```
1. Algebraic structure: a DSL for monoids
\begin{code}
module P1 where
import Prelude hiding (Monoid)
class Monoid m where unit :: m; op :: m->m->m     -- 1a class
data ME v = Unit | Op (ME v) (ME v) | V v         -- 1b data
instance Monoid (ME v)  where unit=Unit; op=Op    -- 1b instance
instance Monoid Bool    where unit=True; op=(&&)  -- 1c Bool
instance Monoid Integer where unit=0;    op=(+)   -- 1c Integer
eval :: Monoid m => (v->m) -> (ME v -> m)         -- 1d eval
eval f = e where e Unit = unit; e (Op x y) = op (e x) (e y); e (V v) = f v
evalB :: (v->Bool)   ->(ME v->Bool);    evalB=eval-- 1e evalB
evalI :: (v->Integer)->(ME v->Integer); evalI=eval-- 1e evalI
e1=Op(V"a")Unit;e2=Op(V"b")e1;e3=V"b"`Op`V"a"     -- 1e three expr
fB=("a"==);testB=map(evalB fB)[e1,e2,e3]==[True,False,False]
fI"a"=1;fI"b"=2;testI=map(evalI fI)[e1,e2,e3]==[1,3,3]
main=print$testB&&testI
\end{code}
----------------------------------------------------------------
2. Differentiable
2a.
\begin{code}
module P2 where
type REAL = Double
data U    -- or |U :: PowerSet REAL|
f :: U -> REAL
t :: U
f':: U -> REAL
type RPos = REAL -- should be real numbers > 0
epsilon :: RPos
delta :: (U,RPos)->RPos

(f,t,f',epsilon,delta) = undefined
\end{code}
2b.
\begin{spec}
DifferentiableAt(f,f',t) =
  Exists delta.
    Forall epsilon.
        (t-delta(t,epsilon),t+delta(t,epsilon)) `included` U
      &&
       Forall h.
         (0 < abs h < delta(t,epsilon)) =>
         abs ((f(t+h)-f(t))/h - f'(t)) < epsilon
\end{spec}
2c. The key expression in the formula is
\begin{spec}
  (f(t+h)-f(t))/h - f'(t)
= let f=sq; f'=tw
  (sq(t+h)-sq(t))/h - tw(t)
= -- def. of sq, def. of tw
  ((t+h)*(t+h)-t*t)/h - 2*t
= -- simplification
  (t^2+2*t*h+h^2-t^2)/h - 2*t
= -- simplification
  2*t+h - 2*t
= -- simplification
  h
\end{spec}
Thus the inner part is
\begin{spec}
  Forall h. (0 < abs h < delta2(t,epsilon)) => (abs h < epsilon)
\end{spec}
We can see that this is guaranteed if we defined
\begin{code}
delta2 :: (U,RPos)->RPos
delta2 (t,epsilon) = epsilon
\end{code}
2d. We want to find a function |dfg| such that
  Forall h.  (0 < abs h < dfg(t,epsilon)) =>
             abs (((f+g)(t+h)-(f+g)(t))/h - (f+g)'(t)) < epsilon
and we know there is a |df| such that
  Forall h.  (0 < abs h < df(t,epsilon)) =>
             abs ((f(t+h)-f(t))/h - f'(t)) < epsilon
and a |dg| such that
  Forall h.  (0 < abs h < dg(t,epsilon)) =>
             abs ((g(t+h)-g(t))/h - g'(t)) < epsilon
```

```
We compute with the core expression:
    ((f+g)(t+h)-(f+g)(t))/h - (f+g)'(t)
  = -- def. of + for functions, derive for +
    (f(t+h)+g(t+h)-f(t)-g(t))/h - (f'(t)+g'(t))
  = -- rearrange terms
    (f(t+h)-f(t))/h - f'(t)  +  (g(t+h)-g(t))/h - g'(t)
  < -- [XX]: require that (abs h < df(t,epsilon/2)) and that (abs h < dg(t,epsilon/2))
    epsilon/2  +  epsilon/2
  =
    epsilon
To fulfill [XX] we define
  dfg(t,epsilon) = min (df(t,epsilon/2)) (dg(t,epsilon/2))
----------------------------------------------------------------
3. Laplace: Solve f(t) = (f''(t) + f'(t))/2, f(0) = a, f'(0) = b
Short answers:
a) fs = integ fs'  a; fs'  = integ fs'' b; fs'' = 2*fs-fs'
   fs = a : b : a-b/2 : ...
b) f t = ((2*a+b)/3)*exp(t)+((a-b)/3)*exp(-2*t)
More details:
3a. ----------------
\begin{code}
module P3 where
import Data.Ratio
import PS
fs   = integ fs'  a
fs'  = integ fs'' b
fs'' = 2*fs-fs'
\end{code}
Computing the first few coefficients (four here, three in the exam question):
fs   = a     :  b      :  a-b/2   : -a/3+b/2 : ...
-- move b up, move 2*a-b up, divide by 2, etc
fs'  = b     :  2*a-b   : -a+3*b/2 : ...
-- move 2*a-b up,
fs'' = 2*a-b : -2*a+3*b : ...
3b. ----------------
  f(t) = (f''(t) + f'(t))/2, f(0) = a, f'(0) = b
simplified to
  2*f - f'' - f' = 0
Start calculating Laplace of both side:
  LHS
=
  L (2*f - f'' - f') s
= -- linearity
  2*L f s - L f'' s - L f' s
= -- L f' s = -f 0 + s*L f s = -a + s*L f s
  2*L f s - L f'' s + a - s*L f s
= -- L f'' s = -f' 0 - s*f 0 + s^2*L f s = -b - a*s + s^2*L f s
  2*L f s + b + a*s - s^2*L f s + a - s*L f s
= -- simplify
  (2-s-s^2)*L f s  + a + b + a*s
-- Note that s=1 and s=-2 are zeros of (2-s-s^2)=-(s-1)*(s+2)
Thus
  L f s = (a+b+a*s)/(s-1)/(s+2)
Ansatz: L f s = A/(s-1) + B/(s+2) and multiply both sides by (s-1)*(s+2)
    a+b+a*s == A*(s+2) + B*(s-1)
  <= the same but with s=1 and with s=-2
    a+b+a == A*(1+2)  &&   a+b-2*a == B*(-2-1)
  <=> simplify
    2*a+b == 3*A  &&  b-a == -3*B
  <=>
    A == (2*a+b)/3  &&  B == (a-b)/3
Inverse transform by inspection:
  f t = A*exp(t)+B*exp(-2*t)   -- with A and B as above
-- Checking:
  f'  t = A*exp(t)-2*B*exp(-2*t)
  f'' t = A*exp(t)+4*B*exp(-2*t)
  Original RHS
= -- def.
  (f''(t) + f'(t))/2
= -- Def. of f' t and f'' t, then simplification
  A*exp(t)+B*exp(-2*t)
= -- Ansatz
  f t
= -- def.n
  Original LHS
f  0 = A*1+B*1 = (2*a+b)/3 + (a-b)/3 = a  -- OK
f' 0 = A-2*B = (2*a+b)/3 - 2*(a-b)/3 = b  -- OK
----------------------------------------------------------------
```

```
4. Homomorphisms.
4a.  H0(toComplex,zero,zero)
   = -- def. of H0
     toComplex zero == zero
   = -- def. of toComplex
     CS (zero, 0) == CS (0,0)
   = -- def. of zero
     CS (0, 0) == CS (0,0)
   = -- reflexivity
     True
4b.  H2(toComplex,add,add)
   = def. of H2, let t=toComplex for brevity
     Forall x (Forall y (t(x+y) == addC (t x) (t y)))
   = def. of t
     Forall x (Forall y (CS (x+y,0) == addC (CS (x,0)) (CS (y,0)))
   = def. of addC
     Forall x (Forall y (CS (x+y,0) == CS (x+y,0+0)))
   = Simplification
     True
4c.  H2(toComplex,scale,scale)
  -- type problem: toComplex :: REAL -> CC
  -- scaleRR :: REAL -> REAL -> REAL  -- OK
  -- scaleCC :: CC -> CC -> CC        -- not OK
4d.  H2(circle,(+),mulC)
   = -- def. of H2, shorten circle to just c
     Forall x (Forall y (c(x+y) == mulC (c x) (c y)))
   = -- def. of c
     Forall x (Forall y (  CS (cos (x+y), sin (x+y)) ==
                           mulC (CS (cos x, sin x)) (CS (cos y, sin y))))
   = -- def. of mulC
     Forall x (Forall y (  CS  (cos (x+y), sin (x+y)) ==
                           CS  (  (cos x)*(cos y) - (sin x)*(sin y)
                               ,  (sin x)*(cos y) + (cos x)*(sin y) )))
   = -- def. == for Complex
     Forall x (Forall y (    cos (x+y) == (cos x)*(cos y) - (sin x)*(sin y)
                         &&  sin (x+y) == (sin x)*(cos y) + (cos x)*(sin y) ))
   = -- trigonometry
     True
4e.  Here is one way to work towards a solution. There are other ways.
     Exists a, b. H1(addC i,mulC a,mulC b)
   -- It looks suspicious, let's try to negate it.
     Forall a, b.  Exists z. addC i (mulC a z) /= mulC b (addC i z)
   -- Try with some simple values of z, starting with 0
     addC i (mulC a 0) /= mulC b (addC i 0)
   = -- simplify
     i /= mulC b i
   -- This is true for all b/=1
   -- Thus we can pick z=0 for all a and almost all b.
   -- What about when b=1?
     addC i (mulC a z) /= mulC b (addC i z)
   = addC i (mulC a z) /= mulC 1 (addC i z)
   = addC i (mulC a z) /= addC i z
   = mulC a z /= z
   = -- assume z/=0, for example z=1
     a /= 1
   Thus we can pick z=1 when b=1 and a/=1.
   What about a=b=1?
     addC i (mulC a z) /= mulC b (addC i z)
   = addC i z /= addC i z
   = False
Thus, we cannot prove the negation, the law seems to hold.
Check:
     Exists a, b. H1(addC i,mulC a,mulC b)
   -- let a=b=1
     H1(addC i,mulC 1,mulC 1)
   -- alg. properties
     H1(addC i,id,id)
   -- expand H1
     addC i (id z) == id (addC i z)
   -- simplify
     addC i z == addC i z
   -- simplify
     True
Thus, yes, there exist a and b, both 1, so that addC i is a homomorphism.
```