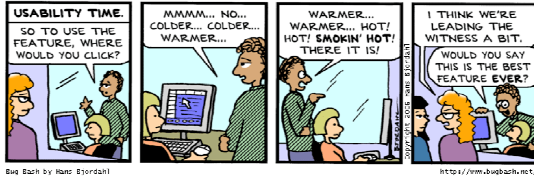


SML: Declarations, Functions, and Tuples

Prof. Evan Chang
Meeting 8, CSCI 3155, Fall 2009



Announcements

- Assignment 3 due tonight at 11:55pm
 - Submit in pairs
- Assignment 4 out tonight
 - Individual

2

Feedback on HW2. Thanks!

- Hard Stuff: Learning MYSTERY
- Clarifications
 - Typesetting not necessary; legible scans of hand-written write-up is ok, particularly for pictures (just put in 1 PDF)
 - Slides are posted online after class
 - Ask for clarifications on hw questions (in office hours, on moodle)
- To Improve
 - Notice for reading
 - Recitations
- Dislike
 - Working in pairs (will still have more pair assignments)
- Like: Tablet notes, discussion forums

3

Functional Programming Overview

Functional Programming

- You know OO and Imperative
- [Functional Programming](#)
 - Computation = evaluating (math) functions
 - Avoid "global state" and "mutable data"
 - Get stuff done = apply (higher-order) functions
- Important Features
 - Higher-order, first-class functions
 - Closures and recursion
 - Symbolic processing (lists, ASTs)

5

Functional-Style Advantages

- Tractable program semantics
 - Procedures are functions
 - Formulate and prove assertions about code
 - More readable
- [Referential transparency](#)
 - Replace any (sub)expression by its value without changing the result

6

How do functional languages give you referential transparency?

```
int x = 0;
int n() {
  x = x+1;
  return x;
}
```

$n()$ → 1
 $n()$ → 2

Not referentially transparent

Side effect

examples: I/O printing exceptions

W/o side-effects
⇒ "pure"

7

No side-effects

- Why is referential transparency useful?

For compiler - can choose how to compute it

- order evaluation
- replace w/ other functions the same result

8

Functional-Style Disadvantages?

- Higher-level (harder see hardware implications)
⇒ efficiency of execution
- NEW to YOU

9

Functional-Style Disadvantages

- Efficiency
 - Copying takes time (if needed)
- Compiler implementation
 - Frequent memory allocation
- Unfamiliar (to you!)
 - New programming style
- Not appropriate for every program
 - Operating systems, etc.

10

SML

Short History

- ML (Meta Language)
 - Edinburgh, 1973
 - Part of a theorem proving system LCF
- SML/NJ (Standard ML of New Jersey)
 - Bell Labs and Princeton, 1990
- OCaml (Objective Caml)
 - INRIA, 1996
- Same core ideas but some (annoying) syntactic differences

12

ML Innovative Features

static type checking - at compile time

- Type system
 - Strongly typed
 - Type inference
 - Abstraction
- Modules
- Patterns
- Polymorphism
- Higher-order functions *(pass around functions, take functions as arguments)*
- Concise formal semantics

weak/strong - providing some

There are many ways of trying to understand programs. People often rely too much on one way, which is called "debugging" and consists of running a partly-understood program to see if it does what you expected. Another way, which ML advocates, is to install some means of understanding in the very programs themselves.
- Robin Milner, 1997

13

A Small SML Program

```

(**) for comments (nestable)
(* A small SML program *)
val four = 4
fun plusfour y = y + four
val eight = plusfour four;
  
```

Value and function declarations

No type declarations needed!

; ends top-level expressions, not needed with just declarations

14

SML Top-Level

- Interactive loop: expressions can be typed and evaluated at the top-level

```

Standard ML of New Jersey v110.65 [built: Wed Oct 31 17:24:44 2007]
- 1;
val it = 1 : int
  
```

- Run a file by typing
use "file.sml";

15

Simple SML Examples: Basic Types

16

Typing and Evaluation Assertions

- Typing $e : \tau$ ("expr e has type τ ") - "tau"
 - $3 : \text{int}, (3 + 7) : \text{int}$
 - Annotate any expression in SML
- Justify the assertion $(3 + 7) : \text{int}$

Justify

```

plus(num(3), num(7)) : int
  ① num(3) : int (arbitrary)
  ② num(7) : int (arbitrary)
  ③ int both subexpr of plus so int
    then plus(-, -) : int
    ∴ (3+7) : int
  
```

17

Typing and Evaluation Assertions

- Evaluation $e \Downarrow v$ ("expr e evaluates to value v")
 - $3 \Downarrow 3, (3 + 7) \Downarrow 10$ *num(3) ↓ 3*
 - Justify $(3 + 7) \Downarrow 10$ similarly


```

plus(num(3), num(7)) ↓ 10
  ① num(3) ↓ 3 [num(n) ↓ n]
  ② num(7) ↓ 7
  ③ plus(num(3), num(7)) ↓ 10
     [if e1 ↓ v1, e2 ↓ v2
     then plus(e1, e2) ↓ v1+v2]
          
```

18

Type and Value Binding

- Type binding
`type float = real`
- Value binding
`val n : int = 5` (val) — giving that is fixed ("permanent")
- What's the difference with variable assignment?
`VAR n : INTEGER := 5` (MYSTERY)
`n := 6`

19

Limiting Scope

20

Typing and Evaluation with Bindings

- **Declarations** yield an environment
- An **environment** is a set of bindings (e.g., variable to values) used in subsequent declarations or expressions

$x + 3 : \text{int} ?$ $x + 3 \Downarrow 6?$
 $x : \text{int} \vdash x + 3 : \text{int}$ $x \Downarrow 3 \vdash x + 3 \Downarrow 6$

21

Functions abstract data from a computation

22

For Next Time

- Reading (a bit longer than usual)
- Online discussion forum
 - ≥ 1 substantive question, comment, or answer each week
- Homework assignment 4
 - Start early!

23