CSC 372, Spring 2025

High-order functions and scope

Michelle Strout

THE UNIVERSITY OF ARIZONA

February 11, 2025

Plan

- **Announcements**
  - SA3 MT1 review is posted and due Wednesday Feb 12th
  - Simon Peyton Jones joining us via zoom on Thursday Feb 13th
  - LA1 parser/shapes2svg is posted and due Friday Feb 14th
- **Last time**
  - TopHat Questions from ICA4 and ICA5
  - Recursive descent parsing
  - Polymorphism in SML
- **Today**
  - Questions for Simon
  - High order functions
  - Moved scope to last third of class when covering Chapel

TopHat Questions

THE UNIVERSITY OF ARIZONA
Computer Science

- 10 minutes of a Simon video, https://simon.peytonjones.org/darwin-codes/, starting at game of life 28:15 to 38
- Questions to ask Simon
- Some terminology Simon uses in assigned podcast

Outline for rest of today

- Polymorphic Types in SML
- More SML info
- High-order functions in SML

Different kinds of Polymorphism

- **Ad-hoc Polymorphism (Overloading & Coercion)**
  – Function/operator overloading (e.g., + for ints and reals).
  – Implicit type conversions (e.g., int to float in C/C++).

- **Parametric Polymorphism**
  – Functions and data structures operate on any type.
  – Example: fun identity x = x in SML (‘a -> ‘a)
  – Implemented as generics in languages like Java and Rust.

- **Subtype Polymorphism (Inheritance and Subtyping)**
  – Objects of a class can be used where a superclass is expected.
  – Enables method overriding and dynamic dispatch.
  – Example: Animal superclass, Dog subclass.

Pattern Matching with Polymorphism

```
datatype `a tree = Child
                 | Parent of `a * `a tree * `a tree

fun inOrder Child = []
   | inOrder
```

- **Questions**
  - Finish the rest of the above?
  - How is polymorphism different than overloading?

Outline for rest of today

- Polymorphic Types in SML
- More SML info
- High-order functions in SML

Tuple Pattern Matching

- **Question**
  - Indicate what the variables in the pattern will be bound to.

```
val (x,y) = (1,2) (* answer: x=1, y=2 *)

val (n,xs) = (3,[1,2,3]) (* answer: n=3, xs=[1,2,3] *)

val (x::xs) = [1,2,3]  (* answer: x=1, xs=[2,3] *)

val (_::xs) = [1,2,3]  (* answer: xs=[2,3] *)

val (_::xs) = [3]  (* answer: xs=[] *)
```

Case Expressions also use pattern matching

- **`case`** expression

```
fun length xs =
  case xs
    of []         => 0
     | (x::xs) => 1 + length xs
```

- At top level, **`fun`** is better than **`case`**

```
fun length []      = 0
  | length (x:xs) = 1 + length xs
```

Case works for any datatype

- At top level, **`fun`** is better than **`case`**

```
fun toStr t =
    case t
      of Leaf => "Leaf”
       | Node(v,left,right) => "Node"
```

- Question: how do we rewrite above case using fun?

```
fun toStr ??
```

ML traps and pitfalls

- Order of clauses matters

```
fun take n (x::xs) = x :: take (n-1) xs
  | take 0 xs = []
  | take n [] = []
(* what goes wrong? *)
```

- Gotcha – overloading

```
- fun plus x y = x + y;
> val plus = fn : int -> int -> int
- fun plus x y = x + y : real;
> val plus = fn : real -> real -> real
```

- Gotcha – equality types, `''a` is equality type var

```
- (fn (x,y) => x=y);
> val ''a it = fn : ''a * ''a -> bool
```

Outline for rest of today

- Polymorphic Types in SML
- More SML info
- High-order functions in SML

Higher-Order Functions

- Goal: start with functions on elements, end up with functions on lists
  - Generalizes to sets,
  - arrays,
  - search trees,
  - hash tables, ...

- Goal: Capture common patterns of computation or algorithms
  - `exists` (example: is there a number?)
  - `all` (example: is everything a number?)
  - `filter` (example: take only the numbers)
  - `map` (example: add 1 to every element)
  - `foldr` (general: can do all of the above and more)

List search: `exists`

- Algorithm encapsulated: linear search
- Example: Is there an even element in the list?

```
fun exists p [] = ???

fun exists p (x::xs) = ???

(* What are some example calls to exists? *)
```

List search: `all`

- Algorithm encapsulated: linear checking
- Example: Is every element in the list even?

```
fun all p [] = ???

fun all p (x::xs) = ???

(* What are some example calls to all? *)
```

List search: `filter`

- Algorithm encapsulated: linear filtering
- Example: Given a list of numbers, return only the even ones

```
fun filter p [] = ???

fun filter p (x::xs) = ???

(* What are some example calls to filter? *)
```

- What are the restrictions on `p` for `exists`, `all`, and `filter`?

Defining `filter`


```
-> fun filter p [] = []
   | filter p (x::xs) =
       if p x then x :: filter p xs
       else filter p xs;

-> val test1 = filter (fn n => n > 0) [1, 2, ~3,
~4, 5];
??

-> val test2 = filter (fn n => n <= 0) [1, 2,
~3, ~4, 5];
??
```

List search: `map`

- "Lifting" functions to lists
- Algorithm encapsulated: transform every element
- Example: square every number of a list

```
fun map f [] = ???

fun map f (x::xs) = ???
```

Defining `map`

THE UNIVERSITY OF ARIZONA
Computer Science

```
-> fun map f [] = []
   | map f (x::xs) = f x :: map f xs;

-> val test1 = map (fn n => n*n) [1, 2, ~3, ~4, 
5];
??

-> val test2 = map (fn n => n mod 2) [1, 2, ~3, 
~4, 5];
??
```

`foldr`: the universal list function

- **`foldr` takes two arguments**
  - `plus`: how to combine elements with running results
  - `zero`: what to do with the empty list

- **Example: foldr plus zero [a b]**

```
     a  ::     b   ::   []
     |         |        |
     v         v        v
   plus a (plus b zero)
```

The universal list function: fold

```
-> fun foldr plus zero [] = zero
   |   foldr plus zero (x::xs) =
         plus x (foldr plus zero xs);

-> val sum = foldr (fn x => fn y => x+y) 0 [1,
2, 3, 4];
??

(* How is this different than the builtin foldr?
*)
```

Studying for the midterm

- Implement each of the following using foldr
  - exists
  - all
  - filter
  - map

- Feel free to post possible answers on piazza