PARADIGMES I LLENGUATGES DE PROGRAMACIÓ

PRÀCTICA DE HASKELL

Validador de jugades d'escacs

Jose Garvin Victoria
Mauro Balestra Sastriques

Enginyeria informàtica

Curs 2018/19

Índex

1 Introducció i abast **2**
1.1 Introducció i objectiu 2
1.2 Abast de la pràctica 2

2 Resolució del problema **2**
2.1 Definicions de tipus 3
2.2 Funcions a destacar 4

3 Execució i proves **6**
3.1 Com executar 6
3.2 Jocs de proves 6

4 Observacions **9**

5 Bibliografía **10**

1 Introducció i abast

1.1 Introducció i objectiu

L'objectiu d'aquesta pràctica és l'implementació d'un validador de jugades d'escacs en el llenguatge de programació **Haskell**.
El funcionament principal d'aquest programa és divideix en els següents blocs:

- Interacció amb l'usuari per demanar el fitxer de text amb les jugades de la partida d'escacs.
- Lectura del fitxer de text, on assumim que el format és correcte i que la notació emprada és la **algebraica estesa**.
- Per cada jugada llegida del fitxer, es generen unes jugades on es validarán la seva correctesa, amb indicació de captures, escacs i escac i mat.
- Per pantalla es mostraràn els estats de la partida en curs, fins arribar al final, on trobarem un missatge indicant el guanyador, si n'hi ha, empat, o bé un missatge d'error indicant quina ha estat la jugada invàlida.

1.2 Abast de la pràctica

Per problemes de sintaxi i comprensió del paradigma de programació, l'abast ha estat fins a la nota de 8:

Compliments i notes:
- Fins a un 8. Validació de notació correcta (amb indicació de captures, escacs i escac i mat), de moviments correctes (sense controlar captura al pas, coronació i enrocs), detecció d'escac i mat, representació textual i interacció amb l'usuari correcte

Cal dir que encara que el grup no ha pogut assolir l'objectiu de l'abast de puntuació de 10, s'ha pogut acurar el disseny del programa, emprant mòduls (no programar tot en un sol fitxer), llistes per comprhensió, i funcions d'ordre superior a la majoria de mètodes **importants** del programa.

Gairebé no s'han hagut d'importar mòduls externs de la distribució GHC, ja que creiem que era millor aprendre a implementar aquestes funcions que ens oferia aquesta distribució pel nostre compte, per tal d'adquirir els coneixements adequats d'aquest paradigma.

2 Resolució del problema

Com a part de resolució del problema, el més important, ha estat pensar molt bé quins serien els tipus a implementar del validador d'escacs. En aquest apartat ens centrarem en explicar quins han estat els tipus de dades que hem utilitzat i que creiem que cal destacar i quines són les funcions del nostre programa que considerem més importants, i dignes de comentar.

Dividirem aquesta secció en 3 apartats; en un d'ells parlarem sobre els tipus utilitzats, en un altre comentarem les funcions que creiem més importants i que pensem que hem d'explicar clarament perque s'entengui millor el codi i per últim, un apartat on comentarem altres aspectes importants del nostre codi que pensem que són convenients d'explicat amb detall.

2.1 Definicions de tipus

En aquest apartat llistarem quins són el tipus que hem definit per a poder desenvolupar la pràctica i explicarem la finalitat per a la qual s'ha decidit definir cadascun dels tipus:

Al fitxer **"Jugada.hs"** hem definit els següents tipus:

- data Jugada = Jugada tipusPeca :: Peca, origen :: Posicio, desti :: Posicio, accio :: Accio | Acabada deriving (Eq): Aquest tipus l'utilitzem per representar jugades al joc d'escacs. Una jugada la representem a partir d'una peça, una posició origen, una posició destí, i una acció a realitzar. Aquest tipus deriva de la classe *Equals*, això ens permetra poder comparar taulers.
- data Accio = Matar | Escac | EscacIMat | Res deriving (Eq): Aquest tipus l'utilitzem per representar jugades al joc d'escacs. Una jugada la representem a partir d'una peça, una posició origen, una posició destí, i una acció a realitzar. Aquest tipus deriva de la classe *Equals*, això ens permetra poder comparar accions.

Al fitxer **"Partida.hs"** hem definit els següents tipus:

- data Partida = Partida Tauler Torn deriving (Eq): Aquest tipus l'utilitzem per representar partides al joc d'escacs. Una partida la representem a partir d'un tauler i una acció.

Al fitxer **"Peca.hs"** hem definit els següents tipus:

- data Peca = Peca tipus::TipusPeca, color::ColorPeca | Buida deriving (Eq): Aquest tipus l'utilitzem per representar peces al joc d'escacs. Ens ha estat necessari definir peces de tipus *Buida*. Si una peça no es *Buida*, estarà representada a partir d'un tipus de peça i un color (blanc i negre). Aquest tipus deriva de la classe *Equals*, això ens permetra poder comparar peces durant el desenvolupament d'una partida.
- data TipusPeca = Peo | Cavall | Alfil | Torre | Dama | Rei deriving (Read): Aquest tipus l'utilitzem per representar el tipus d'una al joc d'escacs. Aquest tipus podrà ser peó, cavall, alfil, torre, dama o rei. Hem fet que aquest tipus derivi de la classe *Read*, això ens permetra poder llegir tipus de peces.
- data ColorPeca = Blanc — Negre — NoColor deriving (Eq, Show, Read): Aquest tipus l'utilitzem per representar el color d'una peça. Ens ha estat necessari definir un color de peça de tipus *NoColor*. Hem fet que aquest tipus derivi de la classe *Read*, *Eq*, *Show* això ens permetra poder llegir colors de peça, comparar-los i mostrar-los.

Al fitxer **"Tauler.hs"** hem definit els següents tipus:

- data Tauler = Tauler LlistaParell deriving (Eq): Aquest tipus l'utilitzem per representar partides el tauler d'escacs. Hem decidit representar el tauler utilitzant un tipus creat per nosaltres, que hem anomenat *LlistaParell*. Es pot veure com s'ha implementat aquest tipus a l'arxiu **"Tauler.hs"**. El nostre tauler deriva de la classe *Eq* això ens permetrà poder comparar taulers.

2.2 Funcions a destacar

En aquest apartat explicarem quin es el comportament de les funcions que creiem que cal destacar per tal d'entendre el funcionament del nostre validador d'escacs. Explicar cadascuna de les funcions faria que aquest document fós massa extens i per tant ens limitarem a explicar les funcions que creiem que són més rellevants.

- Funció *moviment* : Aquesta funció rep una Peça, i una posició, i retorna totes les posicions que pot fer la peça en qüestió en un tauler buit.

```
moviment :: Peca -> Posicio -> [Posicio]
moviment _ pos | not (correcte pos) = error "La posició especificada no es troba dins del taulell"
moviment peca pos
    | tipus peca == Torre = movimentsTorre pos
    | tipus peca == Cavall = movimentsCavall pos
    | tipus peca == Alfil = movimentsAlfil pos
    | tipus peca == Dama = movimentsDama pos
    | tipus peca == Rei = movimentsRei pos
    | tipus peca == Peo = movimentsPeo pos (color peca)
    | otherwise = error ("Moviments no definits")
```

- Funció *movimentsRei* : Aquesta funció es específica pels moviments del Rei, es realitza un filter sobre tots els moviments de la Dama, filtrant tots els moviments que siguin més grans a 1. Per tant es queda amb totes les posicions disponibles amb distància ¡= 1.

```
-- Funció per generar les posicions d'un rei tenint en compte la posició en es troba.
-- * Paràmetre 1: Posició en la que es troba la peça.
-- ** Retorn: Llista de posicions (moviments) possibles per a la peça tenint en compte la posició on es troba.
-- *** Aclariments
-- Aprofitem els moviments de la dama limitant el deplaçament a una casella
movimentsRei :: Posicio -> [Posicio]
movimentsRei pos = filter (\x -> ((abs (fst pos - fst x) <= 1) && (abs (snd pos - snd x) <= 1))) (movimentsDama pos)
```

- Funció *escac* : Donat l'estat actual del tauler, i un color de jugador, i indica si aquell bàndol ha rebut escac.

```
escac :: Tauler -> ColorPeca -> Bool
escac t c =
    let posRei = (filter (\x -> (tipus (fst x)) == Rei) (obtenirPecesPerColor t c)) !! 0 -- Cerquem el rei del color donat al tauler
        pecesContrari = obtenirPecesPerColor t (contrari c) -- Obtenim les peces del color contrari
        escac = or (map (\x -> potMatar x posRei t) pecesContrari)
    in escac
```

- Funció *escacIMat* : Donat l'estat actual del tauler, i un color de jugador, i indica si aquell bàndol ha rebut escac i mat, i per tant han perdut la partida.

```haskell
escacIMat :: Tauler -> ColorPeca -> Bool
escacIMat t c =
    let posRei = (filter (\x -> (tipus (fst x)) == Rei) (obtenirPecesPerColor t c)) !! 0 -- Cerquem el rei del color donat al tauler
        movimentsRei = filter (\x -> (buscarPeca x t) == Buida) (moviment (fst posRei) (snd posRei))  -- Dels moviments possibles ens quedem els que van a posicion lliures
        mat = if (length movimentsRei) > 0 then and (map (\x -> escac (moure t ((Peca Rei c), snd posRei) x) c) movimentsRei) else False
    in mat
```

- Funció *fesJugada* : Aquesta és una de les funcions més importants, ja que comprova si la jugada que es va a realitzar es correcte, comprovant: La obligació de matar, comprovant que el contingut del fitxer es correspont amb el del tauler actual, indicant captures, escacs i escac i mat. Retorna un nou tauler amb la jugada feta, si es correcte.

```haskell
-- Funció que donat un Tauler i una Jugada ens torna un nou Tauler amb la jugada feta.
-- * Paràmetre 1: Tauler inicial
-- * Paràmetre 2: Tauler amb la jugada aplicada
-- ** Retorn: Tauler amb la jugada feta
fesJugada :: Tauler -> Jugada -> Tauler
fesJugada t jugada | jugada /= Acabada =
    let esJugadaLegal = (jugadaLegal jugada t)
        taulerAmbJugada = (moure t ((tipusPeca jugada), (origen jugada)) (desti jugada))
        nouTauler =
            if obligatMatar t jugada && (accio jugada) == Res
            then error ("No s'ha indicat la captura a la jugada " ++ (show jugada))
            else if (accio jugada) == Matar && ((buscarPeca (desti jugada) t) == Buida)
            then error ("S'ha especificat una captura inexistent a la jugada " ++ (show jugada))
            else if (accio jugada) == Escac && not (escac taulerAmbJugada (contrari (color (tipusPeca jugada))))
            then error ("S'ha indicat un escac inexistent a la jugada " ++ (show jugada))
            else if (accio jugada) == EscacIMat && not (escacIMat taulerAmbJugada (contrari (color (tipusPeca jugada))))
                then error ("S'ha indicat un escac i mat inexistent a la jugada " ++ (show jugada))
                else if esJugadaLegal
                    then taulerAmbJugada
                else error ("La jugada " ++ (show jugada) ++ " Es errònia.")

    in nouTauler
    | otherwise = t
```

- Funció principal *main* : Funció principal d'entrada del programa, es crea la partida, mostrant el tauler inicial, i es van concatenant jugades dins una llista que després el mètode mostrarJoc, mostrarà.

```haskell
main = do
    putStrLn "Indica el nom del fitxer: "
    nomFitxer <- getLine
    llegirContingutFitxer nomFitxer ReadMode (\handle -> do
        contents <- hGetContents handle
        let jugades = lines contents
            partidaInicial = (Partida crearTauler Blanc)
        print (crearTauler)
        mostrarJoc (tractarJugades jugades partidaInicial) jugades )
```

3 Execució i proves

En aquest apartat del document explicarem com cal executar el nostre programa i mostrarem algunes de les sortides que hem obtingut pasant els fitxers de proves que s'ens han facilitat i altres fitxers de proves que hem creat nosaltres.

3.1 Com executar

Per tal d'executar el nostre programa caldrà tenir instal·lat l'intèrpret de Haskell GHCI. Un cop tinguem l'intèrpret instal·lat al nostre sistema i estem ubicats a la ruta on es troba el codi, podrem executar el programa fent servir l'instrucció que es mostra a continuació:

```
$ runhaskell main.hs
```

En executar el programa, haurem d'especificar la ruta a un fitxer d'entrada que haurà de seguir el format especificat en l'enunciat de la pràctica per tal de començar a utilitzar el nostre validador.

3.2 Jocs de proves

A continuació es mostren unes quantes sortides obtingudes al passar al nostre programa verificador de jugades d'escacs.

- **Sortida del test** *pastor.txt* :

```
Indica el nom del fitxer:
tests/pastor.txt
   ==========
8- |tcadract|
7- |pppppppp|
6- |........|
5- |........|
4- |........|
3- |........|
2- |PPPPPPPP|
1- |TCADRACT|
   ==========
    abcdefgh

Tirada 1. Pe2e4 Pe7e5
   ==========
8- |tcadract|
7- |pppp.ppp|
6- |........|
5- |....p...|
4- |....P...|
3- |........|
2- |PPPP.PPP|
```

```
1- |TCADRACT|
   ==========
    abcdefgh
```

```
Tirada 2. Af1c4 Cb8c6
   ==========
8- |t.adract|
7- |pppp.ppp|
6- |..c.....|
5- |....p...|
4- |..A.P...|
3- |........|
2- |PPPP.PPP|
1- |TCADR.CT|
   ==========
    abcdefgh
```

```
Tirada 3. Dd1h5 Cg8f6
   ==========
8- |t.adra.t|
7- |pppp.ppp|
6- |..c..c..|
5- |....p..D|
4- |..A.P...|
3- |........|
2- |PPPP.PPP|
1- |TCA.R.CT|
   ==========
    abcdefgh
```

```
Tirada 4. Dh5xf7++
   ==========
8- |t.adra.t|
7- |pppp.Dpp|
6- |..c..c..|
5- |....p...|
4- |..A.P...|
3- |........|
2- |PPPP.PPP|
1- |TCA.R.CT|
   ==========
    abcdefgh
```

```
Fi de partida, blanques guanyen!!!
```

- **Sortida del test** *pastor.txt* :

```
Indica el nom del fitxer:
tests/obligar_matar.txt
   ==========
8- |tcadract|
7- |pppppppp|
6- |........|
5- |........|
4- |........|
3- |........|
2- |PPPPPPPP|
1- |TCADRACT|
   ==========
    abcdefgh

Tirada 1. Pd2d4 Pe7e5
   ==========
8- |tcadract|
7- |pppp.ppp|
6- |........|
5- |....p...|
4- |...P....|
3- |........|
2- |PPP.PPPP|
1- |TCADRACT|
   ==========
    abcdefgh

Tirada 2. Pd4d5
main.hs: No s'ha indicat la captura a la jugada amb peca (P)
, posicio origen (4,4), posicio desti (4,5) i accio moure.
CallStack (from HasCallStack):
  error, called at ./Jugada.hs:124:18 in main:Jugada
```

4 Observacions

Durant el desenvolupament d'aquesta pràctica hem asimilat molts aspectes que encara no teniem clar de la forma de programar amb Haskell. Tot i que podría semblar una pràctica simple de fer, utilitzant qualsevol altre paradigme de programació, al fer-ho amb Haskell ens hem anat trobant amb problemes que s'han pogut anar solucionant.

Considerem que en general la pràctica no es gaire complexa si només s'aspira a un 8, peró en el nostre cas, no hem pogut complir els altres objectius per falta de temps ja que els problemes esmentats anteriorment han requerit de temps per ser solucionats.

En resum, ens ha agradat fer aquesta pràctica, perquè hem pogut asimilar conceptes sobre el funcionament d'aquest llenguatge de programació que pensàvem que teniem controlats, pero a l'hora de la veritat ens hem adonat que no, ja que programar utilitzant aquest paradigma requereix un canvi de mentalitat bastant gran.

5 Bibliografía

Aprende Haskell: http://aprendehaskell.es/main.html

How work on lists: https://wiki.haskell.org/How_to_work_on_lists

Let vs. where: https://wiki.haskell.org/Let_vs._Where

Files: https://www.glc.us.es/~jalonso/vestigium/i1m2016-manejo-de-ficheros-en-haskell/

Substring: https://rosettacode.org/wiki/Substring#Haskell

Data type: http://aprendehaskell.es/content/ClasesDeTipos.html#introduccion-a-los-tipo