CSE341 – Section 3
Standard-Library Docs, First-Class Functions, & More
Agenda

1. SML Docs
   • Standard Basis

1. First-Class Functions
   • Anonymous
   • Style Points
   • Higher-Order

1. Examples
Standard Basis Documentation

Online Documentation
http://www.standardml.org/Basis/index.html

Helpful Subset
Top-Level http://www.standardml.org/Basis/top-level-chapter.html
List http://www.standardml.org/Basis/list.html
ListPair http://www.standardml.org/Basis/list-pair.html
Real http://www.standardml.org/Basis/real.html
String http://www.standardml.org/Basis/string.html
Anonymous Functions

An Anonymous Function

\texttt{fn \ pattern \ => \ expression}

• An expression that creates a new function with no name.
• Usually used as an argument to a higher-order function.
• Almost equivalent to the following:
\texttt{let fun \ name \ pattern \ = \ expression \ in \ name \ end}

• The difference is that anonymous functions cannot be recursive!!!
Anonymous Functions

What's the difference between the following two bindings?

```
val name = fn pattern => expression;
fun name pattern = expression;
```

- Once again, the difference is recursion.
- However, excluding recursion, a `fun` binding could just be syntactic sugar for a `val` binding and an anonymous function.
Unnecessary Function Wrapping

What's the difference between the following two expressions?

\[(\text{fn} \; \textit{xs} \; \rightarrow \; \text{tl} \; \textit{xs})\]  

\[\rightarrow\]

**STYLE POINTS!**

- Other than style, these two expressions result in the exact same thing.
- However, one creates an unnecessary function to wrap \(\text{tl}\).
- This is very similar to this style issue:

\[(\text{if} \; \textit{ex} \; \text{then} \; \text{true} \; \text{else} \; \text{false})\]  

\[\rightarrow\]
Higher-Order Functions

A function that returns a function or takes a function as an argument.

- **map**: (\( 'a \rightarrow 'b \)) \( \times 'a \) list \( \rightarrow 'b \) list
  - Applies a function to every element of a list and return a list of the resulting values.
  - Example: \( \text{map} \ (\text{fn} \ x \Rightarrow x*3, \ [1,2,3]) \) === \[3,6,9\]

- **filter**: (\( 'a \rightarrow \text{bool} \)) \( \times 'a \) list \( \rightarrow 'a \) list
  - Returns the list of elements from the original list that, when a predicate function is applied, result in true.
  - Example: \( \text{filter} \ (\text{fn} \ x \Rightarrow x>2, \ [-5,3,2,5]) \) === \[3,5\]

- **fold**: (\( 'a \times 'b \rightarrow 'a \)) \( \times 'a \) \( \times 'b \) list \( \rightarrow 'a \)
  - Initial accumulator: \( 'a \)
  - Returns \( f(\ldots f(f(\text{init}, x1), x2)\ldots xn) \) or \( \text{init} \) if the list is empty
  - Example: \( \text{fold} \ (\text{fn} \ (x, y) \Rightarrow x + y, \ 0, \ [1,2,3,4]) \) === 10
Broader Idea

Functions are Awesome!

- SML functions can be passed around like any other value.
- They can be passed as function arguments, returned, and even stored in data structures or variables.
- Functions like `map` are very pervasive in functional languages.
  - A function like `map` can even be written for other data structures such as trees.

(Let’s see some examples!)
Polymorphic Datatypes

(*Generic Binary Tree Type *)

```ml
datatype 'a tree = Empty
            | Node of 'a * 'a tree * 'a tree
```

(* Apply a function to each element in a tree. *)

```ml
val treeMap = fn : ('a -> 'b) * 'a tree -> 'b tree
```

(* Returns true iff the given predicate returns true when applied to each element in a tree. *)

```ml
val treeAll = fn : ('a -> bool) * 'a tree -> bool
```