CSC 372, Spring 2025

Intro to Standard ML

Michelle Strout

THE UNIVERSITY OF ARIZONA

January 21, 2025

Plan

- **Announcements**
  - SA1 is due tomorrow/Wednesday
- **Last time**
  - Intro, motivation, class logistics
  - ICA1: Pre-assessment quiz
  - **Question:** Is it possible to get the slides before class so that we can write notes on them during class?
- **Today**
  - Review some pre-assessment quiz questions
  - ICA2: Quiz on the syllabus
  - Functional programming and SML intro

TopHat Questions

- **ICA1: Pre-assessment quiz**
  - Go to gradescope to see how you did, see piazza announcement
  - The class median on the 10 questions was 5/10
- **Some pre-assessment questions**
  - Example SML function 'foo' question, got 29 question marks
  - Java 'instanceof' keyword question
- **Link languages to motivation in TopHat**
  - Kotlin
  - C
  - JavaScript
  - Prolog
  - Chapel

ICA2: Quiz on the Syllabus

- **Read the instructions on the quiz**

Functional Programming and SML Outline

- Motivation and History of SML
- Running an SML program (Hands On)
- Functional programming key concepts
- Functions and pattern matching in SML (Hands On)
- Recursion in SML (Hands On)
- Unit testing and exceptions in SML (Hands On)
- Other things to try

SML Overview

- **History**
  - Evolved from ML (Meta-Language, 1970s), which was designed for theorem proving in the Edinburgh LCF system
  - Standardization occurred in the 1980s and 1990s
  - Used in formal verification, writing compilers and interpreters, education
  - Family of languages: ML, Moscow ML, Ocaml, F#, Lazy ML, …
  - Major projects
    - IT University of Copenhagen's enterprise architecture–around 100,000 lines of SML
    - proof assistants HOL4, Isabelle, LEGO, and Twelf

Reference: https://en.wikipedia.org/wiki/Standard_ML#Major_projects_using_SML

SML Overview

- **Important Ideas**
  - (1) Pattern matching is big and important. You might really like it.
  - (2) Recursion instead of iteration
  - (3) Exceptions are easy
  - (4) Static types
  - (5) Functions as values and high-order functions

- **Can't crash.**
  - Can have an infinite loop
  - Can return errors

Writing and executing SML code

- **Steps**

  1. Go to Piazza and then syllabus on GitHub
  2. Git clone the course materias repository which has Sandboxes/
  3. Assume Docker desktop has already been installed (SA1)
  4. ‘cd’ into the repository in a terminal and in vscode terminal
  5. Start the docker container in the terminal
  6. Edit files in vscode (or favorite editor)

- **Code**

```
/workspace$ poly              // start sml interpreter
Poly/ML 5.7.1 Release
> 1234 + 16 ;                 (* expression *)
val it = 1250: int
> it*2;                       (* can use temp variable *)
(* try: "x = " ^ str #"@" ^ ", y = " ^ Int.toString 42 *)
```

Functional Programming Key Ideas

- **Immutability, referential transparency, pure functions**
  - Can't reassign to a variable
  - An expression can always be replaced by its value due to no hidden side effects: enables memorization, optimization, parallelization, …
  - Pure functions, no hidden side effects

- **Drawbacks of functional programming**
  - Can't usually access specific memory address in language
  - Can't really update values in place
  - Since allocating memory to store new values, usually need some kind of garbage collection

Functions in SML

- **Key Concepts**
  - One parameter but can use currying and tuples
  - Functions are first-class values (passed as arguments, returned, …)
  - Type inference used to determine function types without annotations

- **Code**

```
fun square x = x * x;

(* Define a simple curried function *)
fun add x y = x + y;

(* A higher-order function example *)
fun applyTwice f x = f (f x);
```

- **Questions**
  - What is the output of ‘applyTwice square 3’ and why?
  - Thursday preview: What are the types of square, add, and applyTwice?

Pattern Matching in SML

- **Key Concepts**
  - A concise and expressive way to destructure and analyze data.
  - Patterns can include wildcards '_', literal values, and nested patterns
  - Pattern matching is exhaustive, all cases must be covered

- **Code**

```
(* Pattern matching for natural numbers *)
fun factorial 0 = 1
  | factorial n = n * factorial (n - 1);

(* Pattern matching with list recursion *)
fun sumList [] = 0
  | sumList (x :: xs) = x + sumList xs;
```

- **Questions**
  - Are those parens needed around (x:: xs)? Why?

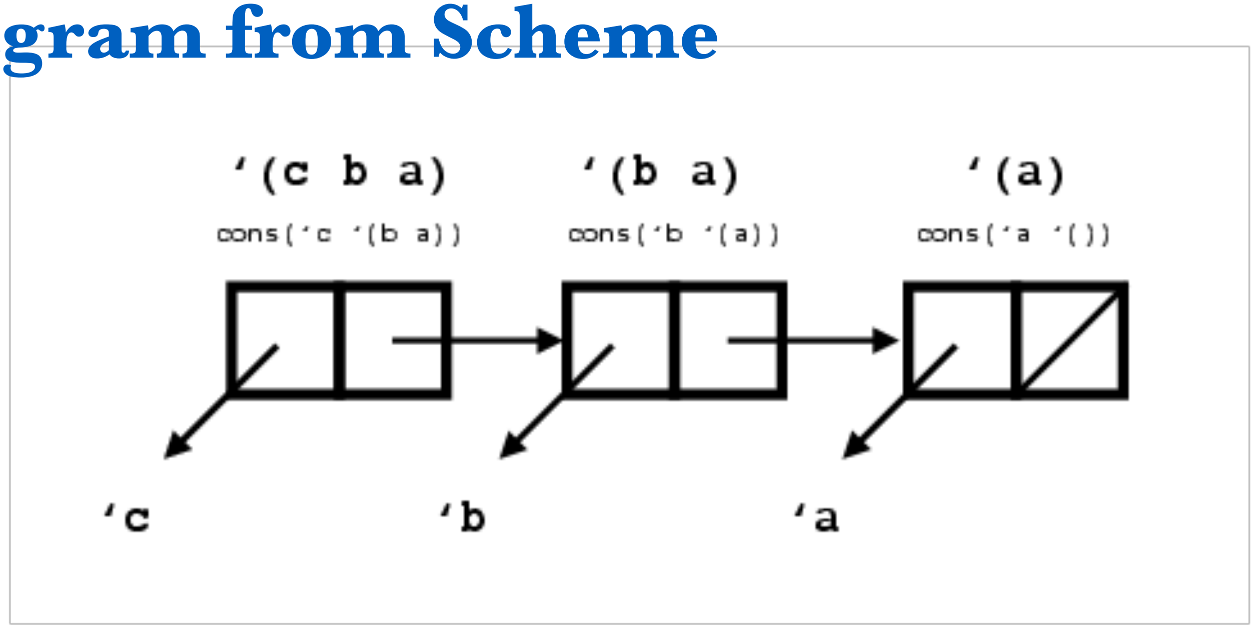
Use AI to experiment with SML

- **Ask an AI for example uses of…**
  - Char.ord
  - Reading in text from a file
  - List concatenation in SML
  - Appending an item to the beginning of a list
  - Appending an item to the end of a list
  - Pattern matching the second item of a list

- **Questions for you to answer (Anki candidates)**
  - How could you use Char.ord to determine if a character is an uppercase letter?
  - Why does appending an item to the end of a list in SML involve copying?
  - Is it possible to pattern match the last item in a list? Explain why or why not.

List Storage for SML

- **Diagram from Scheme**



- **[c, b, a] in SML is stored similarly**
  - It's a singly linked list

Recursion instead of iteration

- **No loops in SML, Recursion is used instead**
  - At least one base case will be needed
  - The recursive part of the function needs to make progress to avoid an infinite “loop”

- **Code: imitating a loop with recursion**

```
fun loop i n =
    if i > n then ()   (* Base case *)
    else (
        print (Int.toString i ^ "\n");
        loop (i + 1) n (* Recurse: increment i *)
    );
```

What is wrong with recursion in these?

```
fun removeNegatives (x :: xs) =
      let
          val result = removeNegatives xs
      in
          if x < 0 then result
          else x :: result
      end;
```

```
fun decrList [] = []
  | decrList (x :: xs) = (x-1) :: decrList xs;
```

Testing your SML functions

- **See sml-intro-in-class.sml**
  - Uncomment the ‘use “Unit.sml”; ‘ code
  - Can check expected results and if an exception has occurred

- **Code: Some example usage**

```
val () =
  Unit.checkExnWith Int.toString
  "minlist [] should raise an exception"
  (fn () => minlist [])

val () =
  Unit.checkExpectWith
  (Unit.listString (Unit.pairString Int.toString Int.toString))
  "zip ([],[]) should be []"
  (fn () => zip ([],[]))
  []
```

Exceptions Example in SML

- **Code**

```
fun drop 0 l = l
  | drop n [] = raise ListTooShort
  | drop n (x::xs) = drop (n-1) xs

val res7 = drop 2 [1,2,3,4]
val res8 = drop 3 [10, 20, 30]
           handle ListTooShort =>
              (print "List too short!"; [])
```

- **Questions**
  - What is res7 going to be?
  - What is res8 going to be?

Other things to try

- **In the poly REPL, try the following:**

```
let x=3 and y=4 in x+y end;  (* poly REPL balks *)
real;
explode;
ord;
trunc;
floor;
ceil;
round
chr;
str;
(op +);
```

- **Questions for you to answer**
  - What do each of the above do?
  - Ask an AI how to fix the error you get for the first one.