CS538 Lecture 11: Functional Programming and ML

March 1997

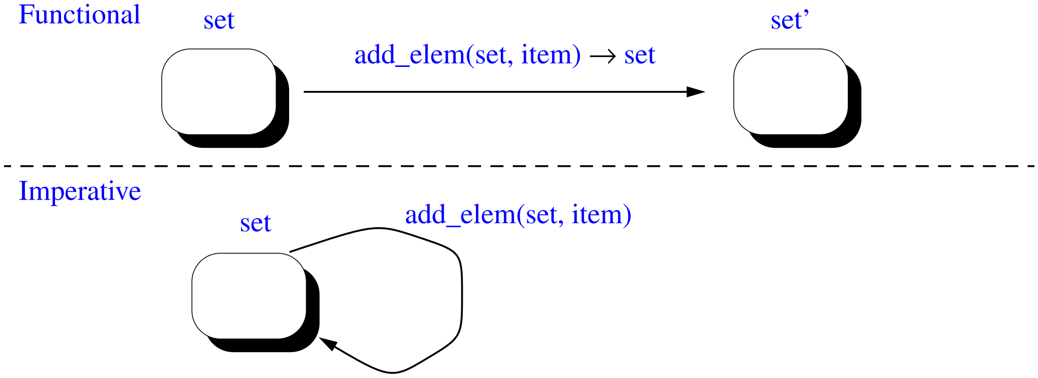
James R. Larus
Computer Sciences Department
University of Wisconsin–Madison
larus@cs.wisc.edu

Outline:

- Functional programming
- Overview of ML

Functional Programming

- Different approach to writing programs
- Program expressed as function from state → state
    - *State* is collection of values
- *Imperative* programs destructively modify state



Functional Example

- Simple functional

```
fun add_elem(s e) =
  if s = nil then e :: nil
  else if (hd(s) = e) then s
  else hd(s) :: add_element(tl(s), e);
```

cons

- With patterns

```
fun add_elem(nil, e) =
      e :: nil
|   add_elem(L as h :: t, e) =
      if h = e then L
      else h :: add_elem(t, e);
```

Pattern

Imperative Code

- More cases to handle

```
list* add_elem(list* s, item e) {
  if (s == NULL)
    return list(e, NULL);
  else if (s->hd == e)
    return s;
  else if (s->tl == NULL)
    {s->tl = list(e, NULL); return s;}
  else
    return add_elem(s->tl, e);
}
```

- Potential sharing problems

Functional Style Advantages

- Tractable program semantics
  - Procedures are functions
  - Formulate and prove assertions about code
  - More readable
- *Referential transparency*
  - Replace any expression by its value without affecting result
- No *side-effects*
  - Fewer errors

Functional Style Disadvantages

- Efficiency
  - Copying takes time
- Implementation requires additional effort
  - Frequent memory allocation
  - Good compiler
- Unfamiliar
  - New programming style
- Not appropriate for every program
  - Arrays
  - Operating systems, etc.

ML*

- Meta Language
  - Robin Milner et al. 1970's developed for proof systems
- Standard ML
  - Extended ~1983 with modules
  - SML/NJ
- Focus of programming language research
- Documentation
  - **`http://www.cs.wisc.edu/~cs538/ml.html`**
  - Jeffrey Ullman, *Elements of ML Programming*, Prentice Hall.
  - LC Paulson, *ML for the Working Programmer*, Cambridge.

*. Pratt & Zelkowitz, § 13.2, Sethi Chapter 9.

Innovative Features

- Type system
  - Strongly typed
  - Type inference
  - Abstraction
- Modules
- Patterns
- Polymorphism
- Higher-order functions
- Concise formal semantics
  - Milner, Tofte, Harper, *The Definition of Standard ML*, MIT Press.

Type System

- Type inference

```
fun add_elem(nil, e) =e :: nil
|   add_elem(L as h :: t, e) =
      if h = e then L
      else h :: add_elem(t, e);

val add_elem = fn : α list * α -> α list
```

- ML infers types
  - Inconsistent or incomplete type is an error
- Optional type declarations
  - Clarify ambiguous cases
  - Documentation

Modules

- Advanced module facility
  - Structures ~ modules
  - Signatures ~ interface
  - Functors ~ functions on modules (templates)

Patterns

- Patterns simplify code

```
fun add_elem(nil, e) = e :: nil
|   add_elem(L as h :: t, e) =
      if h = e then L
      else h :: add_elem(t, e);
```

Function pattern

Argument deconstructor

```
fun add_elem(s e) =
  if s = nil then e :: nil
  else if (hd(s) = e) then s
  else s :: add_element(tl(s), e);
```

  - Eliminates many conditionals and accessor functions
  - Not as general as Prolog patterns (no unification)

Polymorphism

- Functions and type inference are polymorphic
  - Operate on more than one type

```
fun len (nil) = 0
|   len(L as h :: t) = 1 + len(t);
val len = fn : α list -> int

len([1, 2, 3]);
val it = 3 : int

len([1.0, 2.0, 3.0]);
val it = 3 : int

len([1,1.0]);
std_in:19.5-19.11 Error: operator and operand don't agree
(tycon mismatch)
  operator domain: int * int list
  operand:         int * real list
  in expression:
    1 :: 1.0 :: nil
```

Higher-Order Functions

- Functions are first-class values
  - Use in place of other value
  - Closures capture environment

```
fun map(f, nil) = nil
|   map(f, h :: t) = f(h) :: map(f, t);
val map = fn : (α -> β) * α list -> β list

let val offset = 10
in map(fn (v) => offset + v, [1,2,4]) end;
val it = [11,12,14] : int list
```

- Extremely powerful programming technique
  - General iterators
  - Implement abstractions