t)) (= (t_im_part x) (r_to_t (im_part x)))))

;; t_real_part_def
  (assert (forall ((x t)) (= (t_real_part x) (r_to_t (real_part x)))))

;; Unic_decomp
  (assert
  (forall ((i t))
  (forall ((x Real) (y Real))
  (=> (= i (infix_pldt (r_to_t x) (infix_asdt im (r_to_t y))))
  (and (= x (real_part i)) (= y (im_part i)))))))

;; minus_tone
  (assert
  (forall ((a t)) (= (prefix_mndt a) (infix_asdt (prefix_mndt tone) a))))

;; Unitary
  (assert (forall ((x t)) (= (infix_asdt tone x) x)))

;; Mul_distr_l
  (assert
  (forall ((x t) (y t) (z t))
  (= (infix_asdt x (infix_pldt y z)) (infix_pldt (infix_asdt x y)
                                     (infix_asdt x z)))))

;; Comm
  (assert (forall ((x t) (y t)) (= (infix_pldt x y) (infix_pldt y x))))

;; Inv_def_l
  (assert (forall ((x t)) (= (infix_pldt (prefix_mndt x) x) tzero)))

(push)
(check-sat)
(get-model)
```