Lecture Notes Example

Some CPSC Course

Peyton Seigo

2018-10-19

Lecture 19

Clicker questions

- **Functions operating on arbitrary sized data** consume arbitrary sized data, have a 2 case cond, and have a natural recursive call
- **A backtrack search** has several defining properties:
  - Produce `false` if the key was not found (base case)
  - `(if (not (false` in else
  - If found the entry, return that entry (for now, we have to traverse down the tree again to retrieve its value)
  - Calls itself

Two One-Of

- Cross product tables tell us the **minimum** number of `check-expects` to write
- Cross product causes us to create a new template
  - `template` tag is just `(@template 2-one-of)`
- `cond` template questions are based on the cross product axes
- **On problem sets**, you must show your cross product table in a comment box
- When designing cross tables, ask
  - What data do I have access to?
  - What data do I need?
  - What data do I need to produce?

Problem 1

lon1 (right) lon2 (down)	**empty**	**(cons Number ListOfNumber)**
empty	(1) `true`	(2) `false`
(cons Number ListOfString)	(1) `true`	(3) keep/discard first if `(first lon1) == (first lon2)`. natural recursion on `lon1` and `lon2`.

Template

```
(@template 2-one-of)
(define (contains? lon1 lon2)
  (cond [(empty? lon1) (...)]
        [(empty? lon2) (...)]
        [else
         (... (first lon1)
              (rest lon1)
              (contains? (first lon2)
                         (rest lon2)))]))
```

Creating the cond

- For the third case, we have access to the first and rest of `lon1` and `lon2`
- Don't forget to wrap references in their appropriate functions and to create a natural recursive call, if applicable
- If the first in `lon1` equals the first in `lon2`, then call the natural recursion on the rest of each list
- Otherwise, if the first of `lon2` does not match, keep the first in `lon1` and compare it to the rest of `lon2`
  - We must find a match in `lon2` before discarding `(first lon1)`!

Problem 2

bt (right) p (down)	**`false`**	**`(make-node Natural String BinaryTree BinaryTree)`**
`empty`	(1) `false`	(2) `true`
`(cons "L" Path)`	(1) `false`	(3) natural recursion on left node
`(cons "R" Path)`	(1) `false`	(4) natural recursion on right node

- First case: all `false` cells
- Second case: the one `true` cell
- Third case: recursion on left node
  - Information: first p, rest p, key val, l, r, result of recursion on l, result of recursion on r
  - Information we care about: first p, rest p, l, result of recursion on l
- Fourth case: recursion on right node