Haskell Live

[08] CGI mit Haskell und Aufgabenblatt 5

Bong Min Kim
e0327177@student.tuwien.ac.at

Christoph Spörk
christoph.spoerk@inode.at

Florian Hassanen
florian.hassanen@googlemail.com

Bernhard Urban
lewurm@gmail.com

26. November 2010

```haskell
import Data.List
import Data.Char
```

CGI mit Haskell

Im Archiv `08cgi.zip` befinden sich die entsprechenden Dateien.

Aufgabenblatt 5

```haskell
type Cost = Integer
type Vertex = Integer
type MaxVertexNo = Integer
type Edge = (Vertex, Cost, Vertex)
type Row = [Integer]
data ALgraph = ALg [(Vertex, [(Vertex, Cost)])] deriving (Eq, Show)
data AMgraph = AMg [Row] deriving (Eq, Show)
data ELgraph = ELg MaxVertexNo [Edge] deriving (Eq, Show)
type Inp = (MaxVertexNo, [(Vertex, Vertex, Cost)])
```

```
  -- 1.
isValid :: Inp → Bool
isValid (maxno, x) = isValid' x [ ] maxno
isValid' :: [(Vertex, Vertex, Cost)] → [(Vertex, Vertex)] → MaxVertexNo → Bool
isValid' [ ] exists maxno = and [v ⩽ maxno | v ← allvertices]
  where
  allvertices :: [Vertex]
  allvertices = nub $ (λ(x, y) → x ++ y) $ unzip exists
isValid' ((va, ve, cost) : xs) exists maxno
   | (va, ve) ∈ exists = False
   | cost < 0 = False
   | otherwise = isValid' xs ((va, ve) : exists) maxno
  -- 2a.
inp2el :: Inp → ELgraph
inp2el (maxno, list) = ELg maxno (map (λ(x, y, z) → (x, z, y)) list)
  -- 2b.
al2am :: ALgraph → AMgraph
al2am = el2am ∘ al2el
  -- 2c.
al2el :: ALgraph → ELgraph
al2el (ALg alg) = ELg maxno $ al2el' alg
  where
  maxno :: MaxVertexNo
  maxno = fromIntegral $ (length alg) − 1
al2el' :: [(Vertex, [(Vertex, Cost)])] → [Edge]
al2el' [ ] = [ ]
al2el' ((v, neigh) : xs) = (map (λ(vt, c) → (v, c, vt)) neigh) ++ al2el' xs
  -- 2d.
am2al :: AMgraph → ALgraph
am2al = el2al ∘ am2el
```

```
  -- 2e.
am2el :: AMgraph → ELgraph
am2el (AMg rows) = ELg len $ concat
  [
    [
    (vf, cost, vt)
     | vt ← [0 .. len]
    , let cost = row !! (fromIntegral vt)
    , cost > 0
    ]
   | vf ← [0 .. len]
  , let row = rows !! (fromIntegral vf)
  ]
  where
  len :: Integer
  len = (mylen rows) − 1
  mylen :: [[Integer]] → Integer
  mylen [] = 0
  mylen (_ : xs) = 1 + mylen xs
  -- 2f.
el2al :: ELgraph → ALgraph
el2al (ELg maxno edges) = ALg [(v, findneighbors v) | v ← [0 .. maxno]]
  where
  findneighbors :: Vertex → [(Vertex, Cost)]
  findneighbors targetvertex = [(vert, cost) | (target, cost, vert) ← edges, target ≡ targetvertex]
  -- 2g.
el2am :: ELgraph → AMgraph
el2am (ELg maxno edges) = AMg [cost2neigh v | v ← [0 .. maxno]]
  where
  cost2neigh :: Vertex → Row
  cost2neigh x = [find v | v ← [0 .. maxno]]
    where
    find :: Vertex → Cost
    find vertex = case (x, vertex) `lookup` [((vf, vt), c) | (vf, c, vt) ← edges] of
       Just kosten → kosten
       Nothing → 0
  -- 3a.
isNeighbourOf :: ELgraph → Vertex → Vertex → Bool
isNeighbourOf elg vf vt = vt ∈ allNeighboursOf elg vf
  -- 3b.
allNeighboursOf :: ELgraph → Vertex → [Vertex]
allNeighboursOf elg v = [x | (x, _) ← allNeighboursOf′ elg v]
allNeighboursOf′ :: ELgraph → Vertex → [(Vertex, Cost)]
allNeighboursOf′ (ELg maxno edges) vf = sort
  [(vt, c)
   | (v, c, vt) ← edges
  , v ≡ vf
  ]
  -- 3c.
numberOfEdges :: AMgraph → Integer
numberOfEdges amg = fromIntegral $ length edges
  where (ELg max edges) = am2el amg
  -- 3d.
isOnCycle :: ALgraph → Vertex → Cost → Bool
isOnCycle (ALg al) vt cost = [] ≢
  [c
   | (Just c) ← findCycleCosts al vt vt []
  , (c ⩽ cost ∧ c > 0)
  ]
findCycleCosts :: [(Vertex, [(Vertex, Cost)])] → Vertex → Vertex → [Vertex] → [Maybe Integer]
findCycleCosts al start current tabu
   | (start ≡ current) ∧ (tabu ≢ []) = [Just 0]
   | current ∈ tabu = [Nothing]
   | otherwise =
     [Just (c + cnext)
      | (vt, c) ← (allNeighboursOf′ (al2el (ALg al)) current)
     , Just cnext ←
        [cn
         | cn ← findCycleCosts al start vt (current : tabu)
        , cn ≢ Nothing
        ]
     ]
```