# Haskell Answers 7: Tuples

Antoni Diller

4 August 2011

(1) An association list is a list of 2-tuples. For example, `[("temp", 34), ("height", 80), ("weight", 180), ("depth", 7)]`. Define a function *domain* :: *Eq* $a \Rightarrow [(a, b)] \rightarrow [a]$ which takes an association list and returns the list of all those things that occur in the first component of each tuple. Make sure that the value of *domain* does not contain any duplicates.

```
nub :: Eq a => [a] -> [a]
nub = nubBy (==)

nubBy :: (a -> a -> Bool) -> [a] -> [a]
nubBy eq []     =  []
nubBy eq (x:xs) =  x : nubBy eq (filter (\y -> not (eq x y)) xs)

domain :: Eq a => [(a, b)] -> [a]
domain ass = nub [x | (x, _) <- ass]
```

(2) Define a function *range* :: *Eq* $a \Rightarrow [(a, b)] \rightarrow [b]$ which takes an association list and returns the list of all those things that occur in the second component of each tuple. Make sure that the value of *range* does not contain any duplicates.

```
range1 :: Eq b => [(a, b)] -> [b]
range1 ass = nub [y | (_, y) <- ass]
```

(3) Define a function *compose* :: $[(a, b)] \rightarrow [(b, c)] \rightarrow [(a, c)]$ such that a tuple $(x, z)$ is in the list returned as the value of the function *compose ass1 ass2* iff $(x, y)$ is in *ass1* and $(y, z)$ is in *ass2*. For example, `compose [(1, 2), (7, 11)] [(2, 3), (11, 14)]` is `[(1, 3), (7, 14)]`.

```
compose :: [(a, b)] -> [(b, c)] -> [(a, c)]
compose ass1 ass2 = [(x, z) | (x, y) <- ass1, (y, z) <- ass2]
```

(4) Define a function *inverse* :: $[(a, b)] \to [(b, a)]$ such that a tuple $(y, x)$ is in *inverse ass* iff $(x, y)$ is in *ass*.

```
inverse :: [(a, b)] -> [(b, a)]
inverse ass = [(y, x) | (x, y) <- ass]
```

(5) A *homogeneous* association list is one whose tuples contain elements belonging to the same type. Define a function *reflexive* :: *Eq a* $\Rightarrow [(a, a)] \to$ *Bool* which tests to see if a homogeneous association list is reflexive, that is to say, if either $(x, y)$ or $(y, x)$ is in *ass*, then $(x, x)$ is also in *ass*.

```
member x [] = False
member x (y:ys)
  | x == y = True
  | otherwise = member x ys
reflexive :: Eq a => [(a, a)] -> Bool
reflexive ass =
  and [member (x, x) ass | x <- (domain ass) ++ (range1 ass)]
```

(6) Define a function *symmetric* :: *Eq a* $\Rightarrow [(a, a)] \to$ *Bool* which tests to see if a homogeneous association list is symmetric, that is to say, if $(x, y)$ is in *ass*, then so is $(y, x)$.

```
symmetric :: Eq a => [(a, a)] -> Bool
symmetric ass = and [member (y, x) ass | (x, y) <- ass]
```

(7) Define a function *transitive* :: *Eq a* $\Rightarrow [(a, a)] \to$ *Bool* which tests to see if a homogeneous association list *ass* is transitive, that is to say, if $(x, y)$ and $(y, z)$ are in *ass*, then so is $(x, z)$.

```
transitive :: Eq a => [(a, a)] -> Bool
transitive ass =
  and [member (x, z) ass | (x, y) <- ass, (y, z) <- ass]
```

(8) Define a function *closure* :: $[(a, a)] \to [(a, a)]$ which takes an arbitrary association list *ass* and produces the reflexive, transitive closure of *ass*, that is to say, if $(x, y)$ and $(y, z)$ are in *ass*, then $(x, y)$, $(y, z)$ and $(x, z)$ are all in *closure ass*.

(9) Define the function *pairs* such that *pairs i* is the list of all distinct pairs of integers

$(x, y)$ such that $1 \leq x, y \leq i$. For example,

$$pairs\ 3 = [(1,2),(1,3),(2,1),(2,3),(3,1),(3,2)].$$

```
pairs :: Integral a => a -> [(a,a)]
pairs i = [ (x,y) | x <- [1..i], y <- [1..i], x /= y ]
```

(10) Using the function *zip* define the infinite list *factlist* of factorials.

(11) A curried function $f$ of $n$ Boolean arguments is called tautologous if it returns *True* for every one of the $2^n$ possible combinations of Boolean arguments. Write a function *taut* so that *taut* $n$ $f$ is *True* is and only if $f$ is tautologous.