

CS 135 — L18 - Lambda

Luke Lu • 2025-12-17

Anonymous Function

Info — Anonymous Function

1. Anonymous Function is used for defining a function/predicate

```
(local [(define (f a ... z) (expression))] f)
```

2. Lambda Function Equivalence is used for defining the same function/predicate

```
(lambda (a ... z) (expression))
```

Example usage with filter:

```
(filter (lambda (x) (expression)) lst)
```

Beta(β) Reduction

Info — β Reduction

$$\lambda(x_1 x_2 \dots x_n) v_1 v_2 \dots v_n(e) \implies e'$$

where e' is e with all occurrences of x_n replaced by v_n

Example:

```
(lambda (x y) (+ x y) 3 4) => (+ 3 4) => 7
```

Stable Sorting

Simplifying merge

We pass in another predicate `<?` for generalizing for all inputs rather than only numbers.

```
(define (merge <? lst1 lst2)
  (cond
    [(empty? lst1) lst2]
    [(empty? lst2) lst1]
    [(<? (first lst1) (first lst2)) (cons (first lst1) (merge <? (rest lst1) lst2))]
    [else (cons (first lst2) (merge <? lst1 (rest lst2)))]
  )
)
```

Merge Sort

Splitting Function

```
(define (split lst)
  (local
    [(define n (quotient (length lst) 2))]
    (list (first-n n lst) (rest-n n lst))
  )
)
```

Creating sorting function

```
;; produces a sort function given a predicate <?
;; make-sort: (X X -> Bool) -> ((listof X) -> (listof X))
(define (make-sort <?)
  (local
    [(define (mergesort lst)
      (cond
        [(or (empty? lst) (empty? (rest lst))) lst]
        [else
         (local [(define s (split lst))] (merge <? (mergesort (second s)) (mergesort
          (first s))))])
      mergesort
    ])
  )
)
```

Sort

Stable Sorting

A “stable” sorting algorithm is one that preserves the relative order of element that are considered equal, i.e. ($<? x y$) and ($<? y x$) are both false.

This is not a stable sort when it $<?$ is in any form $<$ in this case. This might not be true for all of them.

```
(define (insert n lst <?)
  (cond
    [(empty? lst) (cons n empty)]
    [(<? n (first lst)) (cons n lst)]
    [else (cons (first lst) (insert n (rest lst) <?))]))

(define (sorts lst <?)
  (cond [(empty? lst) empty]
        [else (insert (first lst) (sorts (rest lst) <?) <?)]))
```

Info — sort

sort is a built-in function for sorting.

It is a stable sort

```
;; sort: (listof X) (X X -> Bool) -> (listof X)
;; example usage:
```

```
(sort (list 1 3 3124 32132433423 99 0) <)
;;gives
(list 0 1 3 99 3123 32132433423)
```