Domain-Specific Languages of Mathematics
Course codes: DAT326 / DIT983

Patrik Jansson

Solutions to exam 2024-03-15

Contact	Patrik Jansson, 072 985 2033.
Results	Announced within 15 workdays
Exam check	2023-04-08, 12.15-12.45 in EDIT 6128
Aids	One textbook of your choice (Domain-Specific Languages of Mathematics, or Beta - Mathematics Handbook, or Rudin, or Adams and Essex, or ...). No printouts, no lecture notes, no notebooks, etc.
Grades	To pass you need **a minimum of 5p on each question (1 to 4)** and also reach these grade limits: 3: >=48p, 4: >=65p, 5: >=83p, max: 100p

Remember to write legibly. Good luck!

For reference: the learning outcomes. Some are tested by the hand-ins, some by the written exam.

- Knowledge and understanding
  - design and implement a DSL (Domain-Specific Language) for a new domain
  - organize areas of mathematics in DSL terms
  - explain main concepts of elementary real and complex analysis, algebra, and linear algebra
- Skills and abilities
  - develop adequate notation for mathematical concepts
  - perform calculational proofs
  - use power series for solving differential equations
  - use Laplace transforms for solving differential equations
- Judgement and approach
  - discuss and compare different software implementations of mathematical concepts

1. [25p] **Algebraic structure:** *ring* (lightly edited from the Wikipedia entry)

> ...a *ring* is a set R equipped with two binary operations satisfying properties analogous to those of addition and multiplication of integers. Ring elements may be numbers such as integers or complex numbers, but they may also be non-numerical objects such as polynomials, square matrices, functions, and power series.
>
> Formally, a ring is an abelian group whose operation is called addition, with a second binary operation called multiplication that is associative, is distributive over the addition operation, and has a multiplicative identity element.

Some of the laws are (for all a, b, c in R):

$(a + b) + c = a + (b + c)$
$a + 0 = a$
$a + (-a) = 0$
$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
$a \cdot 1 = a$

(a) Define a type class *Ring* that corresponds to the structure *ring*.

(b) Define a datatype $R\ v$ for the language of ring expressions (with variables of type v) and define a *Ring* instance for it. (These are expressions formed from applying the ring operations to the appropriate number of arguments, e.g., all the left hand sides and right hand sides of the above equations.)

(c) Find and implement two other instances of the *Ring* class. Make sure the laws are satisfied.

(d) Give a type signature for, and define, a general evaluator (on the basis of an assignment function) from the syntax of $R\ v$ expressions to any semantic type with a *Ring* instance.

(e) Specialise the evaluator to the two *Ring* instances defined in (1c). Take three ring expressions (of type *R String*), give the appropriate assignments and compute the results of evaluating, in each case, the three expressions.

Each question carries 5pts.

Solution First some supporting code (not expected at the exam).

```
 {-# LANGUAGE GADTs #-}
import qualified Prelude
import Prelude hiding (Num (..))
law1 a b c = (a + b) + c == a + (b + c)
law2 a     = a + zero == a
law3 a     = a + (neg a) == zero
law4 a b c = a · (b + c) == (a · b) + (a · c)
law5 a     = a · one == a
```

(a) Type class *Ring*:

```
class Ring a where
  (+)  :: a → a → a
  zero :: a
  neg  :: a → a
  (·)  :: a → a → a
  one  :: a
```

(b) Datatype *R v*:

```
data R v where
  Add  :: R v → R v → R v
  Zero :: R v
  Neg  :: R v → R v
  Mul  :: R v → R v → R v
  One  :: R v
  V    :: v → R v
deriving Show
instance Ring (R v) where
  (+) = Add; zero = Zero; neg = Neg; (·) = Mul; one = One
```

(c) Two more *Ring* instances. *Integer* is the classical ring, and the function instance just lifts all operations pointwise (and thus preserves all the laws).

```
instance Ring Integer where
  (+) = (Prelude.+); zero = 0; neg = Prelude.negate; (·) = (Prelude.∗); one = 1
instance Ring a ⇒ Ring (k → a) where
  (+) = addf; zero = zerof; neg = negf; (·) = mulf; one = onef

addf, mulf :: Ring a ⇒ (k → a) → (k → a) → (k → a)
addf f g = λx → f x + g x
mulf f g = λx → f x · g x

negf  :: Ring a ⇒ (k → a) → (k → a)
negf f = λx → neg (f x)

zerof, onef :: Ring a ⇒ (k → a)
zerof = \_ → zero
onef  = \_ → one
```

(d) Evaluator:

```
eval :: Ring a ⇒ (v → a) → R v → a
eval var = e where
  e (Add x y) = e x + e y
  e (Zero)    = zero
  e (Neg x)   = neg (e x)
  e (Mul x y) = e x · e y
  e (One)     = one
  e (V v)     = var v
```

(e) Specialise + examples + evaluation:

```
evalI :: (v →  Integer)      → R v → Integer
evalF :: (v → k → Integer) → R v → k → Integer
evalI = eval; evalF = eval

e1, e2, e3 :: R String
e1 = V "x"; e2 = e1 + e1; e3 = neg e2 · e2

funI :: String → Integer
funI "x"  = 2
funI _    = 3

funF :: String → (Integer → Integer)
funF "x" = id
funF _   = const 0

testIs = map (evalI funI) [e1, e2, e3] == [2, 4, −16]
testFs = map apply1 (map (evalF funF) [e1, e2, e3]) == [1, 2, −4]

apply1 :: (Integer → a) → a
apply1 f = f 1
```

2. [25p] **Laplace**

Consider the following differential equation:

$$f(t) + (3f'(t) + f''(t))/2 = e^{-3t}, \quad f(0) = 1, \quad f'(0) = 0$$

(a) [10p] Solve the equation assuming that f can be expressed by a power series *fs*, that is, use *integ* and the differential equation to express the relation between *fs*, *fs'*, *fs''*. What are the first four coefficients of *fs*? Explain how you compute them.

(b) [15p] Solve the equation using the Laplace transform. You should need this formula (note that α can be a complex number) and the rules for linearity + derivative:

$$\mathcal{L}\,(\lambda t.\, e^{\alpha * t})\, s = 1/(s - \alpha)$$

Show that your solution does indeed satisfy the three requirements.

Solution

(a) Power series version: let $fs, fs', fs'', si, co :: PS\ \mathbb{Q}$ be such that

$$map\ eval\ [fs, fs', fs'', expm3] == [f, f', f'', exp \circ ((-3)*)]$$

Then we have

$$fs + scaleP\ (3\,/\,2)\ fs' + scaleP\ (1\,/\,2)\ fs'' = expm3$$

which can be rearranged to

$$scaleP\ (1\,/\,2)\ fs'' = expm3 - fs - scaleP\ (3\,/\,2)\ fs'$$

and scaled by 2 to get

$$fs'' = scaleP\ 2\ (expm3 - fs) - scaleP\ 3\ fs'$$

We also have

$$\begin{aligned} fs &= integP\ 1\ fs' \\ fs' &= integP\ 0\ fs'' \\ expm3 &= integP\ 1\ (scaleP\ (-3)\ expm3) \end{aligned}$$

Now we can start filling in the coefficients

i. Step 0:

$$\begin{aligned} fs\ &!!\ 0 = 1 \\ fs'\ &!!\ 0 = 0 \\ expm3\ &!!\ 0 = 1 \\ fs''\ &!!\ 0 = 2 * (1 - 1) - 3 * 0 = 0 \end{aligned}$$

ii. Step 1:

$$\begin{aligned} fs\ &!!\ 1 = 0\,/\,1 = 0 \\ fs'\ &!!\ 1 = 0\,/\,1 = 0 \\ expm3\ &!!\ 1 = -3 * 1 = -3 \\ fs''\ &!!\ 1 = 2 * (-3 - 0) - 3 * 0 = -6 \end{aligned}$$

iii. Step 2:

$$\begin{aligned} fs\ &!!\ 2 = 0\,/\,2 = 0 \\ fs'\ &!!\ 2 = -6\,/\,2 = -3 \end{aligned}$$

iv. Step 3:

$$fs\ !!\ 2 = -3\,/\,3 = -1$$

Thus

$$test = takeP\ 4\ fs == [1, 0, 0, -1]$$

(b) Analytic solution with Laplace transform

i. Step 0: Let $F = L\,f$ below and start by multiplying the equation by 2:
$LHS = 2\,f + 3\,f' + f''$
$RHS\ t = 2 * exp\ (-3 * t)$
and still $f\ 0 = 1$, $f'\ 0 = 0$.

ii. Step 1: Use the Laplace-D-law to compute $L\,f'$
$L\,f'\ s = L\,(D\,f)\ s = -1 + s * L\,f\ s = -1 + s * F\ s$

iii. Step 2: Use the Laplace-D-law to compute $L\,f''$

$$\begin{array}{lll} L\,f''\ s & = & \text{-- Def. of } D \\ L\,(D\,f')\ s & = & \text{-- } f'\ 0 = 0 \\ -0 + s * L\,f'\ s & = & \text{-- comp. above} \\ s * (-1 + s * F\ s) & = & \text{-- simplify} \\ -s + s\hat{\ }2 * F\ s & & \end{array}$$

iv. Then we apply L to LHS:

$$\begin{array}{lll} L\,(2 * f + 3 * f' + f'')\ s & = & \text{-- Linearity, def. of } F \\ 2 * F\ s + 3 * L\,f'\ s + L\,f''\ s & = & \text{-- L-D-law results from above} \\ 2 * F\ s + 3 * (-1 + s * F\ s) + (-s + s\hat{\ }2 * F\ s) & = & \text{-- Simplify} \\ (2 + 3 * s + s\hat{\ }2) * F\ s - 3 - s & = & \text{-- Factor} \\ (s+1) * (s+2) * F\ s - (s+3) & & \end{array}$$

v. and we apply to the RHS
$L\,(\lambda t \to 2 * exp\ (-3 * t))\ s$
$=$ -- Laplace law for exponentials
$2\,/\,(s+3)$

By combining LHS=RHS and moving $s+3$ to the right we get:
$(s+1) * (s+2) * F\ s = 2\,/\,(s+3) + (s+3)$

vi. and can start with partial fraction decomposition: Ansatz:
$F\ s = A\,/\,(s+1) + B\,/\,(s+2) + C\,/\,(s+3)$

vii. Multiply by $(s+3)$ to get a polynomial equation:
$(s+1) * (s+2) * (s+3) * F\ s = 2 + (s+3)\hat{\ }2$

viii. Substitute the ansatz and simplify:
$A * (s+2) * (s+3) + B * (s+1) * (s+3) + C * (s+1) * (s+2) = 2 + (s+3)\hat{\ }2$

ix. Solve the polynomial equation by specialising to three cases (s=-1,-2,-3):
$s = -1 : A * (-1+2) * (-1+3) = 2 + (-1+3)\hat{\ }2 \Leftrightarrow 2\ A = 6 \Leftrightarrow A = 3$
$s = -2 : B * (-2+1) * (-2+3) = 2 + (-2+3)\hat{\ }2 \Leftrightarrow -B = 3 \Leftrightarrow B = -3$
$s = -3 : C * (-3+1) * (-3+2) = 2 + (-3+3)\hat{\ }2 \Leftrightarrow 2\ C = 2 \Leftrightarrow C = 1$

x. Thus
$F\ s = 3\,/\,(s+1) - 3\,/\,(s+2) + 1\,/\,(s+3)$

xi. which we can recognize as the transform of
$f\ t = 3 * exp\ (-t) - 3 * exp\ (-2 * t) + exp\ (-3 * t)$

xii. Checking: **Important!**
$f'\ t = -3 * exp\ (-t) + 6 * exp\ (-2 * t) - 3 * exp\ (-3 * t)$
$f''\ t = 3 * exp\ (-t) - 12 * exp\ (-2 * t) + 9 * exp\ (-3 * t)$
$f\ 0 = 3 - 3 + 1 = 1$ -- OK!
$f'\ 0 = -3 + 6 - 3 = 0$ -- OK!

$$\begin{array}{ll} LHS\ t = 2 * f\ t + 3 * f'\ t + f''\ t & \\ \quad = \ \ 2 * (3 * exp\ (-t) - 3 * exp\ (-2 * t) + exp\ (-3 * t)) & \\ \quad\ \ + 3 * (-3 * exp\ (-t) + 6 * exp\ (-2 * t) - 3 * exp\ (-3 * t)) & \\ \quad\ \ + \ \ (3 * exp\ (-t) - 12 * exp\ (-2 * t) + 9 * exp\ (-3 * t)) & \\ \quad = (6 - 9 + 3) * exp\ (-t) + (-6 + 18 - 12) * exp\ (-2 * t) + (2 - 9 + 9) * exp\ (-3 * t) & \\ \quad = 2 * exp\ (-3 * t) & \\ \quad = RHS\ t \quad \text{-- OK!} & \end{array}$$

3. [25p] **Type / Proof / LinAlg:** Random walk on the integers

   A random walk on the integers is a stochastic process where if you start at an integer i, there is a 50/50 chance of going to $i+1$ or $i-1$ at every step. To each stochastic process, we can assign a matrix which describes a "one-step transition".

   (a) [3p] Give the type of the matrix corresponding to a random walk on the integers.

   (b) [4p] Give a Haskell implementation for this matrix.

   For practical purposes, dealing with infinite matrices can be tricky. For any given n, for a random walk of length n starting at 0, we will only need the matrix for the integers $[-n, \cdots, n]$. Luckily we can crop out this part of the matrix.

   (c) [3p] Why do we need no other integers?

   For the following exercises, you may assume we have for each $m : \mathbb{N}$ a type *Fin m* of exactly m elements, labeled $F\ 0, F\ 1, ..., F\ (m-1)$. You don't have to worry about how this would be implemented. In the following questions, $n : \mathbb{N}$ is fixed and at least 2.

   (d) [3p] What would be the type of a function *crop*, which "cuts out the relevant part of the matrix"?

   (e) [4p] Implement *crop*.

   (f) [5p] Using matrix multiplication, compute "by hand" the cropped (5x5) matrix corresponding to taking two steps.

   (g) [3p] What is the probability being back at your starting point $i = 0$ after $n = 2$ steps? Explain how you get that from the "two-step" matrix.

Solution

(a) This is a stochastic process, and the type for a matrix corresponding stochastic process is $G \to \mathit{Vector}\ \mathbb{R}\ G$, for G the domain, which in this case is Int. So the type of the matrix should be $\mathit{Int} \to \mathit{Vector}\ \mathbb{R}\ \mathit{Int}$, alternatively we can denote this as $\mathit{Matrix}\ \mathbb{R}\ \mathit{Int}\ \mathit{Int}$.

(b) The Haskell implementation can look like this:

```
transitionchance :: Int → Int → ℝ
transitionchance n m = if |n − m| == 1 then 0.5 else 0

transitionmatrix :: Matrix Real Int Int
transitionmatrix n = V (λm → transitionchance n m)
```

(c) If we start at 0, we can only reach distance n after n steps. The probability that we go outside this range is therefore 0, and we don't need to consider it.

(d) $\mathit{crop} :: \mathit{Matrix}\ \mathbb{R}\ \mathit{Int}\ \mathit{Int} \to \mathit{Matrix}\ \mathbb{R}\ (\mathit{Fin}\ (2 * n + 1))\ (\mathit{Fin}\ (2 * n + 1))$

(e) $\mathit{crop}\ \mathit{mat}\ (E\ i) = V\ (\lambda(E\ j) \to \mathit{mat}\ (i - n)\ !\ (j - n))$.

(f) The relevant part of the "one-step" transition matrix M is given by

$$M = \begin{array}{c} \\ -2 \\ -1 \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} -2 & -1 & 0 & 1 & 2 \end{array} \\ \begin{pmatrix} 0 & .5 & 0 & 0 & 0 \\ .5 & 0 & .5 & 0 & 0 \\ 0 & .5 & 0 & .5 & 0 \\ 0 & 0 & .5 & 0 & .5 \\ 0 & 0 & 0 & .5 & 0 \end{pmatrix} \end{array} \qquad (1)$$

The two-step transition is given by M^2:

$$\begin{array}{c} \\ -2 \\ -1 \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} -2 & -1 & 0 & 1 & 2 \end{array} \\ \begin{pmatrix} .25 & 0 & .25 & 0 & 0 \\ 0 & .5 & 0 & .25 & 0 \\ .25 & 0 & .5 & 0 & .25 \\ 0 & .25 & 0 & .5 & 0 \\ 0 & 0 & .25 & 0 & .25 \end{pmatrix} \end{array} \qquad (2)$$

(g) The middle number 0.5 in M^2 is the probability of starting at 0 and ending at 0. It can also be found by analysing the possible paths that go from 0 to 0: two ($+-$ and $-+$) out of the four ($++$, $+-$, $-+$, $--$) equally likely paths, thus the probability is 0.5.

4. [25p] **Typing / Proof:** Homogeneous function

Consider the following Wikipedia quote on "homogeneous functions":

> In mathematics, a homogeneous function is a function of several variables such that the following holds: If each of the function's arguments is multiplied by the same scalar, then the function's value is multiplied by some power of this scalar; the power is called the degree of homogeneity, or simply the degree. That is, if k is an integer, a function f of n variables is homogeneous of degree k if
>
> $$f(sx_1, \ldots, sx_n) = s^k f(x_1, \ldots, x_n)$$
>
> for every $x_1, \ldots, x_n$, and $s \neq 0$.

(a) [5p] Give types for k, n, s, x_1, $\ldots$, x_n, f. Explain your reasoning.

(b) [5p] Define (in first-order logic) the predicate $\mathit{Hom}(f, k)$ which expresses that the function f of *two* arguments is homogeneous of degree k.

(c) [5p] Prove or disprove $\exists\ k.\ \ \mathit{Hom}(g, k)$ where $g(x, y) = x^2 * y - 2 * y^3$.

(d) [5p] Formalise the property of a two-argument function f to be *not* homogeneous of any degree using the predicate from item 4b. Simplify the logic statement by pushing negation through the quantifiers.

(e) [5p] Find an example of a two-argument function, which is not homogeneous of any degree. Prove your claim.

Solution

(a) The text says $k : \mathbb{Z}$ and we can guess that $n : \mathbb{N}$. Let $s : S$, $x_i : X$, and $f : V \to Y$ for some types S (for "scalars") and (as yet unknown) types V, Y. We note that the input to f can be the x_i, thus $V = X^n$. But the inputs to f can also be sx_i, thus it seems like there is an invisible operation $\mathit{scale} : S \to X \to X$, or that $X = S$. We also see (from s^k) that scalars can be multiplied (by some invisible $(*) : S \to S \to S$). And the result is used to scale the output of f, thus $Y = X$ seems reasonable. One concrete instance could be $X = \mathbb{R}$ and $S = X - \{0\}$.

(b) $\mathit{Hom}(f, k) = \forall\ s \neq 0.\ \ \forall\ x, y.\ \ f(s * x, s * y) = s^k * f(x, y)$

(c) Compute $g(sx, sy) = (sx)^2 * (sy) - 2(sy)^3 = s^3x^2y - 2s^3y^3 = s^3(x^2y - 2y^3) = s^3 g(x, y)$. Thus we have $\mathit{Hom}(g, 3)$.

(d) $\mathit{NH}(f) = \forall\ k : \mathbb{Z}.\ \ \neg\,(\mathit{Hom}(f, k)) = \forall\ k.\ \ \exists\ s \neq 0.\ \ \exists\ x, y.\ \ f(s * x, s * y) \neq s^k * f(x, y)$

(e) Let $f(x, y) = 1 + x$ and compute the difference between the left-hand and right-hand sides (for an arbitrary k): $f(sx, sy) - s^k f(x, y) = (1 + sx) - s^k(1 + x) = (1 - s^k) + (s - 1)x$. We want to find $s \neq 0$, x, y such that this difference is not zero. The variable y is not even involved so we can pick any number, for example $y = 0$. Let $s = 2$: then we get $(1 - 2^k) + x$ and we can pick any $x \neq 2^k - 1$, for example $x = 2^k$ which gives the difference $1 \neq 0$.