Examen Declaratieve Talen.

Patrick De Causmaecker

17/6/2021, 8:30 - 11:30

Toegelaten documentatie is `https://www.swi-prolog.org`, `https://hoogle.haskell.org`, samen met de cursus en de slides. Zend op het einde je antwoorden naar `patrick.decausmaecker@kuleuven.be`.

1 Prolog: de lus in de rij van Collatz en aanverwante $(\frac{5}{14})$

1.1 Tot aan de lus

Schrijf het predicaat collatz/2, $collatz(+X, -CR)$ slaagt als X met een natuurlijk getal is geünificeerd en CR met de rij natuurlijke getallen beginnende bij X en eindigende bij het eerste element dat een tweede keer voorkomt. Het is nog altijd een veronderstelling dat dit voor de Collatz rijen steeds een getal van de lus $1, 4, 2, 1$ zal zijn, maar deze veronderstelling gebruiken we hier niet. De opvolger van een natuurlijk getal Y in de rij van Collatz is gegeven door Z in:

$$Y \text{is even} \Rightarrow Z = Y/2$$
$$Y \text{is oneven} \Rightarrow Z = 3 * Y + 1$$

Een voorbeeld:

```
?- collatz(7,T),print(T).
[7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1,4]
T = [7, 22, 11, 34, 17, 52, 26, 13, 40|...].
?- collatz(2,T),print(T).
[2,1,4,2]
T = [2, 1, 4, 2].
```

1.2 Veralgemening tot Collatz-achtige rijen

Schijf het predicaat collatzAchtig/3 waarbij $collatzAchtig(P, X, T)$ slaagt als $P/2$ slaagt voor $P(+Y, -Z)$ met Z het getal dat ontstaat door een stap te nemen vanuit Y volgens een regel gelijkaardig aan de Collatz-regel, en T de rij is die begint bij X en stopt bij het eerste herhaalde getal.

```
?- modulo47Step(10,X).
X = 11.
?- modulo47Step(46,X).
X = 47.
?- modulo47Step(47,X).
X = 1.
?- collatzAchtig(modulo47Step,10,L),print(L).
[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,
36,37,38,39,40,41,42,43,44,45,46,47,1,2,3,4,5,6,7,8,9,10]
L = [10, 11, 12, 13, 14, 15, 16, 17, 18|...].
?- collatzAchtig(modulo47Step,100,L),print(L).
[100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,
119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,
138,139,140,141,95,96,97,98,99,100]
L = [100, 101, 102, 103, 104, 105, 106, 107, 108|...].
```

1.3 Extra:een verschil dat een verschil maakt

Verbeter (lichtelijk) de efficiëntie door:

- Gebruik van een vorm van memoïzatie om het herhaaldelijk berekenen van dezelfde rijen te vermijden.
- Gebruik van een predicaat $present(+X, ?Ver)$ waarbij Ver unificeert met het verschil van twee lijsten $L - T$ en dat slaagt als X voorkomt in de lijst L en niet in de lijst T. Bijvoorbeeld:

```
?- present(10,T-T).
false.
?- present(10,[1,2,3|R] - R).
false.
?- present(10,[1,2,3,10|R] - R).
true.
?- present(10,[1,2,3,10] - [10]).
false.
?- collatzLike(modulo47Step,100,T-T),print(T).
[100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,
119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,
138,139,140,141,95,96,97,98,99,100]
T = [100, 101, 102, 103, 104, 105, 106, 107, 108|...].
?- collatzLike(collatzStep,100,T-T),print(T).
[100,50,25,76,38,19,58,29,88,44,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1,4]
T = [100, 50, 25, 76, 38, 19, 58, 29, 88|...].
```

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

Note Deze oefening gebruikt Map uit de module Data en het begrip State Monad zoals dat in de cursus werd gedefinieerd. Het staat je vrij om je eigen benadering en implementatie van de State Monad te gebruiken in de Haskell programmeeromgeving, maar zie 4 voor een mogelijke implementatie.

2.1 Collatz wordt herinnerd.

Schrijf een functie *mcoll* met als parameters

1. Een natuurlijk getal x, het startpunt van de Collatz rij.
2. Een voorstelling van het geheugen met vroegere resultaten.

Het resultaat van *mcoll* is het aantal stappen tot de waarde 1 voor de eerste keer bereikt wordt. De voorstelling van het geheugen (s) kan een Map zijn ($Data.Map$) of (als dat niet lukt) gewoon een lijst van koppels of een andere oplossing. Belangrijk is dat dit werkt als een dictionary die argumenten met functiewaarden verbindt. Het resultaat van *mcoll* is een koppel (v, sn) waarbij v de lengte van de Collatz rij voorstelt en sn de nieuwe toestand van het geheugen is. De functie *mcoll* controleert eerst of de gevraagde lengte in het geheugen (s) kan gevonden worden. Indien niet, dan wordt deze functiewaarde berekend en in de nieuwe toestand van het geheugen (sn) opgeslagen. Hieronder een voorbeeld van uitvoering [1]

```
*Main> mcoll 10 M.empty
(6,fromList [(1,0),(2,1),(4,2),(5,5),(8,3),(10,6),(16,4)])
*Main> mcoll 7 M.empty
(16,fromList [(1,0),(2,1),(4,2),(5,5),(7,16),(8,3),(10,6),(11,14),(13,9),(16,4),(17,12),(20,7),
(22,15),(26,10),(34,13),(40,8),(52,11)])
*Main> (v10,newmemory) = mcoll 10 M.empty
*Main> newmemory
```

[1] $M.empty$, $M.fromList$ en $fromList$ verwijzen expliciet naar Map. Dit kan er anders uitzien als je voor een andere oplossingsmethode kiest.

```
fromList [(1,0),(2,1),(4,2),(5,5),(8,3),(10,6),(16,4)]
*Main> v10
6
*Main> (v7,after7) = mcoll 7 newmemory
*Main> after7
fromList [(1,0),(2,1),(4,2),(5,5),(7,16),(8,3),(10,6),(11,14),(13,9),(16,4),(17,12),(20,7),
(22,15),(26,10),(34,13),(40,8),(52,11)]
*Main> v7
16
```

2.2 De functie *memoColl* maakt hier een State Monad van.

Gebruik *mcoll* om de functie *memoColl* te definiëren die een *State* (*Map Int Int*) *Int* (of equivalent als je een andere voorstelling van het geheugen gebruikt dan Map) teruggeeft. Uitvoering van deze monad (door runState) geeft hetzelfde resultaat als uitvoering van *mcoll*.

```
*Main> (runState (memoColl 7)) M.empty
(16,fromList [(1,0),(2,1),(4,2),(5,5),(7,16),(8,3),(10,6),(11,14),(13,9),(16,4),(17,12),(20,7),
(22,15),(26,10),(34,13),(40,8),(52,11)])
*Main> (v10,monomem) = (runState (memoColl 10)) M.empty
*Main> monomem
fromList [(1,0),(2,1),(4,2),(5,5),(8,3),(10,6),(16,4)]
*Main> v10
6
*Main> (v7,monomemna7) = (runState (memoColl 7)) monomem
*Main> monomemna7
fromList [(1,0),(2,1),(4,2),(5,5),(7,16),(8,3),(10,6),(11,14),(13,9),(16,4),(17,12),(20,7),
(22,15),(26,10),(34,13),(40,8),(52,11)]
*Main> v7
16
```

2.3 De Collatz-lengte groeit langzaam.

Gebruik *memoColl* en do-constructies om een functie *largestAbove* te definiëren die een lijst genereert van stijgende groottes van de lengtes die door *memoColl* gegenereerd worden. *largestAbove* neemt twee argumenten: b is de minimale grootte en c is het eerste getal waarvoor een Collatz-lengte berekend wordt. Het resultaat is een State Monad (welke?).

```
*Main> (l,m) = (runState (largestAbove 0 1)) M.empty
*Main> take 10 l
[(1,0),(2,1),(3,7),(6,8),(7,16),(9,19),(18,20),(25,23),(27,111),(54,112)]
*Main> takeWhile (\x -> (snd x) < 120)  l
[(1,0),(2,1),(3,7),(6,8),(7,16),(9,19),(18,20),(25,23),(27,111),(54,112),(73,115),(97,118)]
*Main>
*Main> (l500,m500) = (runState (largestAbove 500 500000)) M.empty
*Main> take 5 l500
[(626331,508),(837799,524),(1117065,527),(1501353,530),(1723519,556)]
*Main>
```

3 Theorie

1. Leg de resultaten uit van de onderstaande dialoog in swipl. ($\frac{2}{14}$)

   ```
   ?- X == Y.
   false.
   ?- X = 10,Y = 1+9,X==Y.
   ```

```
false.
?- Y = 10,X is Y + 0,X == Y.
Y = X, X = 10.
?- Y = 10,X = Y + 0,X == Y.
false.
```

2. Bewijs dat de lijst monad aan de tweede monad wet voldoet zoals gedefinieerd in sectie 7.3 van de cursus: $(\frac{2}{14})$

$$join \circ fmap\ unit \equiv id$$

4 Hints

```
import qualified Data.Map as M
import Control.Monad
{--
* Our State Monad
--}
data State s a = State (s -> (a,s))
{--
* Functor and Applicative defaults for Monad instance
--}
instance Functor (State s) where
  fmap fab ma = do {a <- ma;return (fab a)}
instance Applicative (State s) where
  pure a = do {return a}
  mfab <*> ma = do {fab <- mfab;a <- ma; return (fab a)}
{--
* The State Monad as defined in 'Introduction to Haskell' by Tom Schrijvers
--}
instance Monad (State s) where
  return x = State (\s -> (x,s))
  m >>= f = State (\s -> let (x,s1) = runState m s
                         in runState (f x) s1)
{--
* runState removes the 'State'  constructor
* result is a function s -> (a,s) from a State Monad
--}
runState::State s a -> s -> (a,s)
runState (State m) = m
{--
* modify adds the 'State' constructor
* to a function s -> (a,s) to obtain a State Monad
--}
modify::(s -> (a,s)) -> State s a
modify m = State m
```