

Haskell Refresher Lecture

Petros Papapanagiotou

pe.p@ed.ac.uk

9 Feb 2012



Haskell refresher!

Haskell

- Purely functional! : *“Everything is a function!”*
- Main topics:
 - Recursion
 - Currying
 - Higher-order functions
 - List Processing functions such as map, filter, foldl, sortBy, etc.
 - The Maybe monad
- For more: <http://www.haskell.org/haskellwiki/Haskell>

A CSP

- Consists of:
 - **X** : a set of **Variables**
 - **D** : a set of **Domains**
 - **C** : a set of **Constraints**

X : a set of Variables

- Variables as Strings:

```
type Var = String
  eg. "X"
```

D : a set of Domains

- Domain as a list of allowable (Int) values for each variable:

```
type Domain = [Int]
  eg. [1,2,3]
```

- D as a list of Domains for each Var:

```
type Domains = [(Var,Domain)]
  eg. [("X",[1,2,3]),("Y",[1,2])]
```

State of a CSP: Assignment

- *"Assignment of values to some or all of the variables" - R&N §6.1*

- Assignment as custom type:

```
newtype AssignedVar = ...
  eg. x=1
type Assignment = [AssignedVar]
  eg. [x=1,y=2]
```

- Manipulate it using functions.

Assignment functions

- `assign ::`
Assignment -> Var -> Int -> Assignment
 - eg. `assign [] "x" 1 → [x=1]`
 - eg. `assign [x=1] "y" 2 → [y=2,x=1]`
 - eg. `assign [x=1] "x" 2 → [x=2]` (Updated!)

Assignment functions

- `assign ::`
Assignment -> Var -> Int -> Assignment
 - eg `assign [] "x" 1 → [x=1]`
 - eg `assign [x=1] "y" 2 → [y=2,x=1]`
 - eg `assign [x=1] "x" 2 → [x=2]` (Updated!)
- **Care!!:** `[y=2,x=1]` is really:
`assign (assign [] "x" 1) "y" 2`

Assignment functions

- `lookup_var ::`
Assignment -> Var -> Maybe Int
 - eg `lookup_var [x=2] "x" → Just 2`
 - eg `lookup_var [x=2] "y" → Nothing`
- `is_unassigned ::`
Assignment -> Var -> Bool
 - eg `is_unassigned [x=2] "x" → False`
 - eg `is_unassigned [x=1] "y" → True`

Relations

- *"Give me a scope and a state and I'll tell you if it's ok!" – Relation*

```
type Relation =  
[Var] -> Assignment -> Bool
```

Relations

- Example: `vars_diff :: Relation`
 - Ensures two variables are different.
 - Scope = "which variables?"
 - Assignment = "what state should I check?"
- Examples:
 - `vars_diff ["x","y"] [x=1,y=2] → True`
 - `vars_diff ["x","y"] [x=1,y=1] → False`
- Care for unassigned variables!!
 - `vars_diff ["x","y"] [x=1] → True (!!)`
 - `vars_diff ["x","y"] [] → True (!!)`

Constraint functions

- `check_constraint :: Constraint -> Assignment -> Bool`
- `check_constraints :: [Constraint] -> Assignment -> Bool`
- `scope :: Constraint -> [Var]`
- `is_constrained :: Constraint -> Var -> Bool`
- `neighbours_of :: Constraint -> Var -> [Var]`

Constraint constructors

- Functions (wrappers) that construct a Constraint from a Relation.
- Already done for you!
- eg.
`vars_diff_constraint :: Var -> Var -> Constraint`
- `vars_diff_constraint "x" "y" -> Constraint`

CSP

- *Bringing it all together...*
- CSPs as a custom type:
`newtype CSP = ...`
- Constructor:
`CSP (String , Domains , [Constraint])`

The BACKTRACK algorithm

```
bt :: CSP -> Maybe Assignment
bt csp = bt_recursion [] csp

bt_recursion :: Assignment -> CSP -> Maybe Assignment
bt_recursion assignment csp =
  if (is_complete csp assignment) then Just assignment
  else find_consistent_value $ domain_of csp var
  where var = get_unassigned_var csp assignment
        find_consistent_value vals =
          case vals of -- recursion over the possible values
                    -- instead of for-each loop
                    [] -> Nothing
                    val:vs ->
                      if (is_consistent_value csp assignment var val)
                      then if (isNothing result)
                           then ret
                           else result
                      else ret
          where result = bt_recursion (assign assignment var val) csp
                ret = find_consistent_value vs
```

The BACKTRACK algorithm

```
bt :: CSP -> (Maybe Assignment, Int)
bt csp = bt_recursion [] csp

bt_recursion :: Assignment -> CSP -> (Maybe Assignment, Int)
bt_recursion assignment csp =
  if (is_complete csp assignment) then (Just assignment, 0)
  else find_consistent_value $ domain_of csp var
    where var = get_unassigned_var csp assignment
          find_consistent_value vals =
            case vals of -- recursion over the possible values
                      -- instead of for-each loop
            [] -> (Nothing, 0)
            val:vs ->
              if (is_consistent_value csp assignment var val)
              then if (isNothing result)
                   then (ret, nodes+nodes'+1)
                   else (result, nodes+1)
              else (ret, nodes'+1)
            where (result, nodes) =
                  bt_recursion (assign assignment var val) csp
                  (ret, nodes') = find_consistent_value vs
```

Unit Testing!
