Review-3

Amer Diwan

Overview

• Wish list:
  – Continuations: an example
  – Closures
• Time for general questions
The continuation example

- Scenario: Would like to search a game tree for a winning position
- Let’s ignore “cycles” for this example--but it is not hard to extend this for cycles
- Let’s say a position with a “0” value is a winning value

An example tree

BNode(10, BNode(5, Leaf(7), Leaf(3)), BNode(6, Leaf(0), Leaf(1)))

A typical algorithm: 10, 5, 7, (backtrack), 5, 3, (backtrack), 5, (backtrack), 10, 6, 0, (success!)
A straightforward implementation

- Each call to traverse returns a success or failure
- If it returns success, then don’t search remaining branch--simply return success
- If it returns failure, then search remaining branches
- If no outgoing branches that haven’t been searched, return failure

Problems with straightforward implementation

- Annoying codes to check and return
- “Success” needs to be propagated through all the levels of recursion to the top level
- “Failure” requires backtracking some number of levels up
- Can be made to work with exceptions, but let’s use continuations instead
Try 1

- datatype 'a Tree =
  Leaf of 'a | BNode of 'a * 'a Tree * 'a Tree
- fun travl (Leaf(v)) f sk = if f(v) then throw sk() else ()
  | travl (BNode(v, l, r)) f sk = if f(v) then throw sk() else (travl l f sk; travl r f sk)

- How does this work?
- What happens if “success” is never found?
- How about implementing backtracking using continuations?

Try 2

- fun travl (Leaf(v)) f bk sk = if f(v) then throw sk() else throw bk()
  | travl (BNode(v, l, r)) f bk sk = if f(v) then throw sk() else (callcc(fn k => travl l f k sk);
  | travl r f bk sk)

- Two continuations are passed now:
  - to indicate success
  - to mark a place to backtrack
Rest of the code

- `fun eq0 v = v = 0;`
- `fun useit atree =`
  ```
  (callcc(fn k => trav atree eq0 k); print "found!\n")
  ```
- `fun trav t f succk =`
  ```
  (callcc(fn k => travl t f k succk); print "Not ")
  ```

What really does the backtrack-continuation represent?

```
print "Not"  
```
```text
5
10 57
57
57
7
7
5
3
0
1
6
```
Closures

• What is a closure
  – A code x environment pair
  – What does the “code” represent?
  – What does the “environment” represent?
• Creating closures
  – When does a programmer need to explicitly create a closure?
  – When does a programming language implicitly create a closure?

Any questions?
The end

- Regular office hours until final
- Thank you!