KU LEUVEN kulak

Groep Wetenschap & Technologie Kulak
2e Bach. Informatica: Declaratieve Talen
Patrick De Causmaecker, Pieter Bonte

Examen Declaratieve Talen

20 juni 2025 Vincent Van Schependom

Het examen duurt 3 uur. Toegelaten documentatie is www.swi-prolog.org, hoogle.haskell.org, samen met de cursus en de slides. Zend op het einde je antwoorden naar patrick.decausmaecker@kuleuven.be en pieter.bonte@kuleuven.be.

1 Prolog ($\frac{5}{14}$)

We willen een predicaat `star(+Commands)` dat, gegeven een lijst van commando's, sterretjes op een bord print. Er zijn 5 commando's: `up`, `down`, `left`, `right` en `star`.

Er is een probleem, want de linkerbovenhoek van het bord moet altijd in de eerste kolom staan. Dit wil zeggen dat de volgende commando's allemaal hetzelfde printen naar het scherm:

```
?- star([right,right,right,right,right,star]).
*
?- star([left,left,left,left,left,star]).
*
?- star([down,up,left,right,left,left,star]).
*
```

1.1

Schrijf het predicaat `collect_stars(+Commandos, -Sterren)` dat, gegeven een lijst van commando's, een lijst van sterren genereert.

```
?- collect_stars([right,right,right,right,right,star],S).
S = [star(5,0)]

?- collect_stars([down,star,left,down,right,up,up,star,right],S).
S = [star(0,-1),star(0,0)]
```

Tip: bij deze eerste implementatie moet je er nog geen rekening mee houden dat we moeten starten op positie 0.

1.2

Schrijf het predicaat `write1line(+StarList, +Y, +StartCol)` die één lijn sterren afdrukt uit `StarList`. Hierbij is `Y` de coördinaat van de rij die we willen afdrukken en `StartCol` de kolom waar het bord moet starten.

```
?- write1line([star(0,0),star(1,1),star(2,0)],0,0).
* *

?- write1line([star(0,0),star(1,1),star(2,0)],0,-5).
* *
```

1.3

Vervolledig nu je implementatie door het predicaat `star/1` te schrijven dat de initialen print van je favoriete vak:

```
?- star([star,up,star,up,star,up,star,up,right,right,right,star,left,star,left,
star,left,star,left,star,left,star,left,left,left,left,left,down,star,
down,star,down,star,down,left,star,left,star,left,star,left,star,up,star,up,star,
up,star,up,star,right,star,right,star,right,star]).

****     *******
*   *       *
*   *       *
*   *       *
****        *
```

2 Haskell ($\frac{5}{14}$)

2.1 Veeltermen hebben nulpunten en dus coëfficiënten

In deze oefening maken we gebruik van `Double`, en van `Complex` uit `Data.Complex`.

Veeltermen hebben een lijst van coëfficiënten en worden voorgesteld door **Polynoom**. Een lijst van nulpunten wordt voorgesteld door **Nullen**. Schrijf een functie `polynoom` die, gegeven een aantal nulpunten, de coëfficiënten berekent van de veelterm met 1 als hoogstegraadsterm.

```
ghci> polynoom (N [1])
P [1.0,-1.0]

ghci> polynoom (N [2,3])
P [1.0,-5.0,6.0]

ghci> polynoom (N [2,3,4])
P [1.0,-8.0,26.0,24.0]

ghci> polynoom (N [1.0 :+ 2.0])
P [1.0 :+ 0.0, -1.0 :+ 2.0]
```

Hint: doe eens $(x-a)(x-b)(x-c)$. (Deze hint werd niet gegeven op het examen zelf)

2.2 De voorstelling kan beter

[1] De voorstelling van de polynomen hierboven was eerder arbitrair. Schrijf een functie show die veeltermen duidelijker afprint.

```
ghci> polynoom (N [1.0,2.0])
+ 1.0x^2 -3.0x + 2.0

ghci> polynoom (N [2,3])
+ 1.0x^2 - 5.0x + 6.0

ghci> polynoom (N [2,3,4])
+ 1.0x^3 - 8.0x^2 + 26.0x + 24.0

ghci> polynoom (N [1.0 :+ 2.0])
+ 1.0x - (1.0+2.0i)
```

[1] Deze oefening is facultatief indien je 2.3.2 maakt

2.3 De `State` Monad

2.3.1 Random getallen

[2] Schrijf een functie `randomAdd` die een random getal toevoegt aan een lijst. Gebruik de implementatie in bijlage voor het gebruik van de `State` Monad, alsook de `randomList` methode en de `RandomGen` typeclass die gezien werden in de les.

```
ghci> radd4 = randomAdd [] >>= randomAdd >>= randomAdd >>= randomAdd

ghci> fst $ runState radd4 (R 17)
[185,89,81,17]

ghci> snd $ runState radd4 (R 17)
R 3145170771490243124
```

Toon de wet van associativiteit van Monads aan door `radd4assoc` te definiëren waarbij deze wet wordt gebruikt.

2.3.2 Vergeet niet wat werd gedaan

Schrijf functies `livingListPlus` en `livingListMin` die een `State` teruggeven waarbij een element ofwel aan een lijst kan worden toegevoegd ofwel verwijderd kan worden uit die lijst.

```
ghci> runState (livingListPlus 10 [1,2,3]) []
([10,1,2,3],[T 10])

ghci> runState (livingListMin 10 [1,2,3,10]) []
([1,2,3],[M 10])

ghci> runState (livingListMin 10 [1,2,3]) []
([1,2,3],[])

ghci> runState (livingListMin 2 [1,2,3,2,3,1,2,1]) []
([1,3,3,1,1],[M 2,M 2,M 2])
```

Gebruik bovenstaande functies om een functie `doeDeKlinkersWeg` te schrijven die alle klinkers uit een gegeven string kan verwijderen.

```
ghci> fst $ runState (doeDeKlinkersWeg "Dit willen we vandaag over Haskell horen.") []
"Dt wlln w vndg vr Hskll hrn."

ghci> snd $ runState (doeDeKlinkersWeg "Dit willen we vandaag over Haskell horen.") []
[M 'o',M 'o',M 'i',M 'i',M 'e',M 'e',M 'e',M 'e',M 'e',M 'a',M 'a',M 'a',M 'a']
```

[2] Deze oefening is facultatief indien je 2.3.2 maakt

3 Theorie $(\frac{4}{14})$

1. Wat doen de functies `p1/2`, `p2/2` en `p3/3`? Leg het verschil in uitvoering uit.

```
p1([], []).
p1([X|Xs], R) :-
  p1(Xs, Rs),
  append(Rs, [X],R).

p2(L, R) :-
  p2_2(L, R-[]).
p2_2([], X-X).
p2_2([H|T], R-X) :-
  p2_2(T, R-[H|X]).

p3(L, R) :-
  p3_2(L, [], R).
p3_2([], R, R).
p3_2([H|T], L, R) :-
  p3_2(T, [H|L], R).
```

2. Bewijs dat voor de Monad† instantie van `Tree` geldt dat $join \circ join \equiv join \circ fmap\, join$.

3. [3] Wat is het verschil in uitvoeringssnelheid tussen onderstaande reeksen commando's in Haskell?

   Reeks 1:

```
ghci> fst $ runState rl (R 17)
[17,81,89,185,52,48,86]
ghci> snd $ runState rl (R 17)
R 5741632773585689292
```

   Reeks 2:

```
ghci> runState rl (R 17)
([17,81,89,185,52,48,86],R 5741632773585689292)
```

[3] Onderscheidingsvraag

Bijlage

```
import Control.Monad
import Data.Bits
import Data.Int

-- === Code paragraaf "7.4.3 The State Monad" in "bookHaskellTS.pdf" ===

type SP s a = s -> (a, s)

data State s a = State (SP s a)

bindSP :: SP s a -> (a -> SP s b) -> SP s b
bindSP m f = \s0 ->
 ( let (x, s1) = m s0
    in f x s1
 )

pureSP :: a -> SP s a
pureSP x = \s -> (x, s)

instance Functor (State s) where
fmap = liftM

instance Applicative (State s) where
  pure x = State (pureSP x)
  (<*>) = ap

instance Monad (State s) where
  m >>= f = State (bindSP (runState m) (runState . f))

runState :: State s a -> SP s a
runState (State m) = m

get :: State s s
get = State (\s -> (s, s))

put :: s -> State s ()
put s' = State (\s -> ((), s'))

modify :: (s -> (a, s)) -> State s a
modify f = State f

-- ======================================================
-- Eigen implementatie van een random-generator
-- zie System.Random voor een Haskell standaard

class RandomGen g where
next :: g -> (Int, g)

-- random number generator (Marsaglia, George (July 2003). "Xorshift RNGs")
xsl x v = x `xor` (shiftL x v)

xsr x v = x `xor` (shiftR x v)

xorShift x = xsl (xsr (xsl x 13) 7) 17
```

```
data R = R Int deriving (Show)

instance RandomGen R where
next (R x) = (mod x 256, R (xorShift x))

-- --------------------------------------
randomList :: (RandomGen g) => Integer -> State g [Int]
randomList n
  | n > 0 = do
     x <- modify next
     xs <- randomList (n - 1)
     return (x : xs)
  | otherwise = return []

{-
-- voorbeeld: rl is een State
-- runState rl (R 17) laat deze lopen met initiele waarde 17 voor de randomgenerator
-- het resultaat is een koppel bestaande uit de lijst van random getallen en de nieuwe
-- toestand van de randomgenerator.

ghci> rl = randomList 7
ghci> fst $ runState rl (R 17)
[17,81,89,185,52,48,86]
ghci> snd $ runState rl (R 17)
R 5741632773585689292
ghci>
-}
```