

Map: More on functional abstraction and polymorphism





Quiz Feedback

- *“Why are we working on the abstraction from templates, when it's a theoretical aspect of programming?”*
- *“I don't understand ‘simple use’.”*
 - Note 2 said: **Simple** means that no part of the definition (including auxiliary functions) may use the template for lists.



Functional Abstraction

- A powerful tool
 - Makes programs more concise
 - Avoids redundancy
 - Promotes “single point of control”
- Combined with polymorphic contracts, it is even more powerful
- What we cover today for lists applies to *any* type that you can define



Look for the pattern

- One function:

```
; my-fun1 : [number] -> [number]
```

```
; adds one to each number in list
```

```
(define (my-fun1 l)
```

```
  (cond [(empty? l) empty]
```

```
        [else (cons (add1 (first l))
```

```
                    (my-fun1 (rest l)))]))
```



Look for the pattern

- Another function function:

```
; my-fun2 : [boolean] -> [boolean]
```

```
; inverts each boolean in the list
```

```
(define (my-fun2 l)
```

```
  (cond [(empty? l) empty]
```

```
        [else (cons (not (first l))
```

```
                    (my-fun2 (rest l)))]))
```



Codify the pattern

- Another function function:

; `map` : $(X \rightarrow X), [X] \rightarrow [X]$

; applies `f` to each element in `l`

```
(define (map f l)
```

```
  (cond [(empty? l) empty]
```

```
        [else (cons (f (first l))
```

```
                    (map f (rest l))))])
```



Generalize the pattern

- Another function function:

; `map` : $(X \rightarrow Y), [X] \rightarrow [Y]$

; applies `f` to each element in `l`

```
(define (map f l)
```

```
  (cond [(empty? l) empty]
```

```
        [else (cons (f (first l))
```

```
                    (map f (rest l))))])
```



Tip on Generalizing Types

- When we generalize, we **only ever** replace
 - specific types (like `number` or `symbol`)
 - by variables (like `X` or `Y`)
- We **never** replace a type by the “any” type, which actually means
 - `number` or `boolean` or `list of number` or `list of`



Use the pattern

- `(map add1 1)`
- `(map not 1)`
- `(map sqr 1)`
- `(map length 1)`
- `(map first 1)`
- `(map symbol? 1)`
- Note: Other data types also have maps!



More about Map

- Powerful tool for parallel computing!
- Has elegant properties:
 - $(\text{map } f (\text{map } g l)) = (\text{map } (\text{both } f g) l)$
 - Soon we will see how to define “both”
- For fun: Checkout Google’s “map/reduce”



Templates as functions

- Recall the template for lists:

```
; (define (fun-for-l l)
;   (cond
;     [(empty? l) ...]
;     [else ... (first l)
;              ... (fun-for-l (rest l))
;              ... ]))
```
- Can we pass the "... "s as parameters?



Templates as functions

- It would look just like this:

```
(define (fun-for-l p1 p2 l)
  (cond [(empty? l) p1]
        [else (p2 (first l)
                    (fun-for-l p1 p2
                              (rest l)))]))
```

- Can we express all functions we've written using `fun-for-l`?



Map in terms of Fold (in class)

```
map : (X->Y), [X] -> [Y]
(define (map f l)
  (local (define (g x l)
            // g : X [Y] -> [Y]
            (cons (f x) l))
    (fun-for-l empty g l)))
```



Templates as functions

- It would look just like this:

```
(define (fun-for-l p1 p2 l)
  (cond
    [(empty? l) p1]
    [else (p2 (first l)
               (fun-for-l p1 p2
                           (rest l))) ]))
```

- What's the MGT for “fun-for-l”?
 - ???



In class

Here is a step by step derivation

- $X \ Y \ Z \ \rightarrow \ A$
- $X \ Y \ [Z] \ \rightarrow \ A$
- $X \ Y \ [Z] \ \rightarrow \ X$
- $X \ (Z \ X \ \rightarrow \ X) \ [Z] \ \rightarrow \ X$



Templates as functions

- It would look just like this:

```
(define (fun-for-l p1 p2 l)
  (cond
    [(empty? l) p1]
    [else (p2 (first l)
               (fun-for-l p1 p2
                           (rest l))) ]))
```

- What's the MGT for “fun-for-l”?
 - $X (Z X \rightarrow X) [Z] \rightarrow X$



More about this function (“fold”)

- What other types can it be defined for?
- How does it help us in programming?
- Can it help in parallel computing?



For Next Class

- Homework due Wednesday
- Reading:
 - Ch 22: Designing with first class functions
- Quiz on reading