CS342: Organization of Prog. Languages

Topic 8: Functional Programming Gymnastics

- Curry
- Uncurry
- Partial application
- Twist
- Compose
- Map
- Reduce
- Left Associative Reduce
- Spread
- Filter
- Functional Programming – Summary

Curry

Functional programming has a number of patterns, which show up frequently no matter what the language.

One of the most basic of these is is the notion of the “curried” function (after the logician Haskell Curry).

- Use the isomorphism between the function spaces

\[(A \times B) \rightarrow C \quad \text{and} \quad A \rightarrow (B \rightarrow C)\]
• *Example:*

```
(define add (lambda (a) (lambda (b) (+ a b))))
```

• We can introduce an operation to do this for us:

```
(define curry (lambda (f)
    (lambda (a) (lambda (b) (f a b)))))

(define times (curry *))

((times 3) 4)
```

• Currying can be used with functions with several arguments:

\[
(A_1 \times \cdots \times A_n) \rightarrow R
\]

\[
A_1 \rightarrow \cdots \rightarrow A_n \rightarrow R
\]
Uncurry

- Use the function isomorphism in the other direction.

\[
\text{(define uncurry (lambda (f)} \\
\text{\quad (lambda (a b) ((f a) b))))}
\]
Partial application

- Apply a curried function to only some of its arguments.

- *Example:*

  (define double ((curry *) 2))

  (double 7)
Twist

- Change the order of arguments:
  
  \[ A \to B \to C \quad \text{becomes} \quad B \to A \to C \]
  
  \[ A \times B \to C \quad \text{becomes} \quad B \times A \to C \]

- This can be implemented as:

  (define twist (lambda (f)
      (lambda (b) (lambda (a) ((f a) b)))))

  (define twist2 (lambda (f)
      (lambda (b a) (f a b))))

- Example:

  (define half ((twist (curry /)) 2))
  (define half ((curry (twist2 /)) 2))
Compose

- Functional composition:

\[
\begin{align*}
\text{(define compose (lambda (f) (lambda (g))}
\quad \text{(lambda (a) (f (g a))))))
\end{align*}
\]

\[
\begin{align*}
\text{(define compose2 (lambda (f g)}
\quad \text{(lambda (a) (f (g a)))))
\end{align*}
\]

- E.g.

\[
\begin{align*}
\text{(define divby (((compose2 twist curry) /))}
\text{(define divby (((compose twist) curry) /))}
\text{(define third (divby 3))}
\end{align*}
\]
Map

- Apply a unary function to all elements of a list.

- Example:

  (map half (list 1 2 3 4))

- This is called map in Scheme and mapcar in various Lisp-s
Reduce

• Convert a binary operation to an N-ary function by applying it from left to right on values in a list.

• This can be defined as:
  ;; (reduce f (list a b c d)) -> (f a (f b (f c d)))
  (define reduce (lambda (f l)
       (if (null? (cdr l))
         (car l)
         (f (car l) (reduce f (cdr l))))

• E.g.
  (reduce + '(1 2 3 4)) ; gives 10
  (reduce - '(1 2 3 4)) ; gives -2
  (reduce (twist2 cons) '(1 2 3 4)) ; gives (((4 . 3) . 2) . 1)

• In APL this is written \( F / L \)
Left Associative Reduce

- Associate binary function in the other direction,
  \((f \ldots (f (f (f a b) c) d) \ldots z)\)
- Example: \((- (- (- (- 1 2) 3) 4) 5)\)
- An implementation

```scheme
(define (lareduce f l)
  ; Tail recursive formulation
  (define (red x f l)
    (if (null? l) x (red (f x (car l)) f (cdr l)))
  )

  (red (car l) f (cdr l))
)

; Another, elegant version
(define (lareduce f l)
  (reduce (twist2 f) (reverse l))
)
```
Spread

- Apply several functions to the same value.
- *E.g.*

  (spread (list half double third) 9)
Filter

- Retain only certain elements from a list.
- Specify which elements with a boolean-valued function.
- *E.g.*

  (filter even? (list 1 2 3 4))

  (filter pair? (list ’() 7 ’(a . b) (list 2 3 4) 2))
Functional Programming – Summary

- Construct new values from old, rather than updating values.
- Construct new functions from old, rather than writing loops.
- Define names once, rather than updating with assignments.

- Easier to prove facts about programs.
- Easier to parallelize programs.