Lazy Functional Logic Programming and Encapsulated Search

Sebastian Fischer
Not my own Work

Encapsulating Nondeterminism in Functional Logic Computations
Brassel, Hanns, Huch 2004

Computing with Subspaces
Antoy, Brassel 2007

Set Functions for Functional Logic Programming
Antoy, Hanns 2009
The Curry Language

- first-class nondeterminism
- call-by-need (laz) semantics
- encapsulated search
First-class Nondeterminism

```
coin :: Int
coin = 0 ? 1
```

```
c> coin
0
More? yes
1
More? yes
No more results
```
Example: Permutations

\[
\text{perm} :: [a] \rightarrow [a]
\]
\[
\text{perm } l =
\]
\[
\begin{align*}
\text{if null } l & \text{ then } l \\
\text{else insert (head } l) (\text{perm } (\text{tail } l))
\end{align*}
\]

\[
\text{insert} :: a \rightarrow [a] \rightarrow [a]
\]
\[
\begin{align*}
\text{insert } x xs & = (x:xs) \ ? \\
\text{if null } xs & \text{ then fail } \\
\text{else head } xs : \text{insert } x (\text{tail } xs)
\end{align*}
\]
Example: Permutations

\texttt{cyi> perm [1,2,3]}

\[
\begin{array}{l}
[1,2,3] \\
[1,3,2] \\
[2,1,3] \\
[2,3,1] \\
[3,1,2] \\
[3,2,1] \\
\end{array}
\]

More? all
First-class Non-determinism

expressions can have multiple values

interactive environment for examining them
Infinitely Many Results

zeros :: Int
zeros = 0 ? zeros

>    zeros
0
More? yes
0
More? yes
0
More? no
Infinite Values

coins :: [Int]
coins = coin : coins

ghci> coins
^C
ghci> head coins
0
More? all
1
Laziness

\[
\text{isSorted} :: \text{[Int]} \rightarrow \text{Bool}
\]
\[
isSorted \ e = \\
\text{if null } e \ \text{|| null (tail } e) \ \text{then True} \\
\text{else head } e \ \text{<= head (tail } e) \\
\text{&& isSorted (tail } e)
\]

\[
\text{cyi > isSorted [0, -1..]} \\
\text{False}
\]
Lazy Nondeterminism

\[
\text{permsort} :: [\text{Int}] \rightarrow [\text{Int}]
\]
\[
\text{permsort } l =
\]
\[
\text{let } p = \text{perm } l \text{ in }
\]
\[
\text{if } \text{isSorted } p \text{ then } p \text{ else fail}
\]

same value

shared variable
Lazy Nondeterminism

\[
\text{cyi> let } x = \text{coin in } x + x
\]

0
More? yes
2
More? yes
No more results

\[
\begin{align*}
0 + 0 &= 0 \\
0 + 1 &= 1 \\
1 + 0 &= 1 \\
1 + 1 &= 2
\end{align*}
\]
Latiness

- infinitely many results
- infinite (intermediate) values
- evaluation order independence

\[
\text{let } x = a \text{? } b \text{ in } e \\
\uparrow
\]

\[
\text{(let } x = a \text{ in } e) \text{? (let } x = b \text{ in } e)
\]
Encapsulated Search

primitive operation

values :: a → [a]

idea:
- values coin = [0, 1]
- head (values zeros) = 0
- values (a ? b) ≠ values a ? values b
Encapsulated Search

1. Weak Encapsulation
2. Strong Encapsulation
3. Set Functions
Weak Encapsulation

may be nondeterministic

values coin = [0, 1]

but

let x = coin in values x = [0] ? [1]

because x is introduced "outside" of encapsulated expression
Weak Encapsulation

sharing between "inside" and "outside"

\[
\begin{align*}
\text{let } x &= \text{coin in} \\
\text{values } x &+ (x : \text{values } x) \\
\downarrow \\
[0, 0, 0] &? [1, 1, 1]
\end{align*}
\]
N-Queens Problem

\[
\begin{bmatrix}
3 & 1 & 4 & 2 \\
\end{bmatrix}
\]
N-Queens Problem

queens :: Int -> [Int]
queens n =
    let p = perm [1..n] in
    if null (values (capture p)) then p
    else fail

capture :: [Int] -> ()
capture (_++ x:xs++y:_) =
    if abs (x-y) == length xs + 1 then () else fail
Weak Encapsulation

different results based on syntactic difference (scope)

not all choices encapsulated
Strong Encapsulation

encapsulates all choices

let x = coin in values x

although x is introduced "outside" of encapsulated expression
Strong Encapsulation is Reusable

`hasValue :: a -> Bool
hasValue x = not (null (values x))`

`firstValue :: a -> a
firstValue x = head (values x)`

note: x is introduced "outside"
Strong Encapsulation

sharing between "inside" and "outside"

let x = coin in values x ++ (x:values x)

is not [0,1,0,0,1] ? [0,1,1,1,0,1]

but [0,1,0,1,0] ? [0,1,1,1,1,1]

(in all implementations of strong encapsulation)
Strong Encapsulation

result depends on evaluation order

let \( x = \text{coin} \) in values \( x \times (x: \text{values}(x)) \)

if \( x \) not yet evaluated

values \( x = [0, x] \)

if \( x \) already evaluated to \( 0 \) (or \( x \))

values \( x = [0] \) (or \( [x] \))
N - Queens

\[ \text{queens} :: \text{Int} \rightarrow [\text{Int}] \]

\[ \text{queens } n = \]
\[ \quad \text{let } p = \text{perm } [1..n] \text{ in} \]
\[ \quad \text{if } p \neq p \land \text{null values (capture } p) \]
\[ \quad \text{then } p \]
\[ \quad \text{else fail} \quad \text{force evaluation of } p \]
Strong Encapsulation

- encapsulates all choices
- is reusable
- no evaluation-order independent implementation exists
Set Functions

No primitive values :: a → [a]

Instead: set-valued variant of every defined function
Set Functions

\[
\text{addCoin} :: \text{Int} \to \text{Int}
\]
\[
\text{addCoin } x = x + \text{coin}
\]

\[\text{generates}\]

\[
\text{addCoin}_{\text{set}} :: \text{Int} \to [\text{Int}]
\]

\[\text{conceptually: set, not list}\]
Set Functions

e encapsulate choices in body, but not in arguments

\[
\text{addCoin}_{\text{set}}(10 \lor 20)
\]

\[
[10, M] \lor [20, 21]
\]

choice between 0 and 1

choice between 10 and 20
N - Queens

\[
\text{queens} :: \text{nat} \rightarrow [\text{nat}]
\]
\[
\text{queens } n =
\]
\[
\text{let } p = \text{perm } [1 .. n] \text{ in }
\]
\[
\text{if } \text{null } (\text{capture}_{\text{set}} p) \text{ then } p \backslash
\]
\[
\text{else fail}
\]
Set Functions

similar to weak encapsulation

but separation of choices
based on argument-body distinction
rather than on scoping
Laxy Functional Logic Programming

- first-class nondeterminism
- evaluation-order independent call-by-need semantics
- interactive environment for examining results
- encapsulated search
Weak Encapsulation
depends on scoping
not reusable

Strong Encapsulation
reusable
depends on evaluation order
(at least as implemented for Curry)

Set Functions
evaluation-order independent
no (reusable) strong encapsulation
Delimited Continuations?

fail = []

\( x ? y = \text{shift } k \cdot k x + k y \)

reset: weak or strong encapsulation?
evaluation order independence?
ありがとうございました