

6. Suponha que tem a relação $db_1 \in (Dat \times Jog^*)^*$ de todos os jogos que se efectuaram numa dada competição, organizados por data. Seja ainda $db_2 \in (Jog \times Atl^*)^*$ a indicação, para cada jogo, dos atletas que nele participaram.

Um comentador desportivo pede-lhe que derive de db_1 e de db_2 a relação, ordenada por nome, das datas em que cada atleta jogou, datas essas também ordenadas:

$$f : (Dat \times Jog^*)^* \rightarrow (Jog \times Atl^*)^* \rightarrow (Atl \times Dat^*)^*$$

$$f \ db_1 \ db_2 = \dots$$

Mostre que f pode ser escrita numa só linha usando os combinadores $f \cdot g$, $f \times g$, etc que até agora estudou, desde que tenha à sua disposição a seguinte biblioteca de funções genéricas:

- $sort : A^* \rightarrow A^*$
— ordena listas de A segundo uma ordem previamente assumida (sobre A)
- $collect : (A \times B)^* \rightarrow (A \times B^*)^*$
— agrupa uma sequência de pares segundo os respectivos primeiros elementos, e.g.
 $collect [(1, 2), (5, 6), (1, 3)] = [(1, [2, 3]), (5, [6])]$
- $discollect : (A \times B^*)^* \rightarrow (A \times B)^*$
— inversa da anterior
- $converse : (A \times B)^* \rightarrow (B \times A)^*$
— troca os elementos de cada par entre si
- $comp : (A \times B)^* \rightarrow (B \times C)^* \rightarrow (A \times C)^*$
— encadeia as sequências de entrada de acordo com os elementos em comum (de tipo B).

Use um ininterpretador de Haskell para verificar que o seu plano para f está bem tipificado, *sem* implementar as funções dadas.

Resolução

In [172...]

```
sort :: [a] -> [a]
sort = undefined

-- type checking

:t sort
```

sort :: forall a. [a] -> [a]

In [173...]

```
collect :: [(a,b)] -> [(a,[b])]
collect = undefined

-- type checking

:t collect
```

collect :: forall a b. [(a, b)] -> [(a, [b])]

In [174...]

```
discollect :: [(a,[b])] -> [(a,b)]
discollect = undefined

-- type checking

:t discollect
```

discollect :: forall a b. [(a, [b])] -> [(a, b)]

In [175...]

```
converse :: [(a,b)] -> [(b,a)]
converse = undefined

-- type checking

:t converse
```

converse :: forall a b. [(a, b)] -> [(b, a)]

In [176...]

```
comp :: [(a,b)] -> [(b,c)] -> [(a,c)]
comp = undefined

-- type checking

:t comp
```

comp :: forall a b c. [(a, b)] -> [(b, c)] -> [(a, c)]

In [177...]

```
(f >< g) (a,b) = (f a , g b)

-- type checking

:t (><)
```

(><) :: forall t1 a t2 b. (t1 -> a) -> (t2 -> b) -> (t1, t2) -> (a, b)

In [178...]

```
--- data types

data Jog
data Dat
data Atl
```

In [179...]

```
:i Jog
```

```
type Jog :: *
data Jog
    -- Defined at <interactive>:1:1
```

In [180...]

```
:i Dat
```

```
type Dat :: *
data Dat
    -- Defined at <interactive>:2:1
```

In [194...]

```
:i Atl
```

```
type Atl :: *
data Atl
    -- Defined at <interactive>:3:1
```

In [182...]
`f :: [(Dat, [Jog])] -> [(Jog, [Atl])] -> [(Atl, [Dat])]`
`f = undefined`

-- type checking

:t f

`f :: [(Dat, [Jog])] -> [(Jog, [Atl])] -> [(Atl, [Dat])]`

In [183...]
`db1 :: [(Dat, [Jog])]`
`db1=undefined`

-- type checking

:t db1

`db1 :: [(Dat, [Jog])]`

In [184...]
`db2 :: [(Jog, [Atl])]`
`db2 = undefined`

-- type checking

:t db2

`db2 :: [(Jog, [Atl])]`

In [185...]
-- type checking (step 0)
:t (db1,db2)

`(db1,db2) ::([(Dat, [Jog])], [(Jog, [Atl])])`

In [186...]
-- type checking (step 1)
:t (discollect >< discollect) \$ (db1,db2)

`(discollect >< discollect) $ (db1,db2) ::([(Dat, Jog)], [(Jog, Atl)])`

In [187...]
-- type checking (step 2)
:t uncurry comp . (discollect >< discollect) \$ (db1,db2)

`uncurry comp . (discollect >< discollect) $ (db1,db2) :: [(Dat, Atl)]`

In [188...]

```
-- type checking (step 3)

:t converse . uncurry comp . (discollect >< discollect) $ (db1,db2)

converse . uncurry comp . (discollect >< discollect) $ (db1,db2) :: [(Atl, Dat)]
```

In [189...]

```
-- type checking (step 4)

:t collect . converse . uncurry comp . (discollect >< discollect) $ (db1,db2)

collect . converse . uncurry comp . (discollect >< discollect) $ (db1,db2) :: [(Atl, [Dat])]
```

In [190...]

```
-- type checking (step 5)

:t map (id >< sort) . collect . converse . uncurry comp . (discollect >< discollec

map (id >< sort) . collect . converse . uncurry comp . (discollect >< discollect) $ (db1,db2) :: [(Atl, [Dat])]
```

In [196...]

```
-- type checking (step 6)

:t sort . map (id >< sort) . collect . converse . uncurry comp . (discollect >< discollec

sort . map (id >< sort) . collect . converse . uncurry comp . (discollect >< discollect) $ (db1,db2) :: [(Atl, [Dat])]
```

In [193...]

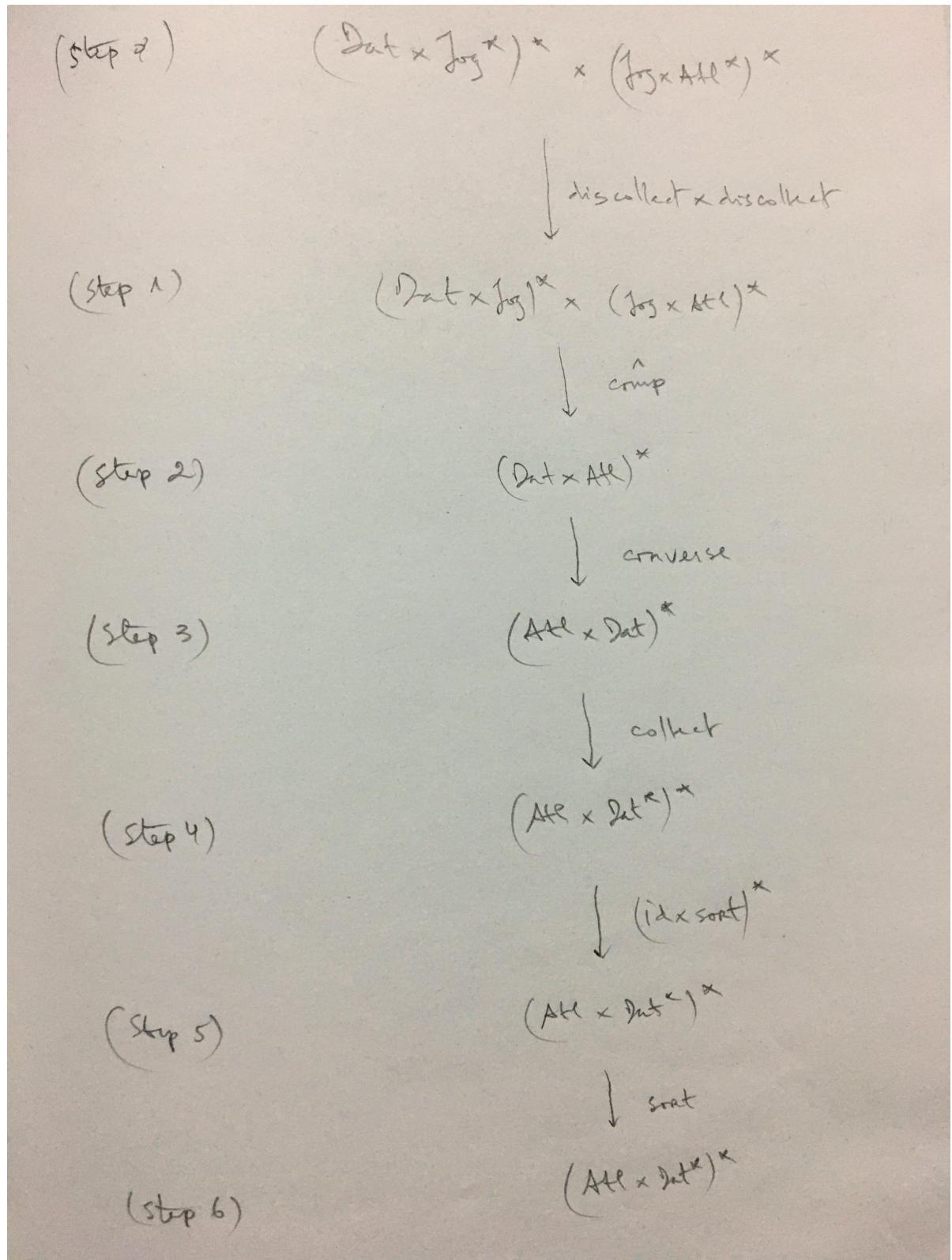
```
f :: [(Dat,[Jog])] -> [(Jog,[Atl])] -> [(Atl,[Dat])]

f db1 db2 = sort . map (id >< sort) . collect . converse . uncurry comp . (di

-- type checking

:t f

f :: [(Dat, [Jog])] -> [(Jog, [Atl])] -> [(Atl, [Dat])]
```



In []: