Computer Science 131
Programming Languages

August 29, 2000
Introduction to SML
Expressions in SML

• Every expression
  - has a type
    • e.g., int, bool, string
  - may have a value
  - may cause side-effects
    • assignment, I/O, exception
Base Types and Values

- true    false       : bool
- 3      17      ~4     : int
- 3.14    2.17    6.02e~23 : real
- "hello world\n" : string
- #"a"    #"\n" : char
- ()      : unit
Aggregate Types and Values: Pairs

- (3, true) (~17, false) : int*bool
- ("pi", 3.14) : string*real
- (~17, 4) : int*int
Aggregate Types and Values: Tuples

• (4, "cs131", ~18) : int*string*int
  - and so on, for as many components as you want

• Compare
  - int * string * int
  - (int * string) * int
  - int * (string * int)
Aggregate Types and Values: Records

- \{x=3, y=4\} : \{x:int, y:int\}
- \{y=4, x=3\} : \{x:int, y:int\}
- \{x=3, y=4\} : \{y:int, x:int\}

- \{size=7.3, color="magenta", weight=126.43\} : \{size : real, color : string\}
Aggregate Types and Values: Lists

- `nil 2::nil` : int list
- `1::(2::nil)` : int list
- `[] [1,2]` : int list
- `[true,true,false]` : bool list
- `[[1,2],[3,4,5]]` : (int list) list
Simple expressions

- 3+4
  - Type? Value?
- 3.14 <= 2.17
  - Type? Value?
- (3-4, (5 mod 2, "no"))
  - Type? Value?
- if (7<2) then "yes" else "no"
  - Type? Value?
Variable Binding

• val x = 3 + 4
• val x' = ~x
• val s1 = "foo" ^ "bar"
• val lst = [1+2, 3+4]
• val long_variable_name = lst @ lst
Variable Binding

- val x = 3 + 4
- val x' = ~x
- val s1 = "foo" ^ "bar"
- val lst = [1+2, 3+4]
- val long_variable_name = lst @ lst
- val s1 = true
Function types and values

• fn (x:int):int => x+1 : int->int
Function types and values

- \( \text{fn (x:int):int} => x+1 \) : \( \text{int->int} \)

- \( \text{val succ = fn(x:int) => x+1} \)

- Now can evaluate \( \text{succ(3)} \) or \( \text{succ 3} \)

- But: \( \text{succ(3*2)} \) is not \( \text{(succ 3)*2} \)

- Careful of \( \text{succ 3 * 2} \)
Defining successor

• val succ = fn (x:int):int => x+1
• val succ = fn x => x+1

• fun succ (x:int) : int = x+1
• fun succ x = x+1
Recursion

• fun fact n = if (n=0) then 1 else n * fact (n-1)
Pattern Matching

- fun fact n = if (n=0) then 1 else n * fact (n-1)

- fun fact 0 = 1 |
  fact n = n * fact (n-1)
Multi-argument functions

- Every function takes exactly one argument
Multi-argument functions

• Every function takes exactly one argument
  - but this might be a pair or a record
Multi-argument functions

• Every function takes exactly one argument
  - but this might be a pair or a record

• `fun power(x, n) =`
  
  ```
  if (n = 0) then 1.0
  else x * power(x, n-1)
  ```
Multi-argument functions

• Every function takes exactly one argument
  - but this might be a pair or a record

• fun power(x,n) =
  if (n = 0) then 1.0
  else x * power(x,n-1)

• fun power(x,0) = 1.0
  | power(x,n) = x * power(x,n-1)
More pattern-matching

- val pair = (1+1, 2+2)
- val (x, y) = pair

- val (h::t) = [1, 2, 3]
Local variable bindings

let
    val x = 3
in
    x+1
end

• General form:
  let <definitions> in <expr> end
fun solve_quadratic(a,b,c) =
  let
  
  val disc = b*b - 4*a*c
  val sqrtdisc = Math.sqrt disc
  val denom = 2*a

  in

  ((~b + sqrtdisc) / denom, 
   (~b - sqrtdisc) / denom)

  end
Local variable bindings

val x = 3
val y = let
    val z = x + 1
    val x = 2 * z
    val w = z + x
in
    w + 1
end
Length of a list

- fun length ([] : int list) = 0
  | length (x::xs) = 1 + length xs
Length of a list

• fun length ([] : int list) = 0
  | length (_::xs) = 1 + length xs
Length of a list

• fun length [] = 0
  | length (_::xs) = 1 + length xs

• What is the type of length?